

ADDRESSING TASKS THROUGH ROBOT ADAPTATION

Tarik Tosun

A DISSERTATION

in

Mechanical Engineering and Applied Mechanics

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2018

Mark Yim, Supervisor of Dissertation
Professor of Mechanical Engineering and Applied Mechanics

Kevin Turner, Graduate Group Chairperson
Professor of Mechanical Engineering and Applied Mechanics

Dissertation Committee

Mark Yim, Professor of Mechanical Engineering and Applied Mechanics
Cynthia Sung, Professor of Mechanical Engineering and Applied Mechanics
Daniel Koditschek, Professor of Electrical and Systems Engineering
Hadas Kress-Gazit, Professor of Mechanical and Aerospace Engineering, Cornell University

ADDRESSING TASKS THROUGH ROBOT ADAPTATION

© All Rights Reserved

2018

Tarik Daniel Tosun

To my family, Güray Tosun, Rebecca Tosun, and Leyla Tosun

ABSTRACT

ADDRESSING TASKS THROUGH ROBOT ADAPTATION

Tarik Tosun

Mark Yim

Developing flexible, broadly capable systems is essential for robots to move out of factories and into our daily lives, functioning as responsive agents that can handle whatever the world throws at them. This dissertation focuses on two kinds of robot adaptation. *Modular self-reconfigurable robots* (MSRR) adapt to the requirements of their task and environments by transforming themselves. By rearranging the connective structure of their component robot modules, these systems can assume different morphologies: for example, a cluster of modules might configure themselves into a car to maneuver on flat ground, a snake to climb stairs, or an arm to pick and place objects. Conversely, *environment augmentation* is a strategy in which the robot transforms its environment to meet its own needs, adding physical structures that allow it to overcome obstacles.

In both areas, the presented work includes elements of hardware design, algorithms, and integrated systems, with the common goal of establishing these methods of adaptation as viable strategies to address tasks. The research takes a systems-level view of robotics, placing particular emphasis on experimental validation in hardware.

Contents

Abstract	iv
Contents	v
List of Tables	ix
List of Figures	x
1 Introduction	1
1.1 Motivation	1
1.2 Reconfiguration	2
1.3 Environment Augmentation	4
2 Overview of Related Work	6
2.1 Modular Self-Reconfigurable Robots (MSRR)	6
2.1.1 MSRR Hardware Systems	6
2.1.2 Algorithms and Control	6
2.1.3 Tasks and Autonomy	7
2.2 Environment Augmentation	8
I Reconfiguration	9
3 The SMORES-EP Modular Robot	10
3.1 Mechanical Design	10
3.2 Electrical Design	10
3.3 Software and Networking	11
4 The EP-Face Connector	14
4.1 Related Work	14
4.1.1 Electro-Permanent Magnets	14
4.1.2 Modular Robot Connector Systems	15
4.2 Connector Design	15
4.2.1 Physical Design	15
4.2.2 Manufacturing	16
4.2.3 Electrical Design	17

4.2.4	Integration with the SMORES-EP Module	19
4.3	Experimental Results	20
4.3.1	EP Magnet Characterization	20
4.3.2	EP-Face Characterization	20
4.4	Discussion	22
4.4.1	Advantages	22
4.4.2	Disadvantages	27
4.5	Conclusions and Future	27
5	PaintPots	29
5.1	Introduction	29
5.2	Related Work	30
5.2.1	Potentiometers	30
5.2.2	Ubiquitous Electronics	31
5.3	Background: Potentiometer characterization	31
5.3.1	Conformity (Accuracy)	31
5.3.2	Resolution	31
5.3.3	Hysteresis	32
5.3.4	Lifetime	33
5.4	Design and Manufacturing	33
5.4.1	Design Overview	33
5.4.2	PaintPots used in SMORES-EP	33
5.4.3	Cost	36
5.5	Calibration	36
5.5.1	Ground-Truth Data: AprilTags	36
5.5.2	Model fitting	37
5.6	Performance	37
5.6.1	Accuracy	37
5.6.2	Resolution	38
5.6.3	Hysteresis	38
5.6.4	Lifetime	38
5.6.5	Comparison to Commercial Potentiometers	41
5.7	Two-Dimensional PaintPots	41
5.8	Conclusion	43
6	Design Embedding	46
6.1	Introduction	46
6.2	Related Work	46
6.3	Preliminaries	47
6.4	Topological Embedding	48
6.4.1	Definitions and Statement of Main Result	48
6.4.2	Outline of Algorithm	49
6.4.3	Formal Analysis	51
6.4.4	2-pass Approach	52
6.5	Kinematic Admissibility	54
6.5.1	Extending Definitions	54

6.5.2	Kinematic Admissibility	55
6.5.3	Checking Kinematic Admissibility	57
6.6	Experiments	57
6.7	Applications	58
6.8	Conclusion and Future Work	60
7	Accomplishing High-Level Tasks with Modular Robots	61
7.1	An End-to-End System for Accomplishing Tasks with Modular Robots	61
7.1.1	System Overview	62
7.1.2	Contributions	63
7.2	Related Work	64
7.3	Background	65
7.3.1	Modular Robot Systems	65
7.3.2	Controller Synthesis	65
7.4	System	66
7.4.1	Modular Robot Hardware - SMORES-EP Robot	66
7.4.2	Design and Simulation Tool: VSPARC	67
7.4.3	Design Library	68
7.4.4	Reactive Controller Synthesis and Execution with the Library	71
7.5	Experimental Results	73
7.5.1	Simulated Task Scenarios	73
7.5.2	Hardware Experiments	75
7.6	Discussion and Future Work	79
7.6.1	Simulator-to-hardware translation	79
7.6.2	Library Creation: Lessons Learned	81
7.6.3	Composing Library Elements to Complete Missions	81
7.7	Conclusion	82
8	Autonomy	83
8.1	Introduction	83
8.2	Results	84
8.3	Discussion	88
8.3.1	Challenges and Limitations	89
8.4	Methods and Materials	90
8.4.1	Hardware - Sensor Module	90
8.4.2	Perception and Planning for Information	91
8.4.3	High-Level Planning and Library	92
8.4.4	Reconfiguration	93
8.5	Additional Commentary on Related Work	95
II	Environment Augmentation	97
9	Environment Augmentation	98
9.1	Introduction	98
9.2	Related Work	99

9.3	Approach	100
9.3.1	Environment Characterization	100
9.3.2	Hardware: Augmentation Modules	102
9.3.3	High-Level Planner	103
9.4	System Integration	105
9.5	Experiment Results	106
9.5.1	Experiment I	106
9.5.2	Experiment II	107
9.6	Discussion	107
9.6.1	Conclusion	108
10	Optimal Structure Synthesis	109
10.1	Related Work	109
10.2	Problem Formulation	110
10.2.1	Preliminaries	110
10.2.2	Structures	111
10.2.3	Conflicts	112
10.2.4	Problem Statement	113
10.2.5	Approach Summary	113
10.3	Waterfall Algorithm - Generating All Useful Structures	113
10.3.1	Algorithm	115
10.3.2	Proofs and Analysis	115
10.4	BB-MST Algorithm – Solving for the Minimum Spanning Tree of Structures .	116
10.4.1	NP-Hardness of struct-MST	116
10.4.2	Algorithm	118
10.4.3	Proof	120
10.4.4	Runtime	120
10.5	Results	121
10.5.1	Examples and Experiments	121
10.5.2	Runtime Performance	123
10.6	Discussion and Future Work	124
10.7	Conclusion	125
11	Conclusion and Future Work	126
11.1	Limitations	126
11.1.1	Reconfiguration	126
11.1.2	Environment Augmentation	127
11.2	Future Work	127
	Bibliography	129

List of Tables

1	Comparison of Connectors	28
2	Error metrics for PaintPots	37
3	Track lifetimes	40
4	Examples of property names	68
5	Matrix of designs and properties. Length and mass units are module-lengths and module-masses.	70
6	High-level Action Definitions for Scenario 1	74
7	High-level Action Definitions for Scenario 2	75
8	Reasons for demonstration failure.	90
9	A library of robot behaviors	93
10	Outcomes for Experiments 1 and 2	108
11	Runtime Performance	124

List of Figures

1	SMORES-EP module (left) and exploded CAD view (right)	11
2	Electronic Architecture	12
3	Left: Internal view of magnets in EP-Face. Right: Internal view of EP-face with circuit board and slipring.	17
4	EP-face on a SMORES-EP module.	17
5	Left: EP magnets, before and after winding. Right: Technical drawing of EP magnet, dimensions in millimeters.	18
6	Gluing fixture (left) and winding machine (right)	18
7	Misaligned EP magnet. The left pole (circled in red) makes contact on its edge rather than its face, reducing contact surface area and flux transmission.	18
8	Circuit for Driving EP Magnets.	19
9	(Left) Shear and (Right) Angular Offset test setups	22
10	Plot of Holding Force vs. Gap at Firing. Exponential fit: $y = 18.63 \exp(-7.019x) +$ $12.08 \exp(-0.1697x)$	23
11	Histogram of Holding Forces Under Normal Load. Mean=88.4N, Std=13.9N.	23
12	Face Break Force vs. Supply Voltage. Cubic fit: $y = 0.313x^3 - 14.3x^2 +$ $215x - 981$	24
13	Force vs. displacement in shear. Peak force of 31.8N at displacement of 3.1mm. Note while static friction failure appears to occur at a displacement of 1mm, this large displacement is due to deformation (slop) of the module, not movement of the magnets.	24
14	Normal Holding Force vs. Angular Offset. Linear fit: $y = -3.194x + 90.7$	25
15	Slide-by docking is made possible by the thin profile of the EP-Face.	25
16	Ledge exploration. (Left) Seven-module snake lifts its head to the top of a 3- module high ledge. (Center) Head module detaches, and explores the surface. (Right) Head module autonomously reattaches, and snake descends.	25
17	SMORES-EP module moving a 1kg metal block while lifting another module in the air.	26
18	Potentiometer schematic showing the three parts (track, terminals, and wiper) as well as position and voltage labels.	30
19	A bead of conductive paint applied beneath the screw head forms a good electrical connection with the track.	33
20	SMORES-EP module with labeled joints. The module is the size of an 80mm cube.	34

21	Top Left: Wheel PaintPot installed in chassis. Top Right: Drawing of wheel PaintPot with dimensions in mm. Bottom: Circuit board used with wheel PaintPots showing Harwin S1791-42 wipers mounted at a 50° angle.	35
22	Top: Tilt PaintPot installed on chassis with cylindrical curvature (28.5mm radius). Bottom: Drawing of tilt PaintPot track (laid flat) with dimensions in mm.	36
23	Top: Plot of mean absolute RGV with increasing window size. Red line indicates resolution limit, $\overline{ RGV }(w) = 1$. Right: Bottom of RGV computed on the same dataset with different window sizes. We can see that the data becomes smoother with increasing window size.	39
24	Testing setup used to evaluate RGV and hysteresis.	40
25	Plots of cost vs. conformity and cost vs. lifetime for commercial potentiometers with features similar to the wheel PaintPot (360° sensing range and continuous rotation). PaintPot marked with red square.	42
26	Left: Spherical PaintPot that senses position on the top hemisphere. Right: Flat-sheet PaintPot capable of sensing the X-Y position of the wiper.	43
27	Top: Top-down view of voltage gradients on the sphere PaintPot. Bottom: Voltage gradients on sheet PaintPot.	44
28	Topological conditions for embedding.	49
29	The three cases of message construction.	53
30	Def. 4, 1 (nodes). In fig. (c), a position and orientation have been found in which each child joint of b is aligned with a corresponding child joint of b' . . .	55
31	Def. 4, 2 (paths). For path $\pi_{ab} = (a', c', b')$ to embed edge (a, b) , there must be angles $[\Theta(\pi_{ab})]^*$ for which ${}^B\mathbf{r}_{B'/B} = [{}^B\mathbf{r}_{B'/B}]^*$ and ${}^B R^{B'} = [{}^B R^{B'}]^*$	56
32	2-pass against naive embedding on random trees. x -axis is benchmark size as a function of the nodes in the subdesign and the superdesign. Timeout is 2 hours.	57
33	Grasper (left) and a walker (right) designs for the SMORES robot [17].	58
34	Walker design on top and grasper design on bottom. Red arrows show the discovered embedding.	59
35	SuperBot subdesign [81] embeds in SMORES superdesign.	60
36	Six configurations from the design library	62
37	System flowchart	63
38	VSPARC user interface	67
39	The same behavior file can be used by both the simulator and the physical robot.	68
40	Controller synthesis and execution	73
41	Environments for Scenarios 1 (top) and 2 (bottom) in the simulator.	76
42	Map of the hardware demo	76
43	Simulated Demo	79
44	Moving the Waste Bin	80
45	Cleaning the Table	80
46	System Overview Flowchart	84

47	Sensor Module with labelled components. UP board and battery are inside the body.	85
48	Environments and Tasks for Demonstrations	87
49	Demonstrations 1, 2, and 3	88
50	Environment Characterization	92
51	Module movement during reconfiguration. Left: initial configuration (“Car”). Middle: module movement, using AprilTags for localization. Right: final configuration (“Proboscis”).	94
52	A task specification with the synthesized controller.	95
53	<i>Left:</i> Example template used to characterize a “ledge” feature. <i>Right:</i> Example template overlayed on elevation map (top view) to evaluate candidate feature pose.	101
54	Characterization of an environment with a “ledge” feature. Red indicates a detected feature, pink indicates the start of the feature, demonstrating orientation.	102
55	Wedge and Block Augmentation Modules	103
56	Bridge and Ramp	104
57	An example of synthesized robot controller	104
58	System Overview Flowchart	105
59	Snapshots throughout Experiment I. From left to right, top to bottom: i) Experiment start ii) Opening first drawer iii) Picking up ramp iv) Placing ramp next to open drawer. v) Reconfiguring and climbing ramp vi) Opening second drawer	106
60	Snapshots throughout Experiment II. From left to right, top to bottom: i) Experiment start ii) Assembling bridge iii) Transporting bridge iv) Placing bridge over gap. v) Reconfigure and cross bridge. vi) Arrive at the target zone.	107
61	Example environment and structure. Top: Example table-and-chair environment (left), and corresponding height field world (right), with labeled regions. Bottom: Example structure X-Y view (left) and $\hat{\mathbf{u}}$ -Z view (right).	114
62	Optimal solutions in simulated environments.	122
63	Real-world environments, and algorithm solutions generated from 3D map data taken with a Kinect sensor.	123
64	Log-histogram of solution times for 1118 random environments. In addition to the data shown, 17 environments timed out after 10 minutes.	124

Chapter 1

Introduction

Developing flexible, broadly capable systems is essential for robots to move out of factories and into our daily lives, functioning as responsive agents that can handle whatever the world throws at them. This dissertation focuses on two kinds of robot adaptation. *Modular self-reconfigurable robots* (MSRR) adapt to the requirements of their task and environments by transforming themselves. By rearranging the connective structure of their component robot modules, these systems can assume different morphologies: for example, a cluster of modules might configure themselves into a car to maneuver on flat ground, a snake to climb stairs, or an arm to pick and place objects. Conversely, *environment augmentation* is a strategy in which the robot transforms its environment to meet its own needs, adding physical structures that allow it to overcome obstacles.

In both areas, the presented work includes elements of hardware design, algorithms, and integrated systems, with the common goal of establishing these methods of adaptation as viable strategies to address tasks. The research takes a systems-level view of robotics, placing particular emphasis on experimental validation in hardware.

1.1 Motivation

Robots have been deployed in factory automation roles for decades. Modern robotics research strives to deploy robots “in the wild”: outside of factories and labs, and in our wider world. This has proven to be extremely challenging. Robots in industrial automation roles move swiftly, precisely, and reliably by relying on tightly controlled environments: the bolt they need to screw into a hole on one part will be screwed in to the exact same place on the next part, allowing the robot to complete this repetitive task faster than a human counterpart. But this reliance on stiffness and precision makes these systems brittle to small changes: if the position of the bolt hole is changed by one centimeter, it can result in failure of the entire task, and even damage to the robot.

Operation in the wild requires robots to respond to a huge degree of variability in the environments, objects, and scenarios the robot could encounter; where a factory robot might complete a task the same way hundreds or thousands of times, a robot in the wild might never complete a task exactly the same way twice. Furthermore, a high-level task (like exploring a disaster zone to find survivors) might require several disparate capabilities, like climbing stairs and manipulating objects.

Designing robots that adapt to this level of variability is challenging. Some modern

systems take inspiration from humans and animals: bipedal and quadrupedal robots have multiple articulated kinematic chains with large dextrous workspaces, allowing them to complete a wide range of tasks. However, introducing so many degrees of freedom makes control difficult, requiring each task to be solved in a complicated way. For example, to pick up and move an object, a humanoid must balance on two legs while using a high degree of freedom arm to manipulate the object. In contrast, a robot purpose-built for a single task can often accomplish it with far fewer DOF, lower-fidelity sensing, and less complicated control algorithms.

While the full set of capabilities required by a robot is very broad, few tasks require all those capabilities simultaneously. The success of industrial robots provides evidence that specialization to a single task can provide advantages in terms reduced complexity and increased reliability. Modular reconfigurable robots operate on the principle of specialization-on-demand: the ability to physically transform lets them to take on a morphology tailored to the needs of each new task they encounter. This stands in contrast to other modern systems with bipedal or quadrupedal morphologies. These systems can complete a wide range of tasks, but often need to solve individual tasks in a complicated way: for example, while manipulating an object with its arm, a humanoid must also work to balance on two legs. In contrast, a modular reconfigurable robot could transform between morphologies specialized to manipulation (an arm) and mobility (a car or walking robot) to address individual tasks requiring those capabilities.

Likewise, industrial robots serve as a testament to the power of structured environments: with perfect knowledge of the environment and a good match between the capabilities of the robot and the task at hand, robots are efficient and effective. As roboticists, we typically view the conditions of the environment as problem constraints, and strive to create systems that act within those constraints. Rather than accommodating the inconvenient conditions robots often encounter in the wild, environment-augmenting robots actively build favorable structure on unstructured environments, making the world around them a little more like a robot-friendly factory floor.

This dissertation presents hardware systems, algorithms, and planning frameworks that lay the foundation for reconfiguration and environment augmentation to be employed as strategies to address tasks. In hardware experiments, we show how these strategies enable the SMORES-EP robot to autonomously complete several tasks in office-like environments. By providing foundational tools and evidence of effectiveness, this dissertation strives to pave the way for self-reconfigurable and environment-augmenting robots to be deployed in real-world applications.

1.2 Reconfiguration

Modular self-reconfigurable robot (MSRR) systems are composed of repeated robot elements (called *modules*) that connect together to form larger robotic structures, and can *self-reconfigure*, changing the connective arrangement of their own modules to form different structures with different capabilities. Since the field was in its nascence, researchers have presented a vision that promised flexible, reactive systems capable of operating in unknown environments. Modular self-reconfigurable robots would be able to enter unknown environments, assess their surroundings, and self-reconfigure to take on a form suitable to the task and environment at hand [105]. Today, this vision remains a major motivator for work in

the field [107].

Continued research in MSRR has resulted in substantial advancement. Existing research has demonstrated MSRR self-reconfiguring, assuming interesting morphologies, and exhibiting various forms of locomotion, as well as methods for programming, controlling, and simulating modular robots [12, 22, 26, 39, 52, 60, 62, 72, 76–78, 102, 105, 108]. However, achieving autonomous operation of a self-reconfigurable robot in unknown environments requires a system with the ability to explore, gather information about the environment, consider the requirements of a high-level task, select configurations whose capabilities match the requirements of task and environment, transform, and perform actions (like manipulating objects) to complete tasks. Existing systems provide partial sets of these capabilities. During my Ph.D, I have developed hardware, algorithms, and integrated systems that enable MSRR to meet this longstanding goal of reactive, task-driven reconfiguration.

My early work in hardware design culminated in the SMORES-EP modular robot, the core hardware platform for my research. In the process of designing SMORES-EP, I have developed novel connector and position encoder technologies, covered in Chapters 4 and 5. The EP-Face [92] is an array of electro-permanent (EP) magnets that provides a strong (90N) connection between SMORES-EP modules. EP magnets combine the advantages of permanent magnets and electromagnets - the magnetic force between two modules can be switched on (attractive) and off (no force) by applying a very short pulse of current. The magnets will then maintain either state indefinitely without consuming energy. PaintPots [93] are low-cost, highly customizable potentiometers for position encoding. Applying off-the-shelf carbon-embedded spray paint to plastic surfaces creates position sensors in a variety of shapes and sizes. A calibration process yields accuracy and precision comparable to commercial sensors.

My work in algorithms focuses on the relationships between robot morphologies and robot capabilities. Reconfigurability introduces a theoretical challenge that traditional robots do not face: to address a task, we first need to select an appropriate morphology for the robot to assume. Taking some of the first steps to solve this problem, Chapter 6 presents my algorithmic work in modular robot design embedding, which formally defines conditions under which structurally different modular robot configurations can be considered functionally equivalent. Robots are represented as labelled graphs, and embeddability is defined in terms of conditions on robot topology and kinematics. Along with my collaborators, we have developed an algorithm that can automatically detect embeddability of modular robot designs in polynomial time using dynamic programming [52].

In collaboration with researchers specializing in high-level control, I have developed systems that enable modular robots to address complex, multi-part tasks. Chapter 7 presents a system enabling users to solve complex tasks using modular robot hardware. Our approach is library-driven: rather than attempting to generate new designs from scratch, users specify task requirements and a design tool retrieves designs satisfying the requirements from a library of existing useful designs. User-specified task requirements are synthesized into provably correct state machine controllers using behaviors and configurations from the library. A seamless pipeline from high-level task specification down to hardware configurations and behaviors results in an end-to-end system for accomplishing tasks with modular robots.

Finally, Chapter 8 presents a novel integrated system incorporating modular robot hardware, perception tools, and high-level planning, which allows modular robots to complete

complex high-level tasks autonomously. The system automatically selects appropriate behaviors to meet the requirements of the task and constraints of the perceived environment. Whenever the task and environment require a particular capability, the robot autonomously self-reconfigures to a configuration that has that capability. By providing a clear example of how a modular robot system can be designed to leverage reactive reconfigurability in unknown environments, this system begins to lay the groundwork for reconfigurable systems to address tasks in the real world.

1.3 Environment Augmentation

Robotics research typically aims to create *reactive* systems, capable of sensing, thinking, and acting in response to the constraints of their environment. But what if we could create robots that were *proactive*, actively altering their environments to enhance their ability to move, manipulate, and sense? Could we make robots that overcome the challenging “unstructured” environments of the real world by reshaping them to be more predictable and convenient?

Augmenting the environment to accomplish tasks is a familiar human experience: to reach an object on a high shelf, we place a ladder near the shelf and climb it, and at a larger scale, we construct bridges across wide rivers to make them passable. I believe environment augmentation can provide even greater benefit for robots than it does for humans. Robots can be specifically designed for tasks like construction (imagine a robot with nail guns for arms, and an onboard silo of building material), and are physically (and emotionally) capable of permanently integrating themselves with their environment if needed.

Developing robots that augment their environments involves research challenges in design and planning. In both areas, there are parallels with modular robots. From a design perspective, modular robots are well-suited to environment augmentation: modular systems are designed for connection, making it natural to think about modules manipulating passive pieces about the same size as themselves, or even attaching themselves to their environment. The physical benefits of environment augmentation also complement a weakness of modular robots, which often struggle with obstacles much larger than a module. From the perspective of task planning, environment augmentation is similar to reconfiguration: in both cases, the robot can take an action (reconfiguration or augmentation) that fundamentally changes its future capabilities with respect to its environment and task. Chapter 9, introduces passive block and wedge building blocks that SMORES-EP can use to build ramps and bridges. Expanding on our existing frameworks, we present a library-based system allowing SMORES-EP to autonomously deploy structures to complete high-level tasks in office environments

Existing work in collective construction robotics and provides algorithms to generate assembly plans for arbitrary shapes in 2D and 3D [83, 100]. In most of this work, construction of a prescribed structure is the primary objective. An environment-augmenting robot needs another layer of planning to reason about its own capabilities and the constraints of the environment to decide what structures should be built. Consider a scenario where a small robot must move through an environment filled with objects much larger than itself. This task presents a challenging planning problem: given an environment, a robot, and supply of building blocks, can we find a set of structures that could be added to the environment to make it fully accessible to the robot? Furthermore, since structure-building is a time-consuming process, can we find such a set of structures which uses a minimum number of

building blocks? We refer to this as the *optimal structure synthesis* problem.

Chapter 10 presents a mathematical formalism for optimal synthesis of structures made of discrete building blocks, as well as a complete, optimal algorithm that will find a min-cost set of structures to make any input environment traversable, if such a set exists. This work is a theoretical complement to the system presented in Chapter 9, providing a general framework to optimally augment arbitrary environments, rather than relying on a discrete library of structures.

Chapter 2

Overview of Related Work

This chapter provides an overview of the literature related to modular self-reconfigurable robots and environment augmentation. A more detailed discussion of the related work pertinent to each chapters of the dissertation is included within each individual chapter.

2.1 Modular Self-Reconfigurable Robots (MSRR)

2.1.1 MSRR Hardware Systems

Modular self-reconfigurable robot systems distinguish themselves from traditional robots through their ability to reconfigure, changing the connected structure of their component modules to assume different shapes. Systems are typically categorized according to the geometric arrangement of their modules, and their modes of reconfiguration [107]. *Lattice architecture* systems have modules that are connect in a dense lattice or grid pattern, Modules typically have limited movement ability, instead relying on their ability to connect to and interact with neighboring modules to move within the lattice. Some lattice architecture systems have no autonomous movement ability, relying on external, sometimes stochastic forces to move modules into a desired shape. *Chain architecture* systems have modules that connect in a chain or tree topology. Individual modules usually include revolute or prismatic joints, allowing clusters of modules to form traditional kinematic trees that operate in a manner similar to traditional robots. *Mobile architecture* systems have modules that are individually mobile, and can reconfigure by breaking off individual modules and having them individually move to another location on the cluster. Some modular systems fall under more than one of these categories, and are referred to as *hybrid* systems. SMORES-EP is a hybrid system with the capabilities of all three categories, being able to operate as a lattice or tree, and also perform mobile reconfiguration. A comprehensive review of major modular reconfigurable hardware systems is available in [107].

Making and breaking connections between modules is one of the most essential physical capabilities of MSRR systems, and connector design is a major area of research in the field. Chapter 4 provides an overview of the wide variety of existing connectors found in the literature.

2.1.2 Algorithms and Control

A great deal of work has been done to develop behaviors for MSRR. Many efforts focus on distributed control strategies, taking advantage of the distributed nature of MSRR hardware

[99]. Distributed strategies include central pattern generators [85] and hormone-based control [80]. Genetic algorithms have been used to automatically generate both modular robot designs and behaviors [36]. Historically, gait tables have been a commonly used format in which open-loop kinematic behaviors can be easily encoded [105]. Phased automata have also been presented as a way to easily create scalable gaits for large numbers of modular robots [109].

Efforts have also been made to generate behaviors by automatically identifying the “role” a module should play based on its place in a connected structure [86]. Functionality is propagated downward: based on a high-level goal (like “walk”) and a connected structure of modules, functional sub-structures (like legs and a spine) are automatically identified, and modules are directed to execute appropriate roles in a distributed fashion. This work provides some heuristic automatic matching of structure to function, and has only been demonstrated for locomotion gaits.

In addition to automatically generating behaviors for MSRR, work has also been done to automatically generate MSRR configurations from functional specifications [11, 66, 104]. Chapter 6 presents a more extensive overview of this work. Planning and control for reconfiguration are major topics of interest within the field of MSRR [62, 72, 77, 108]. More detail on existing strategies for reconfiguration control and planning can be found in Chapter 8.

2.1.3 Tasks and Autonomy

Most prior efforts to generate functional behaviors for MSRR systems have focused primarily on establishing the low-level capabilities that distinguish these systems from traditional robots, notably reconfiguration, a wide variety of gaits for movement, basic object manipulation, and other low-level tasks. However, the advantage of a reconfigurable system lies in its ability to select appropriate configurations and capabilities for a high-level task, and even reconfigure multiple times to complete a complex, multi-step task requiring a wide range of abilities. While there has been a great deal of research in high-level planning for traditional robots [6, 7, 42, 46, 75, 103], the application of these methods to MSRR has been limited [10]. Chapter 7 presents a high-level planning framework for MSRR, as well as an overview of other strategies for addressing tasks with MSRR from the literature.

The true value of MSRR systems is achieved when they are able to operate autonomously. Autonomous operation requires a system with the ability to explore, gather information about the environment, consider the requirements of a high-level task, select configurations whose capabilities match the requirements of task and environment, transform, and perform actions (like manipulating objects) to complete tasks. Existing systems provide partial sets of these capabilities. Many systems have demonstrated limited autonomy, relying on beacons for mapping [20, 29] and human input for high-level decision making [21, 59]. Others have demonstrated swarm self-assembly to address basic tasks like hill-climbing and gap-crossing [31, 70]. While these existing systems all represent advancements, none have demonstrated fully autonomous, reactive self-reconfiguration to address high-level tasks. Chapter 8 presents the first system allowing modular robots to autonomously complete complex tasks by reconfiguring in response to their perceived environment, and provides a more detailed overview of related systems.

2.2 Environment Augmentation

The fields of collective construction robotics and modular robotics offer examples of systems that build and traverse structures made of robotic or passive elements [74, 76, 90], and assembly planning algorithms to build arbitrary shapes under a variety of conditions [83, 100]. This existing work provides excellent contributions regarding the generality and completeness of these methods: some algorithms are provably capable of generating assembly plans for arbitrary volumetric shapes in 3D, and hardware systems have demonstrated the capability to construct a wide variety of structures. Other work in manipulation planning allows robots to carry out multi-step procedures to assemble furniture [45] or rearrange clutter surrounding a primary manipulation task [19].

Less work is available regarding ways that robots could deploy structures as a means of completing an extrinsic task, the way a person might use a stepstool to reach a high object. Napp et al present an distributed algorithm for adaptive ramp building with amorphous materials [63]. Using only local information, the algorithm allows one or more robots to deposit amorphous material (like foam) on their environment to make a goal point accessible. In recent work, a similar algorithm allows a team of robots to collectively build with sandbags to reach a goal location [79].

Petersen et al. present Termes [74], a termite-inspired collective construction robot system that creates structures using blocks co-designed with a legged robot. Similarly, Chapter 9 presents augmentation modules that are designed to be easily carried and traversed by SMORES-EP. Werfel et al. present algorithms for environmentally-adaptive construction that can build around obstacles in the environment [100]. A team of robots senses obstacles and builds around them, modifying the goal structure if needed to leave room for immovable obstacles. An algorithm to build enclosures around preexisting environment features is also presented. As with Termes, the goal is the structure itself; while the robots do respond to the environment, the structure is not built in response to an extrinsic high-level task.

For a more detailed overview of work related to environment augmentation, see Chapters 9 and 10.

Part I

Reconfiguration

Chapter 3

The SMORES-EP Modular Robot

The SMORES-EP modular reconfigurable robot is the core hardware system used in my work. This chapter provides an overview of its major mechanical, electrical, and software systems.

3.1 Mechanical Design

SMORES-EP has an 80mm cube-like form factor, intended to allow packing into a cubic lattice. The basic kinematic design consists of two side wheels and a gear-differential mechanism that creates pan and tilt motion. This design was taken directly from its predecessor, the SMORES robot, and allows “emulation” of many past modular robot systems, in the sense that groups of SMORES modules can be arranged in ways that replicate the kinematics of groups of other modular robots [17]. The four degrees of freedom of the modules are shown in Figure 1. The right and left wheels are driven directly by Pololu Micromo motors with 298:1 gearboxes, allowing them to spin at a maximum of 90 degrees per second. The pan and tilt degrees of freedom are coupled, with two Pololu Micromo motors with 1000:1 gearboxes driving three gears in a differential configuration. This allows both degrees of freedom to be operated simultaneously with little mutual interference, and allows both motors to drive the tilt joint, which is frequently used to lift loads. The tilt joint can support a torque of 0.7 N-m, meaning a single module can lift 4 SMORES-EP modules held out in cantilever (horizontally against gravity).

The design of a SMORES-EP module is itself somewhat modular, which makes it easier to testing and maintain the the fleet of 25 modules. The four faces (left, right, top, and bottom) are held together by eight screws, as shown in the exploded view in Figure 1. The left, right, and top faces are nearly identical, each consisting of an inner chassis, a set of gears, and an external face with an array of electro-permanent magnets. The bottom face holds an EP-magnet array and the four motors. The space in the center of the cube is occupied by a layered bank of circuit boards, wiring, and the battery.

3.2 Electrical Design

Each module responds to user commands via WiFi, and individually control four motors and four EP magnet arrays. Electrical design and wiring was challenging, because these subsystems are distributed across five separate bodies that need to rotate relative one another for the module to function. Each module has 12 circuit boards and five microcontrollers

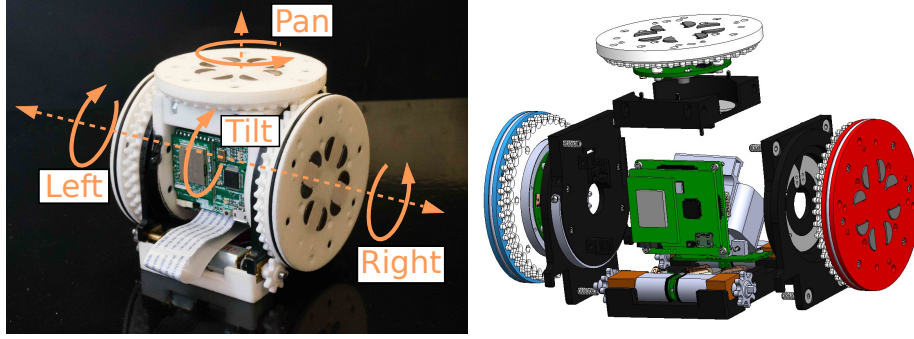


Figure 1: SMORES-EP module (left) and exploded CAD view (right)

distributed across the different moving bodies of the module. Figure 2 shows a schematic representation of the electronic architecture.

The EP magnets in the top, left, and right faces need to rotate continuously relative the center of the module, and are connected to the central motherboard via slip rings. The bottom face rotates through a range of 180 degrees relative the center body, and is connected to the central electronics by a ribbon cable.

With limited space available in the center of the module, the driving circuitry for each face’s magnets is located on the corresponding face board. Each face board also includes an ATmega168a microcontroller, which communicates with the more powerful STM32F303 microcontroller on the motherboard via i2c. The ATmega is also responsible for reading voltages from the PaintPot encoders in each face, discussed in Chapter 5.

3.3 Software and Networking

Computation for SMORES-EP is distributed across three physically separate computing units: the faces, where ATmega168a microcontrollers are programmed in C, the module motherboard, where an STM32F303 microcontroller is programmed in C++, and the controlling computer, where behaviors for one or more modules are programmed in Python. This section provides a brief overview of the software and networking architecture, starting at the high level and working downward.

A cluster of modules may be controlled by any computer capable of running Python. The SMORES-EP control library provides the user with a `SmoresModule` object, with functions to send movement and magnet commands to a single module, as well as querying the modules status and encoder values. When the object is created, it opens a port to communicate with the specified module via wifi. Each module has a unique static IP address determined by its ID number. The Python library also provides a `SmoresCluster` object, which can create and communicate to multiple modules simultaneously. The base module control libraries depend only on the standard Python libraries; care was taken during development to avoid external dependencies (such as ROS), to maximize code portability.

Cluster-level networking is provided by a typical 2.4Ghz 802.11 wireless router. The TI cc3000 WiFi chip on each module can send and receive UDP packets with very little latency; for the purposes of the robot, wireless communication is effectively instantaneous.

The motherboard microcontroller (STM32F303) is responsible for receiving and and re-

Electronic Architecture

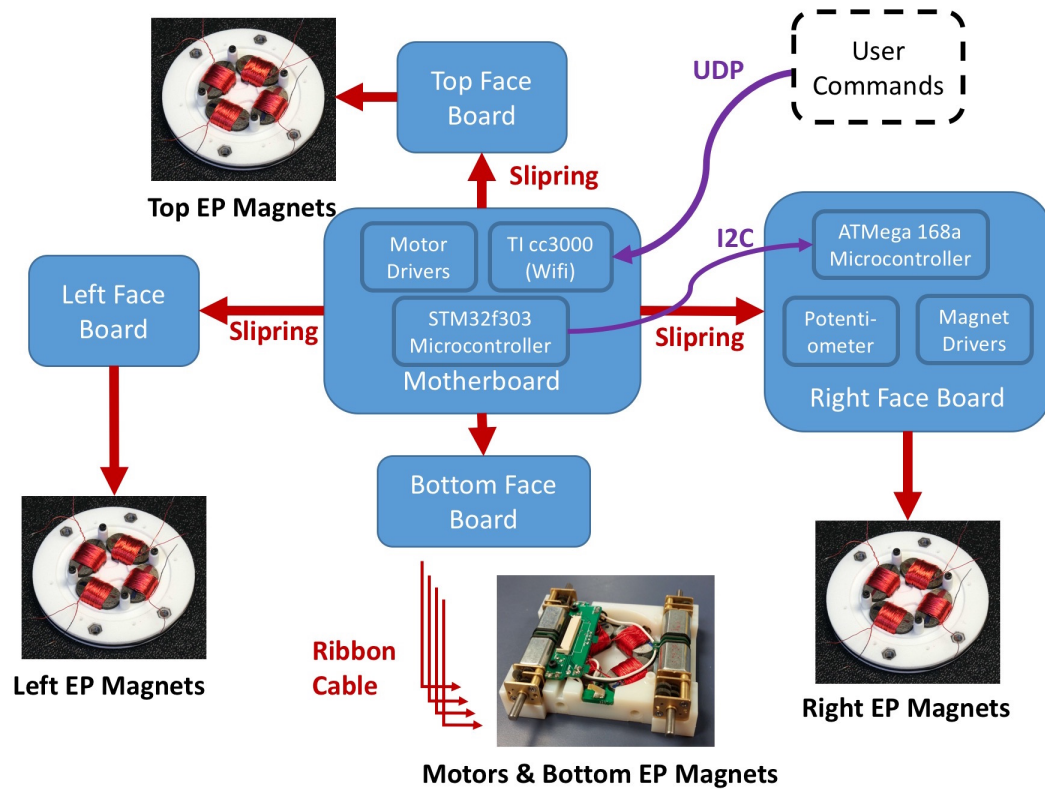


Figure 2: Electronic Architecture

sponding to WiFi commands, performing feedback control for the four degrees of freedom, and communicating with the face boards via i2c. Firmware is written in C++, and the control runs at a rate of 20Hz.

The four faceboard microcontrollers (ATMega168a) are i2c slaves responsible only for dispatching commands received from the motherboard (the i2c master). The firmware, written in C, was designed to be as minimal as possible, because reprogramming these microcontrollers requires disassembling the module. The faceboards are responsible for firing the EP magnets to connect or disconnect from other modules, and for reading and returning the wiper voltage of the PaintPots encoders [93] in their face to the motherboard.

Chapter 4

The EP-Face Connector

This chapter presents the design and analysis of the EP-Face connector, a core component of the SMORES-EP robot. This chapter excerpts heavily from [92]. Credit is due to co-authors Jay Davey and Chao Liu, who contributed significantly to this work.

The EP-Face connector uses an array of four electro-permanent magnets mounted in a planar face to create a high-strength (88.4N) connection between modules. The connector is *fast* (connecting/disconnecting in 80 milliseconds), *compact* (low-profile, solid-state components), *robust* (large area-of-acceptance, self-aligning, genderless, rotationally symmetric, capable of docking from any approach direction), and *energy efficient* (requiring only 2.5 joules to switch states). This chapter presents the connector design, characterizes it through experiments, and compares its performance with existing connector designs. In Section 4.1, we provide an overview of existing modular robot connectors. In Section 4.2.1, we present the connector design, as well as the manufacturing processes and fixtures used to construct hundreds of EP magnets. In Section 4.3, we characterize the connector through experiments. In Section 4.4, we discuss our results and compare the connector with other systems. Finally, in Section 4.5, we discuss possible future extensions, and conclude.

4.1 Related Work

4.1.1 Electro-Permanent Magnets

An electro-permanent magnet consist of two permanent magnet rods with an electromagnet coil wrapped around them. Both rod magnets have the same remnant magnetization, but one has relatively low coercivity (polarization can be changed through exposure to a magnetic field) while the other has high coercivity (a much larger magnetic field is required to change polarization). A short pulse of current through the coil sets the polarization of the low-coercivity magnet, allowing magnetic force to be turned on or off (on when both are polarized the same way, off when opposite). Once set, polarization is maintained until another pulse is applied. The reader is referred to [44] for more information.

EP magnets have been used as connectors in lattice-type modular robots and programmable matter systems [28, 32, 43]. The Pebbles and Lily robots operate in a 2d lattice, and are primarily concerned with cluster self-assembly or self-dissassembly rather than strength. Their magnetic connectors withstand in-plane forces of 3.18N and 1.28N, respectively. Each Pebble is 10mm long and weighs, 4.0g; each Lily is 35 mm long and weighs 26g [28, 32].

The EP-Face is component of SMORES-EP, a hybrid chain-lattice type modular robot, intended to form articulated chains that serve as bodies and legs as well as three dimensional lattices. SMORES-EP is much larger and heavier than the above systems, with a characteristic length of 80mm and a mass of 500g/module. As such, it has very different connector requirements. The EP-Face connector is expected to withstand forces on the order of tens of Newtons under normal, shear, and bending loading.

4.1.2 Modular Robot Connector Systems

A wide variety of connectors for hybrid modular robots can be found in the literature. Other systems that use magnets include MTRAN II [61] and the Telecubes system [87]. Telecubes and MTRAN II both exert connector forces of about $25N$ per magnet, about the same as the EP-Face ($28.3N$). Both use permanent magnets for latching, and disconnect them using shape-memory alloy (SMA) actuators. The disadvantages of SMA are its slow response time (it can take minutes to cool after heating), and notorious energy inefficiency. The EP-Face is able to switch the state of its EP magnets in 80 milliseconds with little energy ($2.5J$).

The MICHE robot [27] is a predecessor to the Pebbles, and uses mechanically switchable permanent magnet connectors that exert about $20N$ of force. The connectors control the flow of magnetic field by changing the relative orientation of two circular permanent magnets using a small gearmotor. The connector uses a small amount of energy, but requires room for the motor, has moving parts, and takes 1.3 seconds to switch states.

Structural hook-type connectors are popular for hybrid self-reconfigurable robots. Examples include the ATRON and MTRAN III robots [47, 69]. The advantage of these connectors is high strength: ATRON can theoretically support up to $800N$ before material failure. Compared to magnets, they sacrifice versatility and often require large amount of space. The majority of volume within each module of the ATRON was consumed by the connection mechanism [69]. They also tend to be mechanically complex, with many moving parts that can break or wear over time. The EP-Face connector is solid-state, requiring only a pulse of current to connect or disconnect.

The SINGO connector, developed for the Superbot robot, is more versatile [84]. It is hermaphroditic, and capable of disconnection even when one module is unresponsive, allowing for self-repair. However, it is mechanically complex, and sacrifices some strength for versatility.

The most natural point of comparison is its predecessor, the original SMORES robot connector [17], with four permanent magnets on a flat face and a mechanical key that enables latching and unlatching. The EP-Face is able to dock in a wider range of conditions than the original SMORES face. It is also stronger in normal loading than the SMORES face ($85N$ compared to $60N$), but weaker in shear ($35N$ compared to (theoretically) $3.6kN$).

A more detailed comparison of the EP-Face connector to existing connectors can be found in Section 4.4.

4.2 Connector Design

4.2.1 Physical Design

The connector is shown in Figures 3 and 4. It consists of an array of 4 EP magnets arranged in a ring, with south poles counterclockwise of north. The ring arrangement of the magnets makes the connector hermaphroditic, and able to connect in four possible configurations.

The magnets are held in place by glue, and externally protrude a distance of 0.5mm beyond the planar surface of the 3d-printed face. This way, the protruding magnets surfaces are the point of contact when connecting faces are brought together, minimizing the possibility of a detrimental air-gap between any pair of magnets.

Internally, leads from the magnets are soldered to a circuit board mounted above the magnets, which includes a microcontroller and driving circuitry, discussed in more detail in the following section. Measuring from the magnet face to the top of the circuit board, the total height of the EP-face connector is 16.6mm. Measuring to the top of the slip ring canister, the total height is 19.6mm.

Figure 5 shows the EP magnets. Each magnet consists of two cylinder magnets (AlNiCo 5 and NdBF_e, both 4.76mm in diameter and 9.53mm long) and two identical pole pieces machined from ASTM 1018 Low-Carbon Steel. Mechanical constraints of the SMORES-EP module require that the magnet array fit within a 42mm diameter circular region. Machining the pole pieces into semicircles maximizes the allowable size of the EP magnets (if rectangular pole pieces were used, the corners would collide). The pole piece has a lip that sits on a corresponding ledge in the face, allowing the foot to protrude out of face by the prescribed amount and serving to transmit force on the magnet to the plastic face. Glue is used to hold the magnets in place in the face, but is not load-bearing.

The surface area of the pole pieces is critical to the strength of the magnetic force. Magnetic force per area is proportional to flux density squared. Therefore, decreasing pole area increases holding force, but only up to the saturation density of low-carbon steel (1.5T). Based on the diameter and average magnetic flux density of the permanent magnet cylinders (1.38T for both NdBF_e and AlNiCo), the minimum contact area of the steel pole piece was found to be 32.7mm². The actual semicircular contact surfaces of the poles have an area of 33mm². Based on these values, the theoretical maximum holding force is 46N per magnet or 184N per EP-face.

The solenoid coil was designed to generate sufficient magnetic field intensity for the AlNiCo magnet to reach saturation. Using the method described by Knaian [44], this was found to be 200 turns of AWG 40 wire.

4.2.2 Manufacturing

To date, we have built 25 SMORES-EP modules, 120 EP-Faces and 480 EP-magnets. The primary manufacturing challenge was ensuring that all pole pieces in an EP-face were aligned into a perfectly flat plane. Misaligned pole pieces have less contact surface area with mated magnets, which can significantly reduce connection strength.

Pole pieces were machined from 0.25x0.5in 1018 Low-Carbon Steel rod stock. Edges were deburred manually and smoothed in a vibratory tumbler overnight. Smoothing was important because sharp pole edges can scrape off wire enamel during the winding process, causing shorts within a magnet.

EP magnets were mechanically assembled using high-viscosity cyanoacrylate glue in a custom 3d-printed fixture shown in Figure 6. Pole pieces and magnet cylinders are vertically clamped, fully constraining motion except for travel along the rails. This gluing process was the most error-prone part of manufacturing, mostly due to glue residue buildup in the fixture. Misaligned magnets (Fig. 7) were bathed in solvent and recycled.

Magnets were wound in a purpose-built machine constructed from Lego Mindstorms (Figure 6). Wound magnets were verified by checking resistance (nominally 2.08 Ohms,

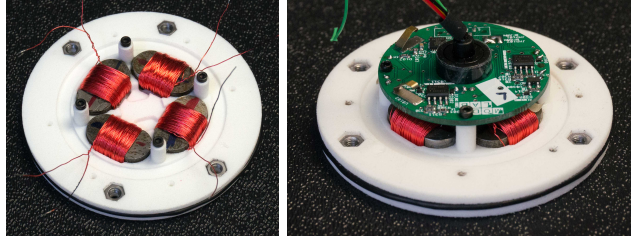


Figure 3: Left: Internal view of magnets in EP-Face. Right: Internal view of EP-face with circuit board and slipring.

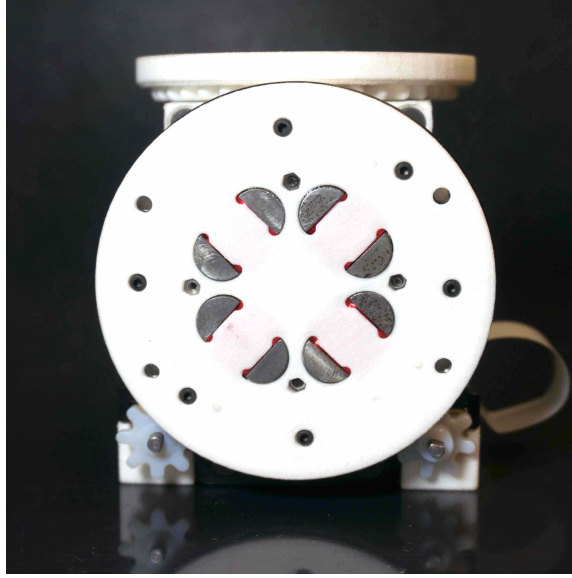


Figure 4: EP-face on a SMORES-EP module.

with lower resistance indicating an internal short), and testing strength (must lift a 2kg steel block when activated manually using a power supply at 11V).

To construct the EP-face, four magnets are inserted into their slots in 3d-printed face. The face is placed on a perfectly flat steel block, and all magnets are activated, forcing them to align to the steel surface as closely as possible. The face is lifted, and if it supports a suspended 5kg load, it is considered up-to-spec. The magnets are fixed in place with glue, and remain attached to the steel surface until the glue cures.

4.2.3 Electrical Design

Driving Circuitry

The four EP magnets in an EP-face are driven by an array of five half-H bridges (Fairchild FDS8958B), capable of sourcing the 6 amps required to activate and deactivate the EP magnets. As shown in Figure 8, one side of each magnet is connected to a dedicated half-H bridge, while the other side is connected to a common half-H bridge shared between all the

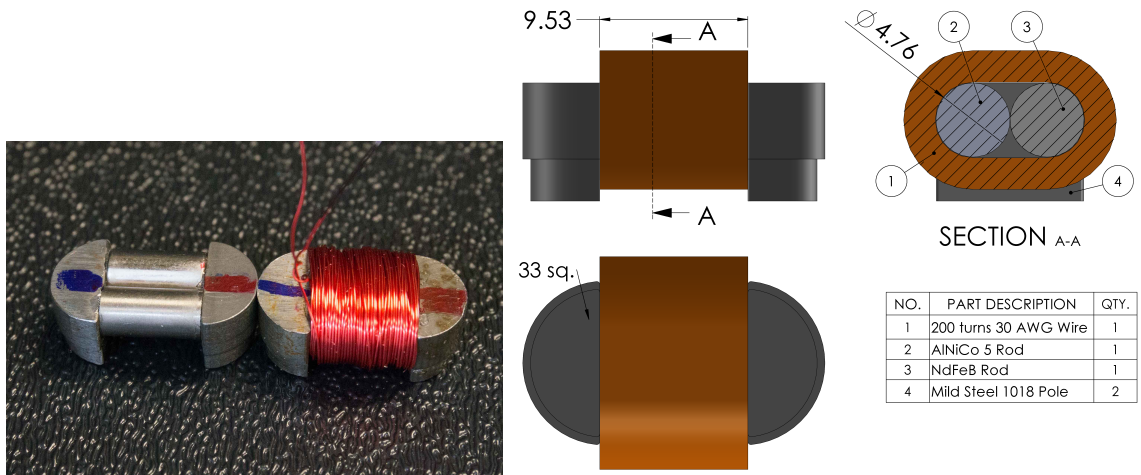


Figure 5: Left: EP magnets, before and after winding. Right: Technical drawing of EP magnet, dimensions in millimeters.

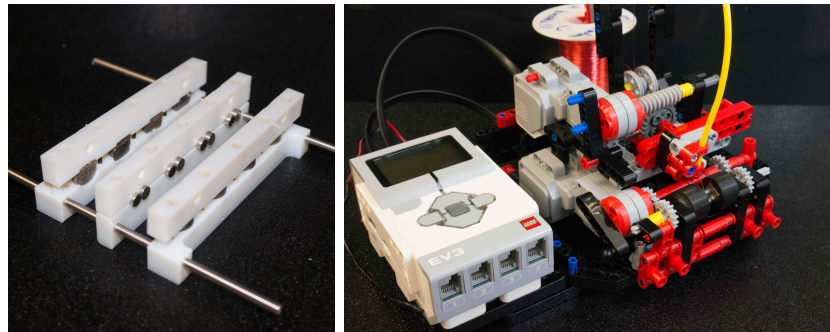


Figure 6: Gluing fixture (left) and winding machine (right)

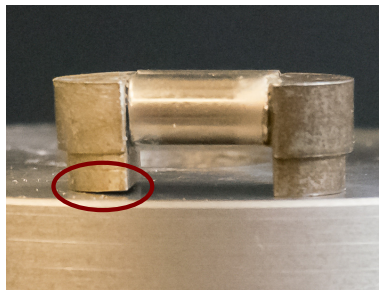


Figure 7: Misaligned EP magnet. The left pole (circled in red) makes contact on its edge rather than its face, reducing contact surface area and flux transmission.

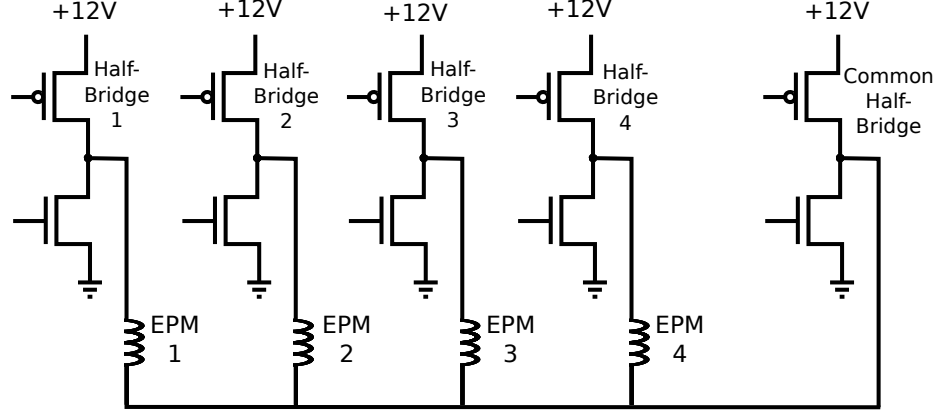


Figure 8: Circuit for Driving EP Magnets.

magnets. This circuit allows bi-directional drive of each magnet, as long as only one magnet is fired at a time. Similar driving circuits are used in [28] and [32].

The array of half-H bridges is controlled by an ATmega 168a microcontroller with 16kb of flash memory, 512kb EEPROM, and 1Kb internal SRAM running at 8Mhz. To activate or deactivate a magnet, three pulses of length 3ms are applied at intervals of 3ms. The magnets are fired at battery voltage (between 11.1V and 12.6V depending on charge). A voltage regulator and capacitor could have been used to provide more consistent firing voltage. The authors chose to omit these components due to tight space requirements, but recommend that others consider them in their own designs.

Inductive Communication

Connected EP-faces can exchange data through the magnetic coupling of connected EP-magnets: when a coil is pulsed, the generated magnetic field also flows through the core of the connected coil, and the changing field generates a voltage across that coil. Through this channel, EP-faces are capable of UART serial communication. Similar capabilities have been demonstrated in [28] and [32].

4.2.4 Integration with the SMORES-EP Module

The EP-face and driving circuitry form a compact self-contained unit with no moving parts (Figure 3). The microcontroller associated with each driver array is configured as an I2C peripheral, and receives commands from the main microcontroller on the module motherboard.

The electrical interface between each EP-face and the rest of the module consists of five lines: High power (battery voltage), logic power (+3.3v), I2C clock, I2C data, and ground. Because the mechanical design of SMORES-EP requires the top and side faces to rotate continuously, these three faces are connected to the central electronics via slip rings (Senring SNM012U-06) mounted in the middle of the face.

The bottom face of SMORES-EP does not rotate continuously, so it does not need a slip ring. Instead, the circuit board is located in the center of the module, and connected to the bottom face magnets through a ribbon cable.

4.3 Experimental Results

4.3.1 EP Magnet Characterization

Holding Force in Normal Loading

The holding force of a pair of EP magnets was characterized using a materials testing machine (MTS) to generate stress/strain plots. In each trial, both magnets were activated by manually pulsing current from a power supply set to 12V. Four pairs of magnets were tested, with five trials per pair. The maximum holding force was 39N, and the average was 28.3N with a standard deviation of 5.2N.

Normal force as function of air gap at firing

The holding strength of an EP magnet is significantly higher when pulsed in contact with another magnet (or ferromagnetic object) than when pulsed in free air. This is because the air forms a magnetic circuit with higher total reluctance, reducing the peak field in the circuit and therefore also reducing the magnetization of the AlNiCo magnet. As mentioned in Section 4.2.2, misaligned pole pieces result in a similar effect.

In this experiment, we characterize the holding force of the magnets as a function of air gap at firing. One pair of magnets was tested. At the beginning of each trial, both magnets were manually deactivated using a power supply. Paper shims were used to create well-controlled effective “air gap” between magnets¹. Both magnets were fired three times with the paper spacer in place. The paper was slid out from between the magnets and measured, the magnets were placed in contact, and a loading test was performed.

Figure 10 plots holding force against gap at firing. Force decreases rapidly with increasing gap distance, and the dropoff is sharp for small gaps. At a gap distance of 0.25mm, holding force is reduced to half of the value with magnets in contact.

4.3.2 EP-Face Characterization

Normal Loading

In this experiment we characterize the holding force of an EP-face. In each trial, faces were aligned and placed in contact, and then magnets were fired three times. Nine pairs of faces were tested. Each pair of faces was rotated through all four possible connection orientations, and five trials were performed for each orientation. By using a large sample size, we capture the variability in holding strength, which is functionally important because the capability of a cluster is limited by the strength of its weakest connector (if one connection breaks, the cluster cannot perform as intended).

The maximum holding force was 115N, and the average was 88.4N with a standard deviation of 13.9N. Figure 11 shows a histogram of holding forces for this experiment. We believe the large variability in holding force is due to the fact that each face-to-face connection consists of four magnet-to-magnet connections, and when loaded in the normal direction, failure of the single weakest magnet-to-magnet connection will cause the entire face-to-face connection to fail. We hypothesize that many of the low-force failures are due to poor contact between a single mated pair of magnets on connected faces, creating a “weakest link” that lowers performance. This is supported by the data: the average standard deviation for

¹We assume the magnetic permeabilities of paper and air are the same. The magnetic permeabilities of nearly all non-ferrous substances (such as paper and air) are very close to μ_0 , the permeability of vacuum.

a given orientation (pairing of magnets)² is 6.13N, while the average standard deviation of all trials for a given face pairing³ is 10.4N.

Effect of Battery Voltage

Magnetization of the AlNiCo magnet depends on the strength of the field created by the coil during pulsing. Since the magnets are fired at battery voltage, changes in battery voltage during operation of a module affects the strength of the magnets. Holding force under normal loading was tested at firing voltages ranging from 9 to 16 volts, using a power supply capable of sourcing sufficient current.

Figure 12 shows the results of these tests. We see a clear trend of increasing holding force with increasing supply voltage, with the curve leveling off around 14V, indicating that the AlNiCo magnetization has saturated. In normal operation, a SMORES-EP module has battery voltage between 12.6V and 11V.

Shear

Holding force under shear loading was tested by connecting two modules side-by-side and pulling one upward (fixture shown in Figure 9). Eight pairs of faces were tested, with five trials performed for each pairing. Different orientations of the pairings were not tested.

The maximum holding force in shear was 41N, and the average was 28.4N with a standard deviation of 6.39N. Figure 13 shows force versus displacement during one of the trials. Two distinct regimes are visible. First, there is a smooth rapid rise in force, due to static friction between the pole pieces. After static friction is overcome (26.52N), force continues to increase as the magnets are pulled away from each other, until failure occurs at 31.8N.

Bending (Characteristic Strength)

Bending strength was tested by connecting two modules side-by-side and pulling upward on a lever mounted on the side of one module. One pair of modules was tested at a battery voltage of 12.6V, with five trials done at each of four lever lengths. The average failure moment at the connected face was $1.8Nm$. Since the SMORES-EP module mass is $0.454kg$ and module length is $81mm$ from magnet to magnet, this is an equivalent load to supporting 3.1 modules in cantilever. This number is used as a figure of merit for modular robots, called *characteristic strength* [23]. In practice, due to variability of connection strength and inertial moments when moving, the functional limit for cantilever structures is two modules in most applications.

Torsion

Torsional strength was tested in a manner similar to bending, except that the modules were mounted so that pulling up on the lever twisted one face relative the other. The average failure moment about the center of the connected faces was $0.83Nm$.

Normal and Parallel Offset Area-of-Acceptance

To test the connection tolerance to offsets in the direction normal and parallel to the connected faces, two modules were positioned offset from one another with bottom faces sitting flat on a table (with magnets turned off), and the magnets were turned on. This process

²Standard deviation of the 5 trials per orientation, averaged over 36 total orientations (4 orientations \times 9 pairings). In a given orientation, the same pairs of magnets are mated in each trial.

³Standard deviation of the 20 trials (5 trials \times 4 orientations) per face pairing, averaged over 9 face pairings.

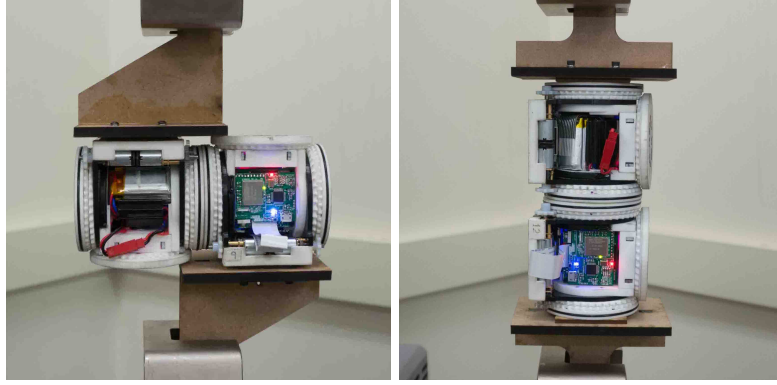


Figure 9: (Left) Shear and (Right) Angular Offset test setups

was repeated with decreasing gap distance until force at magnet activation was sufficient to draw the modules together (demonstrated in the accompanying video).

Magnetic forces can draw two modules together through a gap of 4mm (normal to the faces), and 7mm parallel to the face. The coefficient of friction between the modules and table was experimentally determined to be 0.15.

Rotational Area-of-Acceptance

To test the rotational area of acceptance of the faces, we tested the normal direction breakage strength when the two faces were misaligned. The jig shown in Figure 9 rigidly fixes both the position and orientation of the modules to the clamps of the MTS machine, allowing the connecting faces to be held at a controlled angular offset relative one another.

In each trial, the magnets were first deactivated three times to eliminate any residual magnetization from past trials. The faces were then placed in contact, with a measured angular offset. The magnets were fired three times, and normal load was applied until failure. One pair of faces was tested at three offset angles, with five trials at each offset angle. Figure 14 plots normal holding force against angular offset. Normal force decreases linearly with angular offset, retaining about 10% of the zero-offset value by 25 degrees. Assuming bending strength scales with normal strength, the EP-face can still support one module in cantilever at an offset of 25 degrees (bending failure is $1.8Nm$, and one cantilevered module is $0.19Nm$).

Time and Energy

When an EP-Face is turned on (magnetized) or off (demagnetized), each magnet takes $20ms$ to fire ($9ms$ of current pulses, $11ms$ of wait time). To change the state of an entire face, this requires $80ms$. Assuming a nominal $12V$ battery voltage and 2.08Ω resistance, the peak power consumption is $69.23W$, and the total switching energy is $2.5J$ per face.

4.4 Discussion

4.4.1 Advantages

The EP-Face has a number of desirable qualities. Connection and disconnection are nearly instantaneous, requiring only $160ms$ for a connect-disconnect cycle. Most other connectors require time on the order of seconds or minutes (Table 1). The energy required is also small,

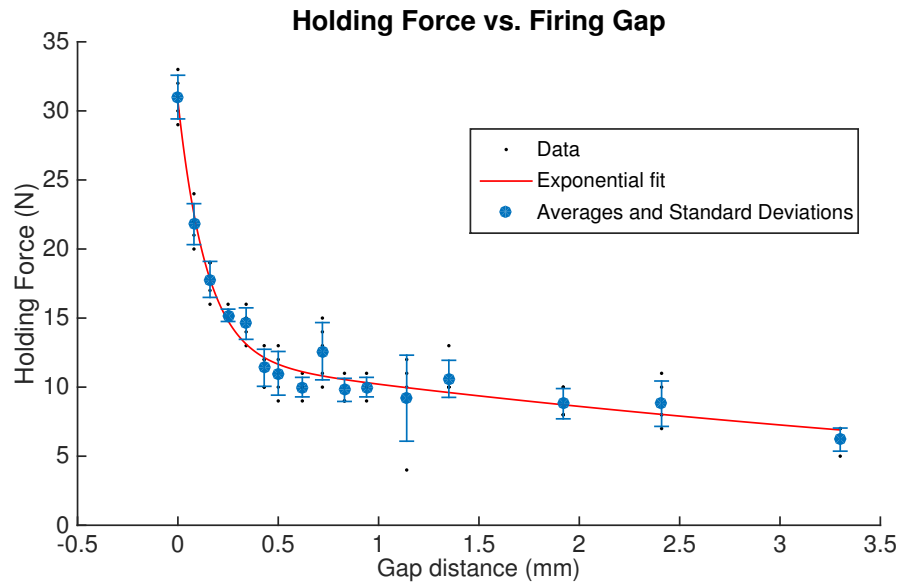


Figure 10: Plot of Holding Force vs. Gap at Firing. Exponential fit: $y = 18.63 \exp(-7.019x) + 12.08 \exp(-0.1697x)$

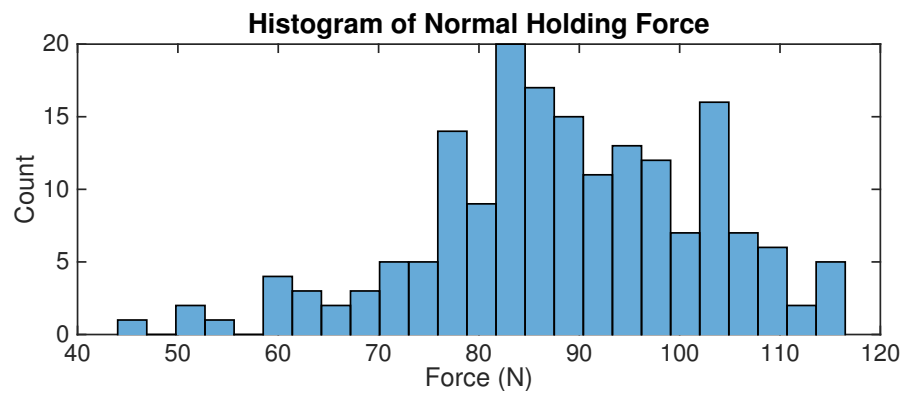


Figure 11: Histogram of Holding Forces Under Normal Load. Mean=88.4N, Std=13.9N.

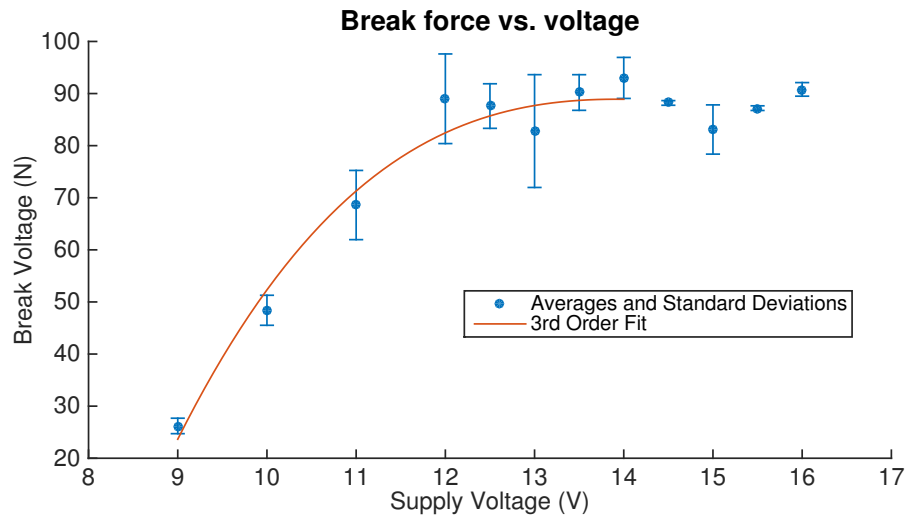


Figure 12: Face Break Force vs. Supply Voltage. Cubic fit: $y = 0.313x^3 - 14.3x^2 + 215x - 981$

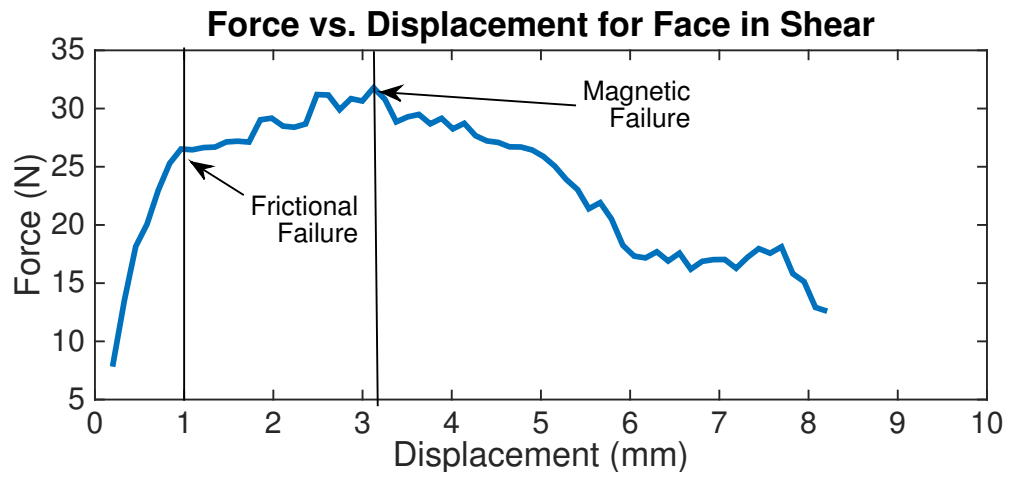


Figure 13: Force vs. displacement in shear. Peak force of 31.8N at displacement of 3.1mm. Note while static friction failure appears to occur at a displacement of 1mm, this large displacement is due to deformation (slop) of the module, not movement of the magnets.

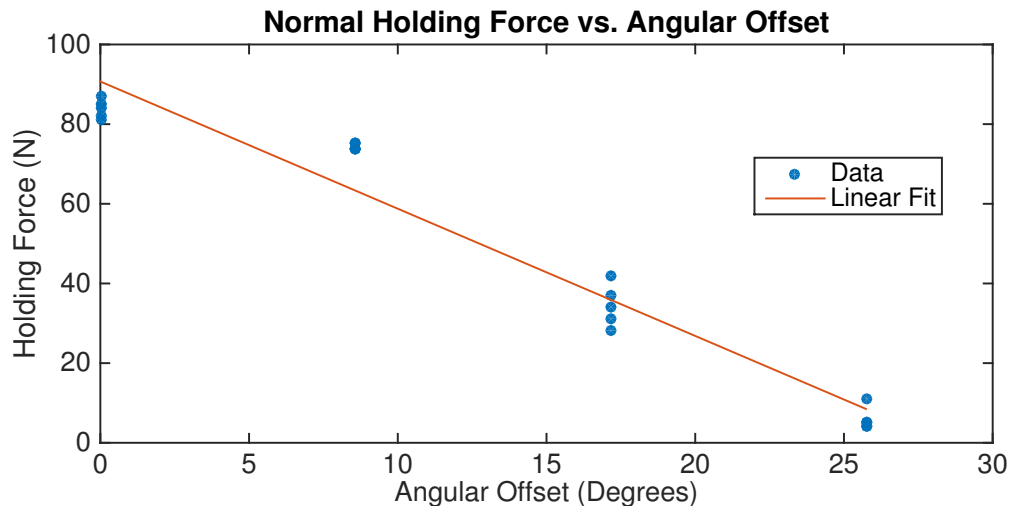


Figure 14: Normal Holding Force vs. Angular Offset.
Linear fit: $y = -3.194x + 90.7$

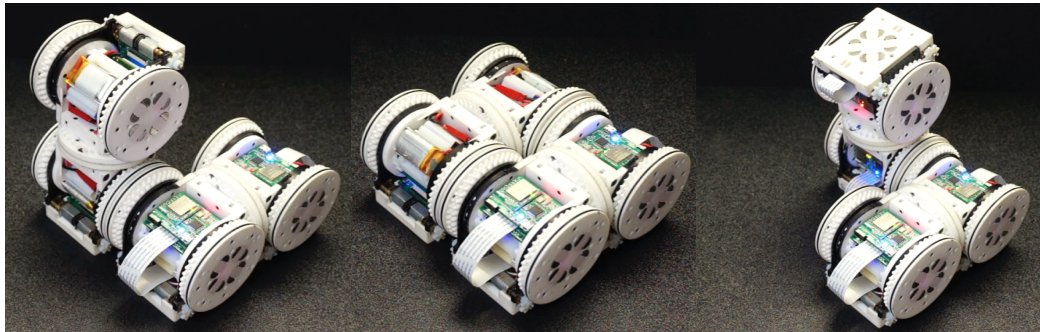


Figure 15: Slide-by docking is made possible by the thin profile of the EP-Face.

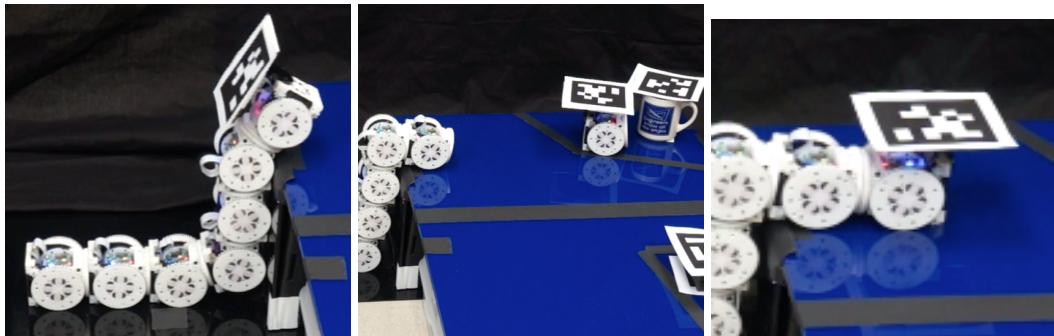


Figure 16: Ledge exploration. (Left) Seven-module snake lifts its head to the top of a 3-module high ledge. (Center) Head module detaches, and explores the surface. (Right) Head module autonomously reattaches, and snake descends.

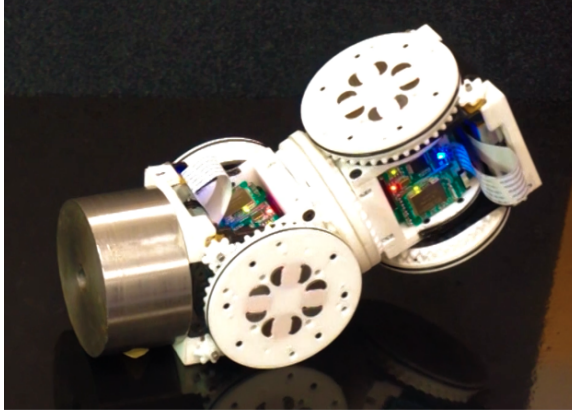


Figure 17: SMORES-EP module moving a 1kg metal block while lifting another module in the air.

$2.5J$ per state-change, as compared to $3.75J$ for the SINGO connector [84]. The time and energy costs of EP-Face connection are vanishingly small in comparison with movement actions SMORES-EP can perform.

The forces supported by EP-faces are comparable to other magnetic connectors (Table 1), and sufficient to perform tasks such as ledge climbing (Fig. 16). Under normal loading, it supports the weight of 17 modules. Additionally, since the connection is magnetic, SMORES-EP modules have demonstrated the ability to manipulate ferrous objects as heavy as 1kg (Figure 17).

The connector is low-profile, with the magnets protruding only $0.5mm$ from the face. The interior thickness is also small (magnets are $10.2mm$ tall, and total thickness including slip ring is $20mm$). Many existing connectors are thicker, sometimes comprising a significant fraction of the total module volume (ATRON, SMORES [17, 69]).

Latching and unlatching actions require no moving parts, and impose no constraints on module movement. Many other connectors are mechanically complex, with moving parts that can break or wear over time (ATRON, SMORES, SINGO, MTRANvIII [17, 47, 69, 84]). Additionally, while mechanical latches are usually stronger than magnetic connections, overloading can result in plastic deformation or fracture, permanently damaging the connector. In this sense, magnetic connections are more robust to overloading, allowing for the possibility of re-connecting after being overloaded.

Connectors with interlocking external features often require a specific angle of approach to connect. For example, the SINGO (SuperBot) and PolyBot connectors would not be capable of the kind of slide-by docking demonstrated by SMORES-EP in Figure 15, because their protruding features would collide [84, 106]. Similarly, the original SMORES connector cannot reliably perform slide-by docking because the permanent magnets get stuck in a local minimum configuration before the faces are actually mated. Because the EP-Face magnets lie in a single plane and can be switched off, there is no dependence on approach angle for docking; when two faces can be brought into contact, they can connect.

EP-Faces do not need to be perfectly aligned to connect successfully; the connector has a forgiving area of acceptance. As presented in the previous section, latching forces can draw two modules together through a gap of $4mm$ (normal to the faces), and $7mm$ parallel to the face. Even if the faces are held with an angular misalignment of 25 degrees, the

connection is strong enough to support the weight of one module. This can be compared to the tolerances of SINGO (max 6mm normal, 5mm parallel, or 5.7 degrees) and MTRANIII (max 2mm normal, 5mm parallel, or 10 degrees). The EP-Face is hermaphroditic and rotationally symmetric, with a pair of faces able to connect in four different orientations (0, 90, 180, and 270 degrees relative one another).

4.4.2 Disadvantages

With no mechanical features on the connection plane, shear loading is supported by static friction and magnetic restoring forces. Connector strength is significantly weaker under shear and torsional loading than normal loading.

The magnetic connection is also somewhat compliant, especially in shear and torsion. In the force vs. displacement plot for shear (Figure 13), we see a displacement of three millimeters before the maximum force is reached.

Connection strength is significantly affected by air gap when the magnets are fired, as demonstrated in the experiments. Fortunately, in some circumstances this is mitigated by the self-aligning properties and low cost of connection: once the magnets establish a weak connection, they can be fired again to strengthen the connection.

Because of this air gap sensitivity, the system needs to run in relatively clean environments. Outdoor environments with dirt and other debris that may be collected or stick to the magnet faces may reduce connection strength.

Some systems (such as SINGO [84]) are able to disconnect even if one module is unresponsive, facilitating removal of the broken module from the cluster. The EP-Face cannot do this: if one module becomes unresponsive with its magnets switched on, they will exert a holding force even if a connected module switches its magnets off.

4.5 Conclusions and Future

We introduced and characterized the EP-Face, an EP magnet-based connector for the SMORES-EP robot. We discussed its advantages relative to existing connectors, most notably its compactness, very fast connection speed (80ms) and wide area of acceptance. We also demonstrated how the EP-Face allows the SMORES-EP module to reconfigure and interact with its environment.

Overall, we think the advantages of the EP-Face make it a good option for modular robots the size of SMORES-EP. The main avenue of future work will be exploring ways to increase the shear and torsional strength of the connector. It may be possible to increase strength by adding a friction-enhancing surface finish to the faces. Another option is the addition of surface features that interlock to resist shear and torsion. Unidirectional features (alternating ridges and valleys in one dimension) could increase strength while still allowing slide-by docking.

Connector	EP face	SMORES	SINGO	ATRON	MTRAN 3	MTRAN 2	PolyBot 2
Type	EP Mag	Mag /Mech	Mech	Mech	Mech	Mag/ SMA	Mech/ SMA
Thickness (mm)	16.6	50	24*	50*	20*	20*	9.5
Gender	H	H	H	G	G	G	H
Orientations	4	2	4	1	4	2	4
Connection Cycle Time	160ms	2.3s	50s	4s	5s	30s	30s
Char. Strength	3.1	3	3.7	2.58	-	2.59	6.14
Connection Energy (Joules)	2.5	< 5*	3.75	< 5*	< 5*	200*	> 100*
Citation	-	[17]	[84]	[69]	[47]	[61]	[106]

Notes: H/G= Hermaphroditic/Gendered, *=approx,
Green=Best, Red=Worst, Characteristic strength values from [101].
SINGO thickness includes estimated 10mm for motor.

Table 1: Comparison of Connectors

Chapter 5

PaintPots

This chapter presents the PaintPot manufacturing process, a method for creating low-cost, low-profile, highly customizable potentiometers for position sensing in robotic applications. This chapter excerpts heavily from the author’s work in [93]. Credit is due to co-authors Daniel Edgar, Chao Liu, and Thulani Tsabedze, who contributed significantly to this work.

The PaintPot process uses widely accessible materials, requires no special expertise, and creates custom potentiometers in a variety of shapes and sizes, including curved surfaces. PaintPots offer accuracy and precision performance comparable with commercial (non-customizable) options through a calibration process that trades small computation for cost. We present manufacturing and calibration processes, as well as experiments that validate the accuracy, precision, and lifetime performance of PaintPots, comparable to commercial sensors. We also provide a case-study application in the SMORES-EP modular robot, and show how the PaintPot process can be used to create resistive surfaces capable of sensing position in 2D on planes and spheres.

5.1 Introduction

Nearly all robots with articulated joints require position sensing to precisely control their motions. Most commercial position sensors are available in a limited set of form factors, which constrain their positioning relative to the joint they are measuring. This can be a serious design challenge, especially in highly space-constrained applications like modular robots.

The PaintPots process is a novel method to create low-cost, low-profile, highly customizable potentiometers for position sensing. PaintPots can be made using widely accessible materials (spray paint and plastic sheet) and tools (laser cutter, or scissors), cost about \$1 USD to make in small batches, and require no special expertise to manufacture. They are highly customizable in terms of shape, size, and surface curvature, and can be directly integrated with existing plastic surfaces on parts. Once calibrated, they provide accuracy and precision performance comparable to off-the-shelf commercial potentiometers of similar cost.

PaintPots open robot design to a broader audience, enabling designers to tightly integrate custom position sensors into their robots, even if they would not normally have the expertise or funding to do so. Recent research into low-cost and printable robotics provides ample motivation for such sensors [55, 58]. Beyond the realm of low-cost robotics, PaintPots offer

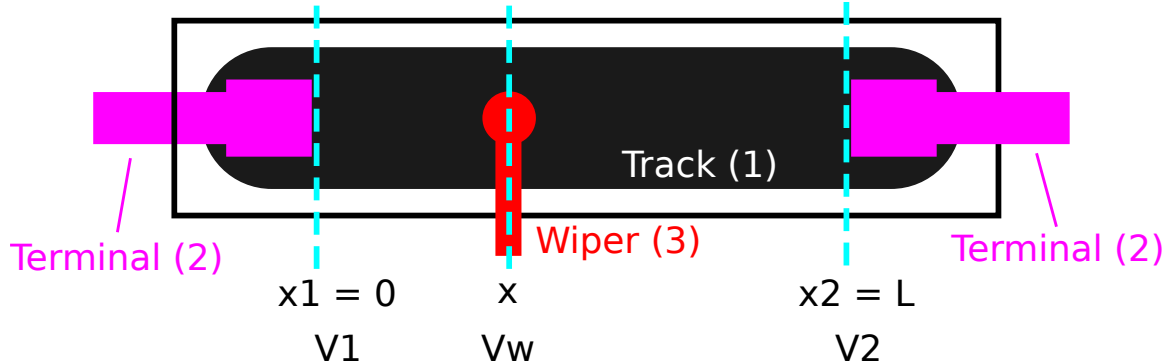


Figure 18: Potentiometer schematic showing the three parts (track, terminals, and wiper) as well as position and voltage labels.

sensing performance and customizability that makes them competitive with commercial potentiometers. As a case study, we present two PaintPots designs used in the SMORES-EP modular robot. Finally, we demonstrate how PaintPots enable 2D position sensing on arbitrary surfaces, including a plane and a sphere.

5.2 Related Work

5.2.1 Potentiometers

Potentiometers are three-terminal devices that can vary the resistance to a moving contact. The typical geometry is either a circular (*rotary pot*) or straight (*slide pot*). All potentiometers have the three basic components shown in Figure 18: (1) a resistive track, (2) fixed electrical terminals at the track ends, and (3) an electrical contact (wiper) that moves along the track surface. Most modern tracks are a continuous semiconductive surface made of graphite, ceramic-metal composites (cermets), conductive plastics, or conductive polymer pastes. PaintPots use an inexpensive carbon-embedded polymer spray paint for the track surface.

For position sensing applications such as robotic joint sensors, potentiometers are almost always configured as voltage dividers where each of the terminals are connected to a known voltage and the wiper voltage can be changed by moving between the two terminals.

Industrial processes are available that allow custom potentiometers to be created. Some electronics manufacturers offer inkjet-printed thick-films that can be deposited on printed circuit boards and other surfaces to form the tracks for custom potentiometer position sensors [4]. The primary distinguishing factors of the PaintPot method as compared to these processes are cost and time: these are industrial processes that often require thousands of dollars in up-front engineering fees, produce sensors that cost tens of dollars each, and have turnaround times of a week or more. Our method can be used by anyone, produces sensors that cost on the order of \$1 USD per unit, and allows rapid iteration (limited only by the drying time of the paint).

5.2.2 Ubiquitous Electronics

Recent work on ubiquitous computing and robotics technologies often leverages rapid prototyping technologies to allow electronics to be integrated into everyday objects at low cost. Miyashita *et al.* [57] introduce self-folding printable resistors, capacitors, and inductors made of cut sheets of aluminum-coated polyester film (mylar). An accordion-like variable resistor is presented, whose resistance changes as the folded layers are compressed and expanded. Kawahara *et al.* introduce Instant Inkjet Circuits [41], a technique for accurately printing low-resistance traces on sheet materials using silver nanoparticle ink deposited using a standard inkjet printer. This method has been used to explore designs for capacitive touch-sensitive sheets, which can be adapted to custom shapes by cutting with scissors [67]. Unlike PaintPots, a primary design concern for these silver-ink printed circuits is attaining sufficiently low trace resistances ($< 1\Omega$), as the printed conductors are intended for use as wires. In the case of PaintPots, relatively high resistances (on the order of $k\Omega$) are desirable, since the potentiometers are intended to be used as voltage dividers.

5.3 Background: Potentiometer characterization

5.3.1 Conformity (Accuracy)

The relationship between wiper position x and wiper voltage V_w (referring to Figure 18) is used to measure position. Let the function $V_w = f(x)$ be the *potentiometer model*. The degree to which $f(x)$ matches reality is referred to as *conformity*. Mathematically, *absolute conformity* is defined as the percent maximum deviation of the measured wiper voltage from the model over a defined travel range [98]:

$$\frac{|V_w(x) - f(x)|}{|V_w(x_2) - V_w(x_1)|} \leq C_{abs} \forall x \in [x_1, x_2]$$

Conformity measures accuracy: neglecting noise, it bounds the error relative to ground truth. Commonly, commercial potentiometers intended for position sensing are modeled as linear, that is: $f(x) = \left(\frac{V_2 - V_1}{x_2 - x_1}\right)(x - x_1) + V_1$. In this case, the term *linearity* is used in place of conformity.

5.3.2 Resolution

Resolution refers to the ability to register small changes in the value being measured. In practice, potentiometer resolution is often limited by the bit depth of the analog-to-digital converter used to read V_w . To compare PaintPot performance to off-the-shelf potentiometers, we are interested in measuring the intrinsic resolution of the device itself, independent of the analog-to-digital converter. Since potentiometers are analog devices, some manufacturers incorrectly list intrinsic resolution as infinite. In fact, the intrinsic resolution is determined by the mechanical noise properties of the strip material.

The *resolution limit* of a measurement system is w_{res} if there is an equal probability that the indicated value of any measurement whose actual value differs from a reference by less than w_{res} will be the same as the indicated value of the reference [3]. Novotechnik Inc. introduce the concept of Relative Gradient Variation (RGV), which can be used to determine the resolution limit w_{res} of a potentiometer [2]. RGV provides a measure of local deviations in resistance caused by material fluctuations at small length scales (typically micrometers).

Consider a one-dimensional straight-line potentiometer configured as a voltage divider, as shown in Fig 18. Let V be the wiper voltage, and x be the wiper position as it travels from $x_1 = 0$ to $x_2 = L$. As the wiper is moved, the *gradient* at position x can be defined $g = \frac{dV}{dx}(x)$. The *mean gradient* over a region $[x_1, x_2]$ can be defined $\bar{g} = \frac{V(x_2) - V(x_1)}{x_2 - x_1}$. RGV at $x \in [x_1, x_2]$ with window size w is defined:

$$RGV(x, w) = \frac{\frac{1}{w} (V(x + w/2) - V(x - w/2)) - \bar{g}}{\bar{g}}$$

RGV compares the local gradient in the window $x \pm w/2$ to the mean gradient of the sensor. Intuitively, RGV measures how much the behavior of the sensor in a small region (the local gradient in $x \pm w/2$) deviates from the nominal model (the mean gradient \bar{g}). Local deviations in gradient are due to fluctuations in the material properties of the potentiometer track at the microscale. As window size increases, we should expect RGV to decrease, since these small fluctuations will tend to cancel each other out (by the central limit theorem), and when $w = x_2 - x_1$, $RGV(x, w) = 0$ by definition. Conversely, as w decreases, RGV will increase.

Using RGV, we can compute the resolution limit w_{res} of a potentiometer. If for some sufficiently small w , $RGV(x, w) = 1$, this means the local variation in gradient is comparable to the mean gradient, so we know that in the region $x \pm w/2$, a change in the output signal of the potentiometer is just as likely to represent a fluctuation in local material properties as an actual movement of the wiper. Thus, $w = w_{res}$ is the resolution limit of the potentiometer [2].

Assume we have collected a dataset $\mathbf{V} = \{V_0, V_1, \dots, V_N\}$ which is a digitally sampled representation of $V(x)$ at points $\mathbf{X} = \{x_0, x_1, \dots, x_n\}$. Selecting a window size w , we may compute the mean absolute RGV for the sample:

$$|\overline{RGV}|(w) = \frac{1}{N} \sum_{i=0}^N |RGV(x_i, w)|$$

The resolution limit w_{res} is the window size for which $|\overline{RGV}|(w_{res}) = 1$.

Potentiometer performance is sometimes characterized in terms of “smoothness,” a metric which is qualitatively similar to resolution [98]. The standard procedure to measure smoothness applies a band-pass filter to the output signal of the potentiometer, and measures the maximum spike size produced when moving the wiper at a fixed speed over a set travel range. If the filter parameters and travel speed are standardized, smoothness measurements can be used to compare the performance of different potentiometers, but do not quantitatively measure resolution.

5.3.3 Hysteresis

Potentiometers can exhibit hysteresis due to friction between the wiper and track and compliance in the wiper. When the sensor approaches the same position from different directions, the actual position of the wiper on the strip (and therefore the output voltage) will be slightly different.



Figure 19: A bead of conductive paint applied beneath the screw head forms a good electrical connection with the track.

5.3.4 Lifetime

While there is no strict standard for potentiometer lifetime testing [2], accelerated lifetime testing typically involves repeatedly moving the potentiometer wiper through its full range until failure, and is reported in number of cycles.

5.4 Design and Manufacturing

5.4.1 Design Overview

Referring to Figure 18, the distinguishing feature of a PaintPot is the resistive track surface, which is made of conductive spray paint. We use MG Chemicals Total Ground conductive paint [56], which is an off-the-shelf carbon-embedded spray paint easily applied by hand. Any non-conductive material can serve as track substrate. The paint adheres well to most plastics, making it possible to paint tracks directly onto existing parts by masking off a region with tape. Acrylonitrile butadiene styrene (ABS) plastic works particularly well, because the paint chemically etches the surface to form a durable resistive coating [56]. ABS sheet can also be cut to precise shapes in a laser cutter, allowing custom tracks to be precisely cut in a wide range of shapes and sizes.

Connecting electrical terminals to the painted surface can be difficult, because wires cannot be soldered and crimp connections would likely crack the painted surface. Good electrical terminals can be created by mounting zinc-coated screws at the ends of the resistive strip. As shown in Figure 19, conductive paint is applied beneath the screw heads before fully screwing them in. Leaded solder adheres to zinc-coated screws, allowing wires to be easily attached and detached.

The wiper can be chosen depending on the application. Wipers with high contact pressure should be avoided, as they may scratch the paint. Larger contact surface area also reduces contact resistance, improving signal quality. The Harwin S1791-42 EMI Shield Finger Contact serves as a good wiper for our PaintPots [33]. The wiper is a 4mm high gold-plated tin spring contact with a 1.45x2.05mm contact area, and a contact force of 1N (mounted at a 3mm working height).

5.4.2 PaintPots used in SMORES-EP

Our decision to use PaintPots in SMORES-EP was driven by the tight space constraints inside the module. Absolute position sensing was necessary on all DoF. Optical track en-

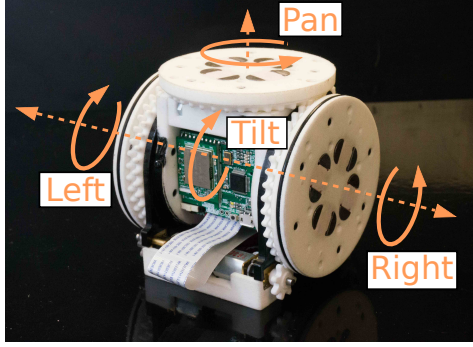


Figure 20: SMORES-EP module with labeled joints. The module is the size of an 80mm cube.

coders were considered for the side wheels, but not enough space was available to fit multiple gray code tracks to measure tilt position. PaintPots proved to be a versatile, robust, and accurate solution.

Wheel PaintPots

The wheel PaintPots, shown in Figure 21, have a circular track and two wiper contacts. They allow continuous rotation and provide position information over the full 360° range of the left, right, and pan joints. The annular geometry allows the slip ring to fit through the center. Tabs on the track extend into the center of the circle to provide space for the terminal contacts. The V-shaped gap provides enough space for the wipers to pass from one side of the track to the other without contacting both simultaneously (which would cause a short circuit).

The two wipers (Harwin S1791-42 [33]) are mounted on a PCB above the track, as shown in Figure 21. Using two wipers at a 50-degree angle to one another ensures that at least one wiper contacts the track even if one is in the gap, providing 360° of position sensing. To decide which wiper to use at a given time, we apply a simple rule. Angles are measured in the range $-\pi < \theta \leq \pi$, and since the two wipers are centered about the gap when $\theta = \pi$, we use one wiper if $\theta_{prev} < 0$ and the other wiper if $\theta_{prev} \geq 0$.

The track substrate is 0.79mm thick ABS sheet. To facilitate easy mounting, a layer of double-sided adhesive is applied to the back of the ABS sheet before cutting. Tracks are cut in batches in a laser cutter. After cutting, each track is gently sanded, removing plastic debris from laser ablation and providing an even rough surface ideal for paint adhesion, and then wiped with water to remove dust.

Tracks are hand-painted using spray cans of Total Ground conductive paint. Three coats of paint are applied, with five minutes of drying time between coats, following the painting guidelines in the datasheet [56]. Strips are allowed to dry for 24 hours before use, to ensure maximum durability. The total thickness of the sensor is 4.1mm, including the adhesive layer, ABS sheet, paint, and wipers.

Strips are mounted in a mated groove in a 3D-printed chassis as shown in Figure 21. The chassis has a raised triangular feature that mates with the gap in the strip, so that the wipers remains at the same level as they pass through the gap region. Zinc-coated screws ($m1.6 \times 6mm$) are used for electrical terminals, as described in Section 5.4.1. The measured

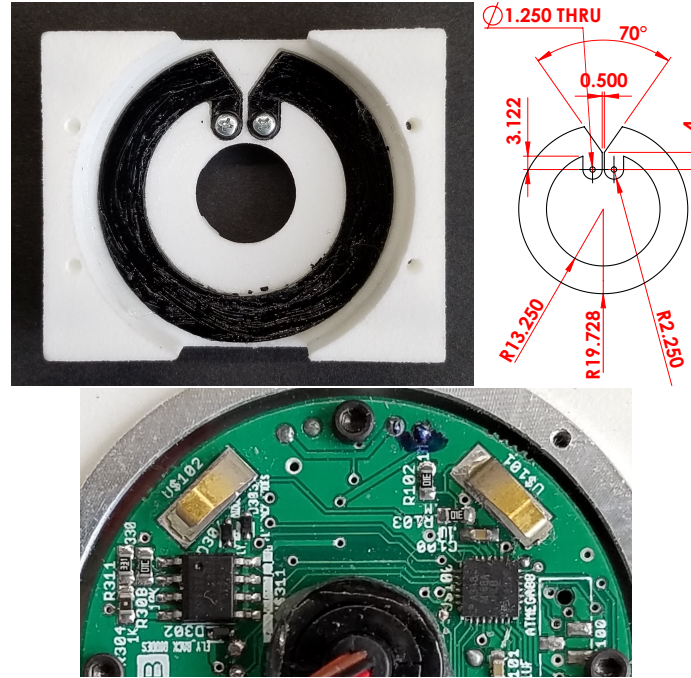


Figure 21: Top Left: Wheel PaintPot installed in chassis. Top Right: Drawing of wheel PaintPot with dimensions in mm. Bottom: Circuit board used with wheel PaintPots showing Harwin S1791-42 wipers mounted at a 50° angle.

terminal-to-terminal resistance of wheel PaintPots is between $2k\Omega$ and $20k\Omega$, depending on the thickness of the paint. Before use, a coat of petroleum-based grease is applied to the track surface.

Tilt PaintPots

The tilt PaintPots, shown in Figure 22, have resistive tracks with cylindrical curvature about their axis of rotation. A single wiper (Harwin S1791-42 [33]) contacts the track and measures position through the full 180° of motion of the tilt joint.

The track geometry of the tilt PaintPot makes very efficient use of space inside the SMORES-EP module. To our knowledge, no off-the-shelf potentiometers replicate this unusual non-planar shape. Some off-the-shelf slide pots can bend into curves [5], but come in predefined widths and lengths, making them difficult to mount in the module.

Tilt PaintPots have the same ABS/adhesive substrate as wheel PaintPots, and similar screw contacts. They are mounted to the 3D printed chassis before painting, allowing them to be painted in their final curved shape. This is preferable to painting flat and then bending: bending the paint after it has dried causes cracks to form, increases the resistance (three orders of magnitude) and causes a non-smooth variation of voltage along the length of the track. The terminal-to-terminal resistances of our tilt PaintPots fall between $3k\Omega$ and $10k\Omega$.

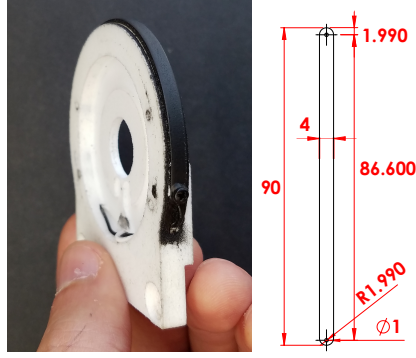


Figure 22: Top: Tilt PaintPot installed on chassis with cylindrical curvature (28.5mm radius). Bottom: Drawing of tilt PaintPot track (laid flat) with dimensions in mm.

5.4.3 Cost

PaintPots are inexpensive. The most expensive components in the SMORES-EP wheel and tilt PaintPots are the Harwin S1791-42 wipers, available from Digikey.com for \$0.35 USD in quantities of 100. MG Chemicals Total Ground spray paint can be purchased from Amazon.com for \$16 USD, and 0.79mm ABS sheet can be purchased from McMaster.com for \$3.70 USD per square foot. Based on these prices, materials for our wheel PaintPots cost \$1.05 USD, and tilt PaintPots cost \$0.70 USD (including material wasted during manufacturing). After quality control testing (described in Section 5.6), we yield about 75% of our wheel PaintPots and 90% of our tilt PaintPots, making the effective materials costs \$1.40 USD and \$0.78 USD, respectively.

5.5 Calibration

As discussed in Section 5.2, potentiometers are typically modeled as linear. Close adherence to the linear model is achieved by ensuring that resistivity is constant along the track, which involves ensuring uniform geometry, thickness, and material properties of the track. This requires careful quality control, which is expensive.

As an alternative, a calibration process can be used to achieve good performance. Our PaintPots are manufactured using a low-cost process (hand spray painting) without significant process control, and are somewhat nonlinear. By relying on a calibration process, we effectively trade manufacturing cost for additional computation, and achieve performance comparable to off-the-shelf potentiometers of greater cost.

5.5.1 Ground-Truth Data: AprilTags

AprilTags are an open-source, inexpensive, marker-based motion capture system, requiring only a camera, paper tags, and open-source software [68]. Unlike many position measurement devices (like shaft encoders), they do not require mechanical fixturing: tracking two rigid bodies (PaintPot track and wiper) only requires attaching paper tags to each body. Mechanical fixturing is particularly difficult for SMORES-EP modules, which have five independently moving rigid bodies.

During calibration, a PaintPot is moved through its entire range of motion (360° for wheel, 180° for tilt) in both directions, while voltage and ground-truth angle data are

Table 2: Error metrics for PaintPots

	Tilt PaintPot	Wheel PaintPot
RMS	$1.96^\circ \pm 0.10^\circ$	$2.92^\circ \pm 1.39^\circ$
Median	$-0.10^\circ \pm 0.28^\circ$	$0.06^\circ \pm 0.29^\circ$
Max	$4.83^\circ \pm 2.25^\circ$	$7.70^\circ \pm 3.56^\circ$
Conformity	$2.68\% \pm 1.25\%$	$2.14\% \pm 1.0\%$

recorded. The data rate is limited by the speed of the AprilTag software, which runs at about 12hz. Calibration takes about 50 seconds, during which about 600 datapoints are gathered.

5.5.2 Model fitting

While the voltage data from our PaintPots is often nonlinear, it does tend to be smooth and monotonic. As a result, the first order (linear) model typically used to model potentiometers is insufficient, but a significantly more complex model is not necessary to capture the variance. We found that a third-order polynomial ($X_w = a_3V_w^3 + a_2V_w^2 + a_1V_w + a_0$) provides a suitable model. In addition to good prediction performance (discussed in Section 5.6.1), third-order polynomials can be accurately and quickly computed with the floating-point unit on the SMORES-EP microcontroller (STM32f303).

While third-order polynomials provided good performance for our applications, alternative models (such as piecewise linear interpolation) could also be explored.

5.6 Performance

5.6.1 Accuracy

Given an N -sample dataset consisting of estimated angles $\hat{\Theta} = \{\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_N\}$ and ground truth angles $\Theta = \{\theta_1, \theta_2, \dots, \theta_N\}$, define the *error* to be $E = \{e_i = \theta_i - \hat{\theta}_i \mid i \in [1, N]\}$. We compute three error metrics for each dataset. *Root-mean-squared (RMS)* error is a measure of average error magnitude over the entire dataset, computed: $E_{RMS} = \sqrt{\sum_{i=1}^N e_i^2}$. *Median error* E_{med} is the median of all e_i . *Maximum error* E_{max} is the maximum error magnitude after applying a median filter (with window size 3) to remove electrical noise, $E_{max} = \max_i |\text{medfilt}(e_i)|$.

These error metrics are listed in Table 2 for a population of 11 wheel PaintPots and 16 tilt PaintPots. Conformity values are computed by dividing the maximum error by the angular travel range for each PaintPot (360° for the wheel, 180° for tilt). The conformity values for tilt and wheel PaintPots are 2.68% and 2.14% respectively, making them competitive with off-the-shelf potentiometers of similar cost. Section 5.6.5 provides a comparison with commercial potentiometers.

To guarantee consistent performance, every PaintPot used in SMORES-EP is evaluated during calibration. Any PaintPot with $E_{RMS} \geq 6.5^\circ$, $E_{med} \geq 2^\circ$, or $E_{max} \geq 20^\circ$ is considered out-of-spec, and is discarded (Estimated to be 25% of wheel tracks and 10% of tilt tracks).

5.6.2 Resolution

The resolution limit of our potentiometers was obtained using the experimental setup shown in Figure 24. It consists of an 80mm long slide PaintPot with the wiper mounted on a linear stage whose position is controlled by a servo-driven micrometer. The PaintPot is configured as a voltage divider, with terminal voltages of 0V and 12V.

Four datasets were gathered with the servo turning at a constant rate of 0.066 revolutions per second, corresponding to a wiper travel rate of $41.91\mu\text{m}$ per second. Data was gathered for approximately 5 seconds, for a total travel distance of about $200\mu\text{m}$. All four datasets traverse the same region of the strip. Voltage was sampled with an oscilloscope range of 40mV at a rate of 0.004s, corresponding to a travel distance of $0.168\mu\text{m}$ per sample. $|\overline{RGV}|(w)$ was computed for each dataset using window sizes ranging from 30 samples ($5.03\mu\text{m}$) to the length of the entire dataset in increments of 1 sample (Figure 23). Results from all four datasets agree closely. Averaging over all datasets, we compute $w_{res} = 8.63\mu\text{m}$, with a standard deviation of $0.216\mu\text{m}$ (3%). For reference, some high-precision potentiometers from Novotechnik have $w_{res} = 1.5$ to $3.5\mu\text{m}$ [65].

The small resolution limit of PaintPots means that the properties of the wiper and track will not be the limiting factor in precision for many applications. In SMORES-EP, the limiting factor is the ADC bit depth (10 bits, or $84\mu\text{m}$ for the 86.6mm long tilt PaintPot). To reach the material resolution limit, 14 bits of ADC depth would be required.

5.6.3 Hysteresis

PaintPot hysteresis was measured using the same experimental setup for RGV shown in Figure 24. To test hysteresis, the wiper was set to an initial position using the micrometer. The oscilloscope ground bias voltage was then adjusted to bring the measured wiper voltage as close as possible to zero, allowing the voltage scaling to be set as small as possible.

The wiper was then moved to the right 6.3mm, moved back to the zero point, and allowed to sit for two seconds before voltage was recorded. The procedure was then repeated, moving the wiper to the left instead of the right. The entire procedure was repeated 10 times. Five such experiments were conducted at five different initial positions on the strip. The hysteresis voltage is: $V_h = \frac{1}{2} |V_L - V_R|$ where V_L and V_R are the voltages after moving left and right during a trial.

The average hysteresis voltage over 30 total trials was 0.49mV with a standard deviation of 0.34mV (70%). Converting to an equivalent distance (by multiplying by the average slope $\frac{\Delta V}{\Delta X}$), we find a hysteresis distance of $10.1\mu\text{m}$, with a standard deviation of $7.1\mu\text{m}$. Like the resolution limit, the small hysteresis of this wiper and track is unlikely to be the limiting factor in overall precision for many applications.

5.6.4 Lifetime

While PaintPots are not intended to be long-life sensors for industrial purposes, adequately long lifetime is required for even low-cost applications. Wheel PaintPot lifetimes were evaluated by fixing a DC gear motor to the outside of a SMORES-EP face, spinning at 167 RPM. Every hour (after 10,020 revolutions), data is collected and evaluated according to the criteria presented in Section 5.5 to determine if the PaintPot still meets our minimum standards for use.

Table 3 shows lifetime tests from seven wheel PaintPots. Tracks from group A have three layers of paint, and were hand painted in small batches (1×4 grid of tracks), allowing

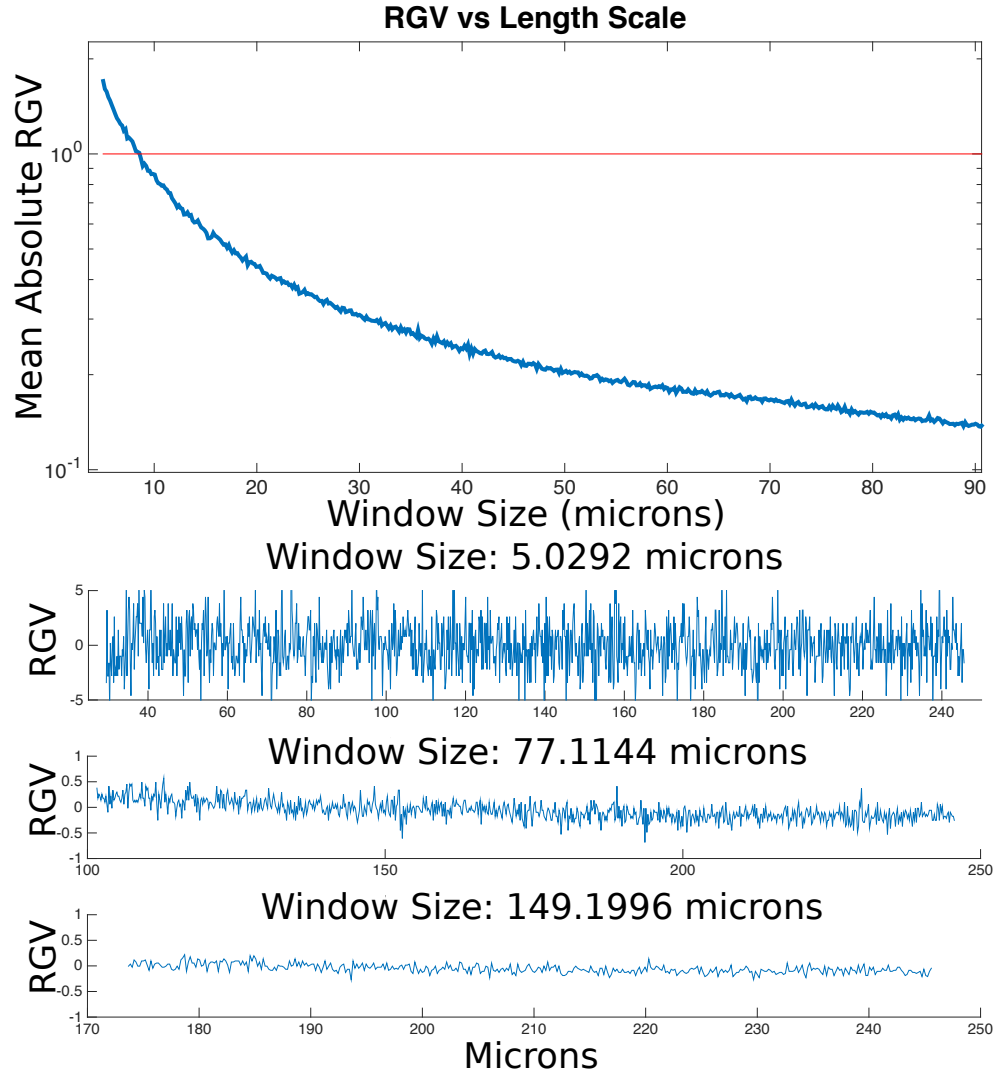


Figure 23: Top: Plot of mean absolute RGV with increasing window size. Red line indicates resolution limit, $\overline{|RGV|}(w) = 1$. Right: Bottom of RGV computed on the same dataset with different window sizes. We can see that the data becomes smoother with increasing window size.

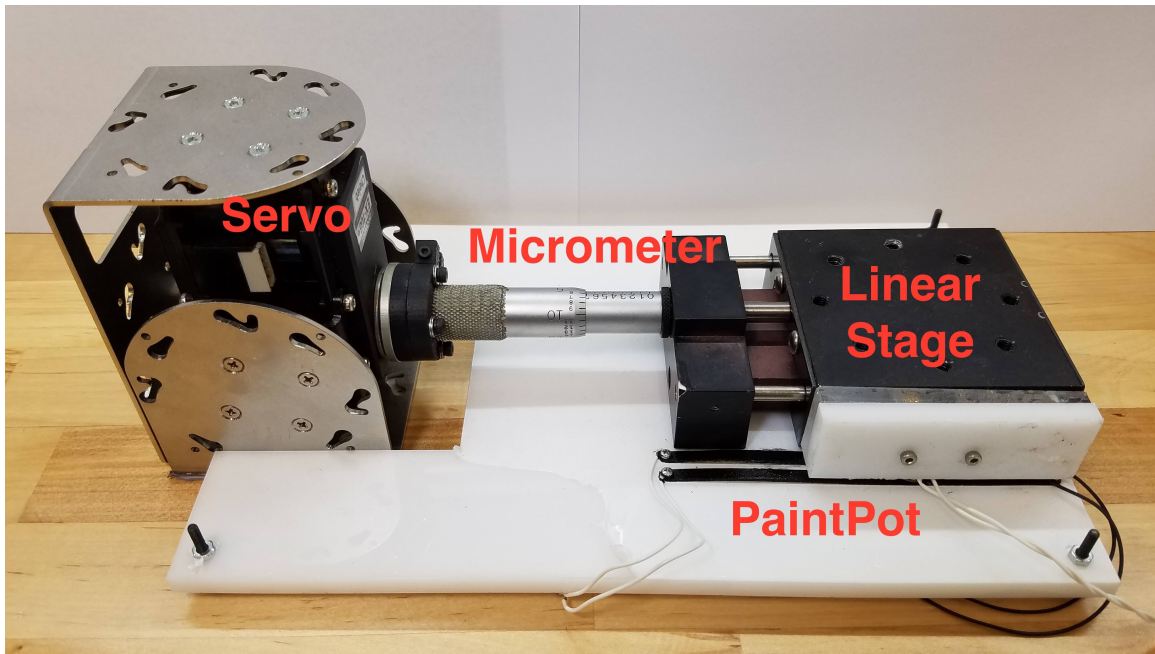


Figure 24: Testing setup used to evaluate RGV and hysteresis.

Table 3: Track lifetimes

#	Cycles (x1000)	Group	Failure mode
1	380	A	Wear through
2	190	A	Wear through
3	60	B	Local Pitting
4	50	B	Local Pitting
5	40	B	Local Pitting
6	20	C	Local Chipping
7	10	C	Local Chipping

the painter to carefully control the thickness of each layer of paint. Tracks in group B were painted in large batches (4×9 grid), making it more difficult to control paint quality. When tracks from group B failed, it was due to visible pitting in the top layer of paint. When the wiper hits these pits, the signal becomes noisy, and falls outside the acceptable bounds for maximum error. Tracks from group A were much more durable, lasting hundreds of thousands of cycles, and typically exhibiting an even wear pattern over the track.

In light of these results and our experience with the painting process, we hypothesize that group B had regions of thick paint that are not well bonded to the ABS surface, creating “soft spots” that wear away more easily. This hypothesis is further supported by the results from group C, which were painted with five coats of paint rather than three. Both tracks from group C failed when paint chipped off the track after a relatively small number of cycles. Based on these experiments, we recommend using a maximum of three coats of paint, and taking care to apply paint evenly.

Lubrication can also contribute to the longevity of PaintPots. Without lubrication, wheel PaintPots can fail at under 10,000 cycles. Silicone-based dielectric grease and petroleum jelly lubricants were found to be equally effective. Tracks tested in Table 3 were lubricated with petroleum jelly.

5.6.5 Comparison to Commercial Potentiometers

PaintPots can be tailored to the needs of an application at a cost of around \$1 USD, while achieving performance of more expensive potentiometers with similar features. Figure 25 plots cost versus linearity for 20 potentiometer position sensors with features similar to our wheel PaintPot (continuous rotation and 360° sensing range). The wheel PaintPot (red square) offers good conformity at a lower cost than the majority of available potentiometers. Its disadvantage is a shorter lifetime, which in many robotics applications is not a major concern. In the case of SMORES-EP, none of these other potentiometers had a form factor that could meet the other design requirements (such as a through-hole large enough for the slip ring in the middle of the face).

When considering cost and conformity, it is important to note that PaintPots rely on a calibration function, which requires additional computation. The calibration process presented here could be used with other potentiometers to increase performance, but in essence demonstrates the computation for cost trade-off.

5.7 Two-Dimensional PaintPots

The customizability of PaintPots enables many interesting sensing modalities. A spherical PaintPot is created by painting a plastic sphere (Figure 26), and can be used to sense the position of a wiper on its top half. The sensor has four terminals as shown in Figure 27. Position sensing is done in two alternating steps. In step 1, terminals A0 and A+ are held at ground and 3.3V respectively, while B0 and B+ are left floating. This creates a voltage field that varies linearly with arc length from A0. By reading the voltage, the wiper is localized to a circle on the surface. In step 2, B0 and B+ are held at ground and 3.3V while A0 and A+ are left floating, localizing the wiper to a different circle. The position of the wiper is the intersection of the two circles within the top hemisphere. If a third pair of electrodes were used, the wiper could be localized on the entire sphere.

A flat-sheet PaintPot (Figure 26) uses a similar method to determine the X-Y position of the wiper. Four contacts positioned at the corners of the sheet are alternately activated and

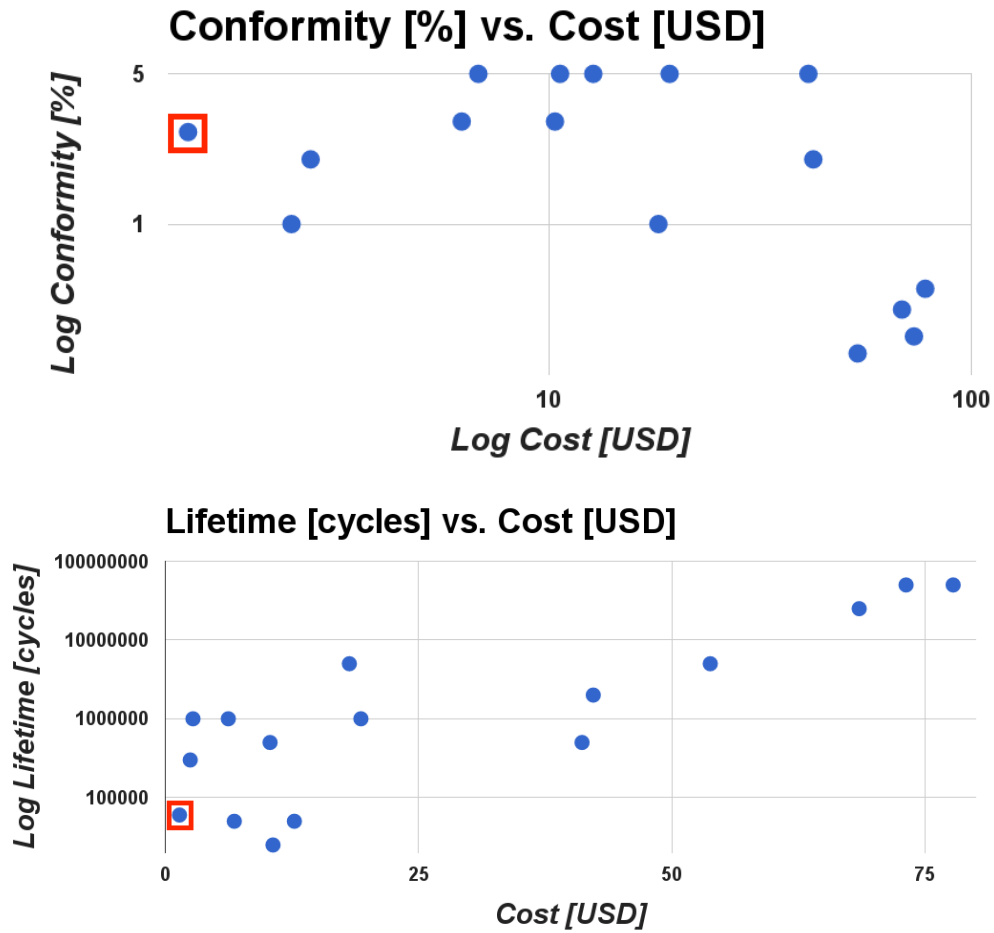


Figure 25: Plots of cost vs. conformity and cost vs. lifetime for commercial potentiometers with features similar to the wheel PaintPot (360° sensing range and continuous rotation). PaintPot marked with red square.



Figure 26: Left: Spherical PaintPot that senses position on the top hemisphere. Right: Flat-sheet PaintPot capable of sensing the X-Y position of the wiper.

deactivated in a similar way to the sphere (Figure 27). Each sensing step localizes the wiper to a horizontal or vertical line. For a simple cartesian mapping from voltage to position, ideally two full sides of the rectangle would be held at known voltages. However, wiring the entire side would create short-circuits at the corners. Instead, two points along each side are used, which creates nearly-even voltage field lines in the middle of the sheet.

The 2D sheet PaintPot can be used as a touchpad to capture writing, as demonstrated in the accompanying video. The surfaces are durable enough that the stylus (a multimeter probe) does not scratch them under normal writing pressure. The sphere and sheet PaintPot each cost about \$1 USD to make. The performance of these 2D PaintPots could be improved through calibration. Similarly to the procedure employed for the 1D PaintPots, the sheet or sphere could be calibrated by measuring voltages at known coordinates on the surface, and fitting a parametric function for each coordinate as a function of the two measured voltages.

5.8 Conclusion

We presented a method to create custom potentiometers for position sensing at low cost. The manufacturing process uses widely accessible materials, requires no special expertise, and can create potentiometers in a variety of shapes and sizes, including curved surfaces. This enables designers to integrate custom sensors into their designs, even if they would not normally have the expertise or funding to do so.

Our calibration process is low-cost and adaptable. Once calibrated, PaintPots offer accuracy and precision on par with commercial potentiometers of comparable cost. We believe this makes them a competitive alternative to off-the-shelf potentiometers, even in high-performance applications.

PaintPots are not without disadvantages. The tracks have shorter lifetimes than commercial potentiometers. Wiper alignment is important: if a wiper contacts the track on its corner or edge, the pressure concentration can scratch the painted surface. Calibration allows good accuracy at low cost, but requires time during manufacturing, and more complex software. Time must also be spent identifying tracks that are up-to-spec for high-performance applications.

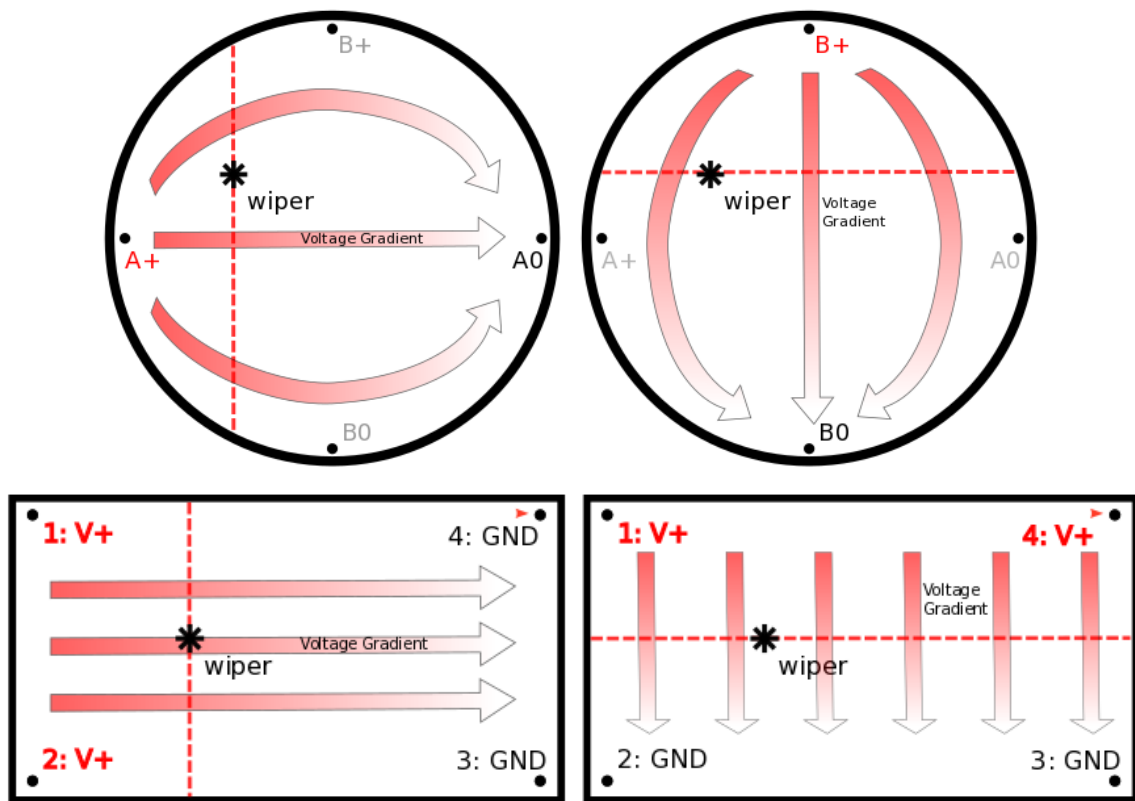


Figure 27: Top: Top-down view of voltage gradients on the sphere PaintPot. Bottom: Voltage gradients on sheet PaintPot.

In the future, well-established automated painting processes could greatly improve Paint-Pot consistency over hand-painting. While hard metal wipers proved the best choice for the SMORES-EP PaintPots (because of their low profile and low hysteresis), other types of wipers might be optimal for other applications. In particular, softer brush-type wipers might afford longer lifetimes.

Chapter 6

Design Embedding

This chapter formalizes the concept of modular robot design embedding, and presents an algorithm to automatically detect embedding. Embedding formalizes a notion of structural similarity between robot designs, which has implications for functional similarity. This chapter excerpts heavily from the author’s work in [52]. Credit is due to Yannis Mantzouratos and Prof. Sanjeev Khanna, who contributed significantly to the work presented here.

6.1 Introduction

One of the most interesting aspects of modular reconfigurable robots is the ability to transform into different shapes to adapt to needed tasks. Techniques to automatically determine which shapes and configurations can accomplish a task would make these systems more powerful. Most tasks are compositions of many sub-tasks: for example, an assembly task could be composed of many pick-and-place operations.

We refer to the automated, generative design of a modular robot from a task specification as *design synthesis*. In a generative system, it would be useful to build systems hierarchically, building subgroups of modules that achieve subtasks. As a first step towards design synthesis, we consider the following problem: can we efficiently determine if a subgroup of modules configured for a kinematic task can be realized in a larger group of modules configured for another kinematic task? For example, given a subgroup of modules that can function as a planar arm, is there a set of modules in a larger configuration that could serve the same function? We call this problem the *design embedding* problem.

In this chapter, we provide a formal definition for design embedding via topological and kinematic conditions, and a poly-time algorithm using dynamic programming and matching to efficiently detect when one design can be embedded in another design. The algorithm is intended to be run offline on a central computer. Information about embeddability can then be used to make decisions about the designs. For example, Section 6.7 discusses how kinematic behaviors can be translated from one design to another design that embeds it.

6.2 Related Work

Related approaches with modular robots in the past have included narrower optimization of specific kinematic linkages for manipulation problems [11], [104] as well as a selection approach, choosing the most appropriate configuration for a task from a given set of configurations [66]. In these cases, the robot would sense the environment for features and

select the most appropriate configuration from among a small set to reach a goal in a locomotion task. Behaviors have also been automatically generated by identifying functional substructures (e.g. knees) in modular robot designs [8].

A more general approach would look at configurations on a finer-grain scale. Since the system is already modular, analyzing the capabilities of assemblages by varying modules is natural. However, the number of possible arrangements of modules grows exponentially with the number of available modules, which makes the selection approach intractable. For very simple tasks such as locomotion in a line, fine-grain generative approaches have been achieved using evolutionary approaches [35].

Design synthesis has been studied in the context of automated machine design [24], [97]. However, in the most general sense this requires an understanding of the space of all tasks, and the relationships between module composition and their interaction with the environment, which is very broad.

In this work, we use graph representations of modular robots. Existing work in graph representations of modular robots includes recognizing if two full configurations are the same [71], identifying graph automorphisms [53], and recognizing identical substructures for efficient reconfiguration [38]. Our work distinguishes itself by including task implications on configurations, defining conditions to replicate the capabilities of a design by replicating its structure. To our knowledge, it is the first representation that captures the full kinematic structure of a modular robot design; in fact it can represent any acyclic kinematic structure composed of revolute joints.

6.3 Preliminaries

This section provides the basic graph-theoretical concepts and definitions that are used throughout the paper; a more elaborate exposition can be found at [18].

Let $G(V, E)$ denote an undirected graph, where V is a set of nodes, and $E \subseteq V \times V$ is a set of undirected edges. Given a subset $V' \subseteq V$ of the vertices, the subgraph induced by V' is given by $(V', \{(u, v) \in E \mid u, v \in V'\})$.

A *simple path* $v_1 \rightsquigarrow v_k = (v_1, v_2, \dots, v_k)$ in G is a sequence of distinct nodes in V such that for consecutive nodes v_i, v_j in the path, $(v_i, v_j) \in E$. The *length* of a path is just the number of edges it contains and the *distance* between two nodes $u, v \in V$ in G , denoted by $\delta_G(u, v)$, is the minimum length of a path from u to v . By convention, $\delta_G(u, u) = 0$ and $\delta_G(u, v) = \infty$ when such a path does not exist.

G is called a *tree* if each pair of nodes is connected by exactly one simple path. When a tree is rooted at a node $\tau \in V$, ancestors and descendants of $u \in V$ are defined as follows. Any $v \in V$ such that $\delta_G(\tau, v) < \delta_G(\tau, u)$ and path $v \rightsquigarrow u$ does not involve τ is called an *ancestor* of u with respect to τ . Similarly, any $v \in V$ such that $\delta_G(\tau, v) > \delta_G(\tau, u)$ and path $v \rightsquigarrow u$ does not involve τ is called a *descendant* of u with respect to τ . We denote the set of descendants of a node u with respect to τ as $\text{desc}(u, \tau) \subseteq V$. We will write $G[u \downarrow \tau]$ to denote the subtree of G that is rooted at u and contains exactly $\text{desc}(u, \tau)$, i.e., $G[\{u\} \cup \text{desc}(u, \tau)]$.

We denote immediate descendants by $\mathcal{N}(u, \tau) = \{v \in \text{desc}(u, \tau) \mid \delta_G(u, v) = 1\}$ and call them the *children* of u with respect to τ ; a node can have multiple children. An immediate ancestor is denoted by $\mathcal{P}(u, \tau) = \{v \in V \setminus \text{desc}(u, \tau) \mid \delta_G(u, v) = 1\}$ and is called the *parent* of u with respect to τ . Each node has one parent, except τ which has none. We can naturally

extend the above to *child* and *parent edges*, which will be denoted by $\mathcal{N}^e(u, \tau)$ and $\mathcal{P}^e(u, \tau)$ respectively.

Finally, a *rigid body* is a set of points capable of rotation and translation in Euclidean space. Each rigid body is defined by a *body frame* and *origin*. Reference frames are denoted with an uppercase calligraphic letter (e.g. \mathcal{W}), and vectors in boldface (e.g. \mathbf{v}). The position of point p relative point o in frame \mathcal{W} will be written ${}^{\mathcal{W}}\mathbf{r}_{p/o} \in \mathbb{R}^3$. The orientation of frame \mathcal{B} relative frame \mathcal{W} will be written ${}^{\mathcal{W}}R^{\mathcal{B}} \in SO(3)$.

6.4 Topological Embedding

6.4.1 Definitions and Statement of Main Result

We now formally introduce the graph representation of modular robotic designs that we will use throughout our discussion of topology, and present the notion of topological design embedding.

Definition 1. (*Unit Block*). A unit block $B = \langle \phi \rangle$ is an elementary rigid body capable of implementing a prespecified set of built-in functionalities $\phi \in \mathcal{F}$.

Built-in functionality is independent of topology; e.g., consider a block equipped with sensors, a processor unit or a battery. We define a partial order on unit blocks on a functional basis: $B_1 \preceq B_2$ if and only if $\phi_1 \subseteq \phi_2$.

Definition 2. (*Modular Robot Design*). Given a set of unit blocks \mathbb{B} , a robot design $D = \langle G(V, E), \beta \rangle$ defined on \mathbb{B} is a labelled, undirected graph G , where nodes of G correspond to unit blocks through $\beta : V \mapsto \mathbb{B}$, and edges between two nodes u and v represent a revolute joint connecting $\beta(u)$ to $\beta(v)$.

In Section 6.5.2, we will extend the definition of unit blocks to represent rigid bodies, and map edges to revolute joints that provide movement. For now, we postpone discussion of kinematics until the topological algorithm is explained completely.

Definition 3. (*Design Embedding*). Given two designs $D_1 = \langle G_1(V_1, E_1), \beta_1 \rangle$ and $D_2 = \langle G_2(V_2, E_2), \beta_2 \rangle$ defined on a set of unit blocks \mathbb{B} , and an injective mapping $f : V_1 \mapsto V_2$, we say that D_1 embeds in D_2 with respect to f , and write $D_1 \sqsubseteq_f D_2$, if and only if:

1. Functionality subsumption: $\forall u \in V_1$, we have $\beta_1(u) \preceq \beta_2(f(u))$.
2. Connectivity preservation: $\forall (u, v) \in E_1$, there exists a simple path $\pi_{uv} = f(u) \rightsquigarrow f(v)$ in G_2 .
3. Path disjointness: for any pair of edges with distinct endpoints $(u_1, v_1), (u_2, v_2) \in E_1$, the corresponding paths $\pi_{u_1 v_1}$ and $\pi_{u_2 v_2}$ in G_2 are vertex-disjoint. In addition, for any $(u, v_1), (u, v_2) \in E_1$, π_{uv_1} and π_{uv_2} share only $f(u)$.

In general, we refer to D_1 as the *subdesign* and D_2 as the *superdesign*. Where there is no chance of confusion, we omit f and write $D_1 \sqsubseteq D_2$.

Fig. 28 offers the intuition behind the definition. Condition (1) requires every vertex in the subdesign to map to a vertex of equal or superior functionality in the superdesign. Condition (2) preserves the connectivity of the subdesign once it is embedded: nodes which

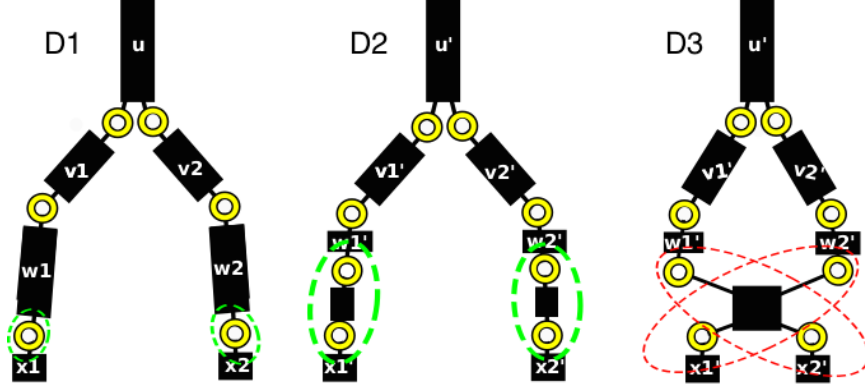
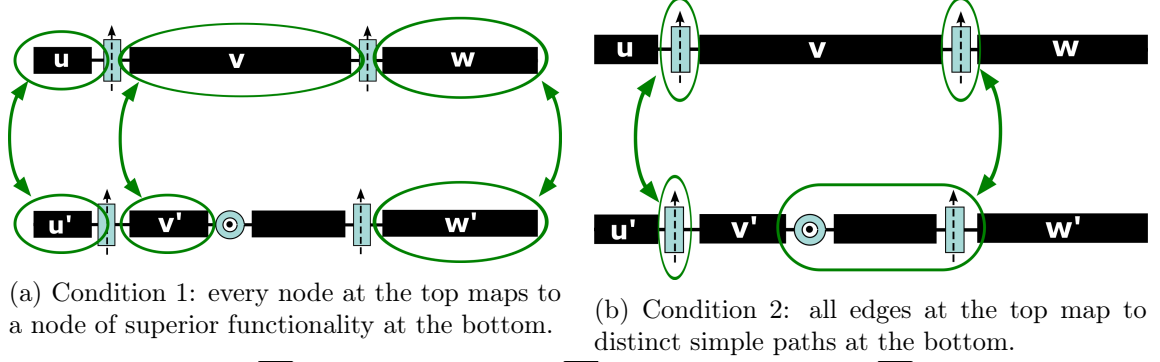


Figure 28: Topological conditions for embedding.

were able to interact through the joints can still do it, albeit maybe through longer paths. Finally, condition (3) ensures that degrees of freedom which are independent in the subdesign remain independent in the superdesign.

From a topological perspective, embeddability is equivalent to whether the subdesign is a topological minor of the superdesign; see [30] and references therein.

We are now ready to state our main result.

Theorem 1. *Given two designs $D_1 = \langle G_1(V_1, E_1), \beta_1 \rangle$ and $D_2 = \langle G_2(V_2, E_2), \beta_2 \rangle$ defined over a set of unit blocks \mathbb{B} , where G_1 and G_2 are trees of maximum degree d , there exists a deterministic algorithm that decides whether $D_1 \subseteq D_2$ in time $O(|V_1| \cdot |V_2| \cdot d^{2.5})$.*

Note that d is the maximum number of edges incident on any node. For most real robot applications, $d \leq 5$.

6.4.2 Outline of Algorithm

We now present a dynamic programming algorithm that decides whether $D_1 \subseteq D_2$ and if so, produces a mapping $f : V_1 \mapsto V_2$. We focus on obtaining a yes or no answer rather than the mapping itself; it can be reconstructed the regular dynamic programming way, by keeping track of the option we selected at each step in a separate array and backtracking; see [13] for details.

We will maintain a $|V_1| \times |V_2|$ truth table T , where $T[v_1, v_2]$ is true under a specified

rooting $\tau_1 \in V_1$, $\tau_2 \in V_2$ if and only if $D_1[v_1 \downarrow \tau_1] \subseteq D_2[v_2 \downarrow \tau_2]$. At the end of the algorithm, $T[\tau_1, \tau_2]$ answers whether $D_1 \subseteq D_2$ under τ_1 and τ_2 ; if the answer is negative, we repeat the process for a new rooting until we either get a positive answer or we exhaust all possible rootings, in which case we conclude that $D_1 \not\subseteq D_2$.

Initially, all entries are false. We subsequently proceed bottom-up, starting from the leaves of D_1 and moving gradually towards τ_1 . As the base case, we consider a leaf $v_1 \in V_1$ and check whether it embeds in the leaves of D_2 , setting the appropriate entries of T accordingly:

$$T[v_1, v_2] = \text{true} \iff \beta_1(v_1) \preceq \beta_2(v_2),$$

for all leaves $v_2 \in V_2$. Intuitively, this corresponds to checking whether a unary design consisting of v_1 is embedded in another unary design that contains only v_2 , in which case the only relevant embedding condition to check is functionality subsumption.

Subsequently, we move towards τ_2 by examining the parents of the leaves of D_2 ; for such a parent $p_2 \in V_2$, we set $T[v_1, p_2]$ to true if $\beta_1(v_1) \preceq \beta_2(p_2)$, or

$$\exists v_2 \in \mathcal{N}(p_2, \tau_2) \text{ such that } T[v_1, v_2] = \text{true}.$$

The intuition is that either v_1 is subsumed by p_2 , or it is subsumed by one of p_2 's children. We continue likewise until we complete the v_1 -th row of T and then repeat for another leaf of V_1 . At the end, we know exactly which nodes of D_2 can host the leaves of D_1 .

We are now ready to move up one level in D_1 as well. Let $p_1 \in V_1$ denote a parent of a leaf from the previous step. Starting from nodes that are at the same height in D_2 as p_1 is in D_1 , we set $T[p_1, p_2]$ to true in the following two cases.

The first one, which corresponds to p_1 embedding directly in p_2 , is triggered if $\beta_1(p_1) \preceq \beta_2(p_2)$ and in addition, there exists an assignment $M \subseteq \mathcal{N}(p_1, \tau_1) \times \mathcal{N}(p_2, \tau_2)$ of children of p_1 to children of p_2 such that

$$\begin{aligned} |M| &= |\mathcal{N}(p_1, \tau_1)|, \text{ and} \\ T[v_1, v_2] &= \text{true} \quad \forall (v_1, v_2) \in M. \end{aligned} \tag{6.1}$$

Essentially, these conditions make sure that every child of p_1 embeds in a unique child of p_2 .

The second case captures that while p_1 might not directly embed in p_2 in the way described above, it may still embed in one of p_2 's children, i.e.,

$$\exists v_2 \in \mathcal{N}(p_2, \tau_2) \text{ such that } T[p_1, v_2] = \text{true}. \tag{6.2}$$

Notice that since we are proceeding bottom-up, $T[p_1, v_2]$ is filled before $T[p_1, p_2]$, and therefore our computations are always well defined.

After the p_1 -th row of T is calculated, we repeat for all nodes at the same height, and then move upwards exactly the same way until we hit τ_1 . Essentially, we fill our table by performing a reverse pre-order traversal on G_1 , where at each step of the traversal we process the nodes of G_2 in reverse pre-order as well. Notice that it is not necessary to process all the leaves first, which we did in our exposition for simplicity, as long as we process a node only after having dealt with all of its descendants.

It only remains to show how to compute $M \subseteq \mathcal{N}(p_1, \tau_1) \times \mathcal{N}(p_2, \tau_2)$ from equation (6.1): we construct a bipartite graph with $\mathcal{N}(p_1, \tau_1)$ on the left side, $\mathcal{N}(p_2, \tau_2)$ on the right, and

an edge (v_1, v_2) if and only if

$$v_1 \in \mathcal{N}(p_1, \tau_1), v_2 \in \mathcal{N}(p_2, \tau_2) \text{ and } T[v_1, v_2] = \text{true}.$$

Intuitively, v_1 is connected to v_2 if and only if subdesign $D_1[v_1 \downarrow \tau_1]$ embeds in $D_2[v_2 \downarrow \tau_2]$. We subsequently compute a maximum cardinality matching [34]. At this point it is also possible to incorporate arbitrary user input that explicitly disallows embedding of particular nodes.

6.4.3 Formal Analysis

Lemma 1. (*Algorithm Soundness*). Given designs D_1 and D_2 , if the algorithm accepts then $D_1 \sqsubseteq D_2$.

Proof. Suppose that the algorithm accepts and let τ_1 and τ_2 reflect the roots of G_1 and G_2 at the time the positive answer was computed, and $f : V_1 \mapsto V_2$ be the suggested mapping. Relabel nodes of both graphs to reflect reverse pre-order, and let $V_1 = \{u_1, u_2, \dots, \tau_1\}$ and $V_2 = \{v_1, v_2, \dots, \tau_2\}$. We will use strong induction on u_i , $i = 1, \dots, |V_1|$.

For the base case, let $T[u_1, v_j]$ be true. Since u_1 is a leaf, $D_1[u_1 \downarrow \tau_1]$ contains only u_1 and the only relevant criterion from definition 3 is functionality subsumption. By construction, $T[u_1, v_j]$ is true if and only if the functionality of $\beta_1(u_1)$ is subsumed by either $\beta_2(v_j)$ or one of its children, and therefore, $D_1[u_1 \downarrow \tau_1] \sqsubseteq D_2[v_j \downarrow \tau_2]$, for any $j = 1, \dots, |V_2|$.

Now assume that for $i \leq k$, if $T[u_i, v_j]$ is true then $D_1[u_i \downarrow \tau_1] \sqsubseteq D_2[v_j \downarrow \tau_2]$, for all j . We are going to show that the same holds for u_{k+1} too. Indeed, fix an arbitrary v_j and suppose that $T[u_{k+1}, v_j] = \text{true}$. If u_{k+1} is a leaf, we are done by the argumentation above, so assume otherwise. Recall that there are two cases which could have forced the entry to be true.

If the first case triggered the truth assignment, then it must be that $\beta_1(u_{k+1}) \preceq \beta_2(v_j)$ and there exists a matching M between $\mathcal{N}(u_{k+1}, \tau_1)$ and $\mathcal{N}(v_j, \tau_2)$ such that (6.1) holds. Obviously the functionality subsumption criterion of embedding holds. By hypothesis and the fact that M covers all the children of u_{k+1} , connectivity is maintained - the subdesigns rooted at the children of u_{k+1} embed in children of v_j , and since $f(u_{k+1}) = v_j$, every edge $(u_{k+1}, u') \in E_1$ corresponds to a path utilizing $(v_j, v') \in E_2$, where $(u', v') \in M$. Finally, path disjointness is also maintained. By hypothesis, we only need to care about paths of the form $\pi_{u_{k+1}, u'} = f(u_{k+1}) \rightsquigarrow f(u')$, where $u' \in \mathcal{N}(u_{k+1}, \tau_1)$. Since $f(u')$ maps to a distinct child of v_j by definition of matching (say $f(u') = v'$), and $f(u_{k+1}) = v_j$, it follows that each $\pi_{u_{k+1}, u'}$ starts with a distinct edge of the form $(f(u_{k+1}), f(u')) = (v_j, v') \in E_2$. Paths cannot share any vertex later, since that would imply a cycle in G_2 , contradicting its tree structure. Consequently, all criteria of embedding are satisfied, and $D_1[u_{k+1} \downarrow \tau_1] \sqsubseteq D_2[v_j \downarrow \tau_2]$.

The second case follows by the same argumentation and a similar (nested) inductive argument on v_j . \square

Lemma 2. (*Algorithm Completeness*). Given designs D_1 and D_2 , if $D_1 \sqsubseteq D_2$ then the algorithm accepts.

Proof. Suppose that $D_1 \sqsubseteq_f D_2$ for $f : V_1 \mapsto V_2$. Let τ_1 be an arbitrary node in V_1 , $\tau_2 = f(\tau_1) \in V_2$ and consider running the algorithm with τ_1 and τ_2 as the roots. As before, relabel the nodes of both graphs to reflect reverse pre-order; we sketch an inductive argument on V_1 .

As a base case, notice that $D_1[u_1 \downarrow \tau_1]$, which consists only of u_1 , embeds into some $D_2[v_j \downarrow \tau_2]$ as a direct result of the functionality subsumption criterion, where $f(u_1) = v_j$. Therefore, $T[u_1, v_j]$ is trivially true.

For the inductive step, consider u_{k+1} and let $f(u_{k+1}) = v_j$. By the functionality subsumption criterion, $\beta_1(u_{k+1}) \preceq \beta_2(v_j)$. We only need to show that there exists a matching M satisfying (6.1) and we are done – equation (6.2) takes care of propagating the result upwards. By the connectivity preservice and path disjointness criteria, it must be the case that for every $u' \in \mathcal{N}(u_{k+1}, \tau_1)$, there exists a unique, vertex-disjoint path $\pi_{u_{k+1}, u'}$ in G . While $f(u')$ might not be in $\mathcal{N}(v_j, \tau_2)$, vertex disjointness implies that each of $\pi_{u_{k+1}, u'}$ passes through a unique, distinct child of v_j . By hypothesis then, the respective entries of T are true, and the suggested matching will satisfy (6.1). \square

Lemma 3. (*Algorithm Runtime*). The algorithm presented above runs in time $O(|V_1| \cdot |V_2|^2 \cdot d^{2.5})$.

Proof. For each rooting $\tau_1 \in V_1$ and $\tau_2 \in V_2$, we fill a table of size $|V_1| \times |V_2|$. Each entry might require solving a matching instance on a bipartite graph of $O(d)$ nodes, which takes time $O(d^{2.5})$ [34]. As a result, the algorithm requires time $O(|V_1| \cdot |V_2| \cdot d^{2.5})$ to check a fixed rooting. Now note that based on the proof of lemma 2, we can fix τ_1 and iterate over V_2 for the choice of τ_2 instead of trying all rootings. The claim follows. \square

6.4.4 2-pass Approach

We now improve the runtime by a factor of $|V_2|$ to obtain the claim of Theorem 1. As is, the algorithm needs to try $O(|V_2|)$ rootings to decide embeddability. Since the root $\tau_1 \in V_1$ is fixed, let T_v be the table computed when D_2 is rooted at $v \in V_2$. We show here that only two passes suffice to compute a table T^* such that $T^*[\tau_1, v] = \text{true}$ iff $T_v[\tau_1, v] = \text{true}, \forall v \in V_2$. It is then not hard to see that $D_1 \subseteq D_2$ iff at least one entry of the τ_1 -th row of T^* is true.

During the first pass, we root D_2 arbitrarily at $\tau_2 \in V_2$ and compute T_{τ_2} as before. The second pass involves top-down message passing. In particular, we iterate through V_2 in pre-order, starting from τ_2 , and each node $p_2 \in V_2$ forwards to every child $v_2 \in \mathcal{N}(p_2, \tau_2)$ a message $\mu(p_2, v_2)$ that is equal to

$$\{v_1 \in V_1 \mid D_1[v_1 \downarrow \tau_1] \subseteq (D_2[p_2 \downarrow p_2] \setminus D_2[v_2 \downarrow p_2])\}.$$

Intuitively, $\mu(p_2, v_2)$ contains all nodes of V_1 that can successfully embed in p_2 without using the subtree hanging from v_2 . Given T_{τ_2} and the respective message from the parent of p_2 , say $p'_2 \in V_2$, we compute $\mu(p_2, v_2)$ by iterating over V_1 in arbitrary order and including $v_1 \in V_1$ if any of the following holds (fig. 29):

1. $T_{\tau_2}[v_1, v'_2] = \text{true}$ for a node $v'_2 \in \mathcal{N}(p_2, \tau_2) \setminus \{v_2\}$, i.e., v_1 embeds in a child of p_2 other than v_2 .
2. $v_1 \in \mu(p'_2, p_2)$, i.e., v_1 embeds in p'_2 even if none of its children are allowed to embed in $D_2[p_2 \downarrow p'_2]$.
3. $\beta(v_1) \preceq \beta(p_2)$ and there exists an assignment $M \subseteq \mathcal{N}(v_1, \tau_1) \times [(\mathcal{N}(p_2, \tau_2) \setminus \{v_2\}) \cup \{p'_2\}]$ that satisfies equation (6.1) with T_{τ_2} as the truth table, and in addition, if $(v'_1, p'_2) \in M$,

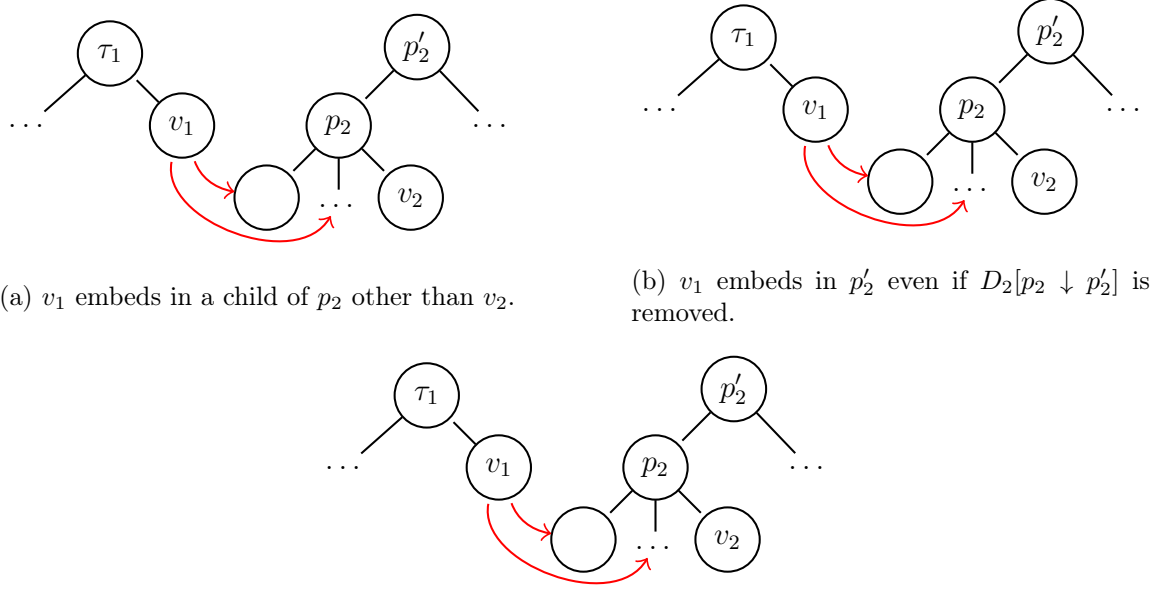


Figure 29: The three cases of message construction.

then $v'_1 \in \mu(p'_2, p_2)$. This means that v_1 is subsumed by p_2 and every child of v_1 embeds either in a child of p_2 other than v_2 , or in the parent of p_2 without reusing any parts of the subtree $D_2[p_2 \downarrow p'_2]$.

Lemma 4. (*Message Correctness*). The process described above computes $\mu(p_2, v_2)$ correctly.

Proof. Let $\mu(p_2, v_2)$ denote the message as computed by the algorithm and define $\mu^*(p_2, v_2)$ to be $\{v_1 \in V_1 \mid D_1[v_1 \downarrow \tau_1] \subseteq (D_2[p_2 \downarrow p_2] \setminus D_2[v_2 \downarrow p_2])\}$.

To show that $\mu(p_2, v_2) \subseteq \mu^*(p_2, v_2)$, suppose that $v_1 \in \mu(p_2, v_2)$ and consider what forced this decision:

1. There exists a child $v'_2 \neq v_2$ such that $T_{\tau_2}[v_1, v'_2] = \text{true}$. Then, by correctness of the original algorithm, $D_1[v_1 \downarrow \tau_1] \subseteq D_2[v'_2 \downarrow \tau_2]$. However, $D_2[v'_2 \downarrow \tau_2]$ is a subset of $D_2[p_2 \downarrow p_2]$, since v'_2 is still in $\mathcal{N}(p_2, p_2)$, and it does not have any nodes in common with $D_2[v_2 \downarrow p_2]$, otherwise we would have a cycle. It then follows that $v_1 \in \mu^*(p_2, v_2)$.
2. If $v_1 \in \mu(p'_2, p_2)$, then by a simple inductive argument we obtain that $v_1 \in \mu^*(p'_2, p_2)$, and since $D_2[p'_2 \downarrow p'_2] \setminus D_2[p_2 \downarrow p'_2]$ is just one branch of $D_2[p_2 \downarrow p_2]$ that does not have anything to do with any other children of p_2 , the claim follows.
3. This condition is straightforward, since v_2 , and therefore the subdesign hanging from it, are completely excluded from the matching process.

Opposite direction is similar and we only sketch it. Assume $v_1 \in \mu^*(p_2, v_2)$ and consider where v_1 embeds:

1. In a child v'_2 of p_2 other than v_2 ; but then $D_2[v'_2 \downarrow p_2] = D_2[v'_2 \downarrow \tau_2]$.
2. In the parent of p_2 , say p'_2 , without involving p_2 . Then, it must be that $v_1 \in \mu^*(p'_2, p_2)$ and inductively we get $\mu^*(p'_2, p_2) = \mu(p'_2, p_2)$.
3. In p_2 and v_2 is not included in the relevant matching.

All our conditions then follow. □

Similarly, once $\mu(p_2, v_2)$ is passed to v_2 , we set $T^*[v_1, v_2]$ to true for each $v_1 \in V_1$ that satisfies any of the following conditions:

1. $T_{\tau_2}[v_1, v_2] = \text{true}$.
2. $v_1 \in \mu(p_2, v_2)$.
3. $\beta(v_1) \preceq \beta(v_2)$ and there exists an assignment $M \subseteq \mathcal{N}(v_1, \tau_1) \times (\mathcal{N}(v_2, \tau_2) \cup \{p_2\})$ that satisfies equation (6.1) with T_{τ_2} as the truth table, and in addition, if $(v'_1, p_2) \in M$, then $v'_1 \in \mu(p_2, v_2)$.

A case analysis identical to that of lemma 4 should persuade us that T^* is built as claimed, and this concludes our algorithm and the proof of theorem 1.

6.5 Kinematic Admissibility

We now extend our notion of embedding to capture kinematic functionality. As mentioned previously, nodes will represent rigid bodies, and edges will represent revolute joints.

6.5.1 Extending Definitions

We extend the definition of a unit block to represent a rigid body. A unit block $B = \langle o_B, \mathcal{B}, \alpha, \phi \rangle$ is a rigid body with origin o_B , reference frame \mathcal{B} , attachment points α , and built-in functionalities ϕ . We refer to o_B simply as B when the meaning is not ambiguous, and adopt the convention that nodes are named with lowercase letters, whereas the rigid bodies they represent are named with the same uppercase letter (*e.g.* node b will represent unit block B , with body frame \mathcal{B}).

The *attachment points* $\alpha = \{\mathcal{B}\mathbf{r}_{\alpha_1/B}, \mathcal{B}\mathbf{r}_{\alpha_2/B}, \dots\}$ are the set of points where B is connected to other unit blocks by revolute joints. As each attachment point is attached to a single revolute joint, we sometimes refer to attachment points by the corresponding joint, *e.g.* if joints J and K attach to B at α_1 and α_2 , we write $\alpha = \{\mathcal{B}\mathbf{r}_{J/B}, \mathcal{B}\mathbf{r}_{K/B}, \dots\}$.

Next, we associate with each edge a revolute joint through $\gamma : E \mapsto \mathbb{J}$, similar to β for nodes. A *revolute joint* $J = \langle \mathcal{A}\mathbf{r}_{J/A}, \mathcal{B}\mathbf{r}_{J/B}, \mathcal{A}\hat{\mathbf{u}}, \mathcal{B}\hat{\mathbf{u}}, \theta, \mathcal{A}R_0^{\mathcal{B}} \rangle$ connects a pair of unit blocks (A and B) and permits rotation about an axis. J has attachment points $\mathcal{A}\mathbf{r}_{J/A}$ and $\mathcal{B}\mathbf{r}_{J/B}$, rotation axis specified by unit vectors $\mathcal{A}\hat{\mathbf{u}}$ and $\mathcal{B}\hat{\mathbf{u}}$, joint angle θ , and reference orientation $\mathcal{A}R_0^{\mathcal{B}}$. The *joint-angle* θ of J specifies the amount of rotation about $\mathcal{A}\hat{\mathbf{u}}$, relative to the *reference orientation* $\mathcal{A}R_0^{\mathcal{B}} = \mathcal{A}R^{\mathcal{B}}(\theta = 0)$. Given a fixed reference orientation, we can find $\mathcal{A}R^{\mathcal{B}}$ as a function of θ : $\mathcal{A}R^{\mathcal{B}}(\theta) = \exp(\theta[\mathcal{A}\hat{\mathbf{u}}]_{\times})\mathcal{A}R_0^{\mathcal{B}}$. Here, $[\mathcal{A}\hat{\mathbf{u}}]_{\times}$ is the cross-product matrix of $\mathcal{A}\hat{\mathbf{u}}$.

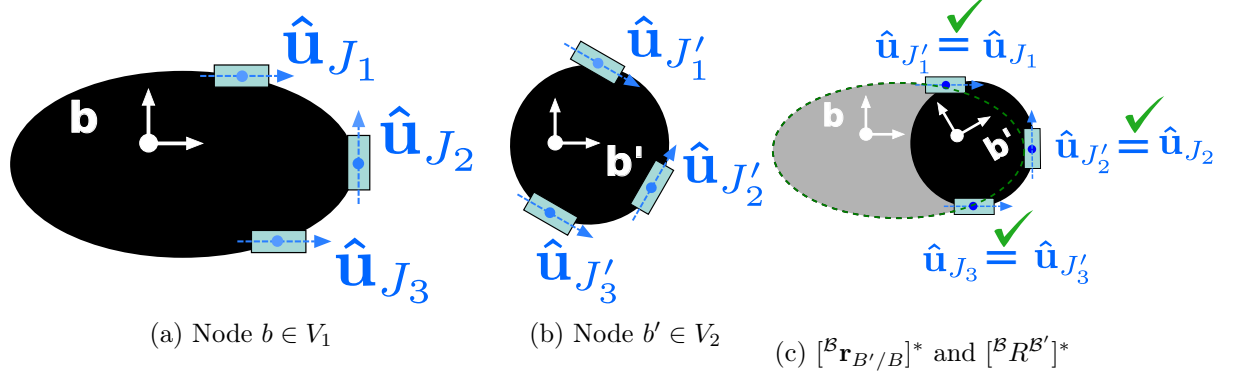


Figure 30: Def. 4, 1 (nodes). In fig. (c), a position and orientation have been found in which each child joint of b is aligned with a corresponding child joint of b' .

6.5.2 Kinematic Admissibility

When we say the embedding $D_1 \sqsubseteq_f D_2$ is *kinematically admissible*, we mean that D_1 exactly replicates every kinematic DoF present in D_2 . Intuitively, to verify kinematic admissibility, we must find a configuration (set of joint angles) for D_2 such that for each joint in D_1 , the corresponding joint in D_2 exactly matches its position and axis. If we lock the position of all joints $\gamma(E_2 \setminus f^e(E_1))$ (those which do not correspond to joints in D_1) while in this configuration, we get a kinematic structure identical to D_1 . We present a local form of this global requirement by augmenting two of the conditions of Definition 3:

Definition 4 (Kinematic Admissibility). Given f such that $D_1 \sqsubseteq_f D_2$, f is kinematically admissible if it satisfies the following two conditions.

- (1) (Nodes) Let $b' \in V_2$ embed $b \in V_1$. There must exist a position $[\mathcal{B}\mathbf{r}_{B'/B}]^*$ and orientation $[\mathcal{B}R^{\mathcal{B}'}]^*$ such that for every $J_i \in \gamma(\mathcal{N}^e(b))$ there is a unique $J'_i \in \gamma(\mathcal{N}^e(b'))$ such that:

$$\mathcal{B}\mathbf{r}_{J_i/B} = [\mathcal{B}R^{\mathcal{B}'}]^* \mathcal{B}'\mathbf{r}_{J'_i/B'} + [\mathcal{B}\mathbf{r}_{B'/B}]^* \quad (6.3)$$

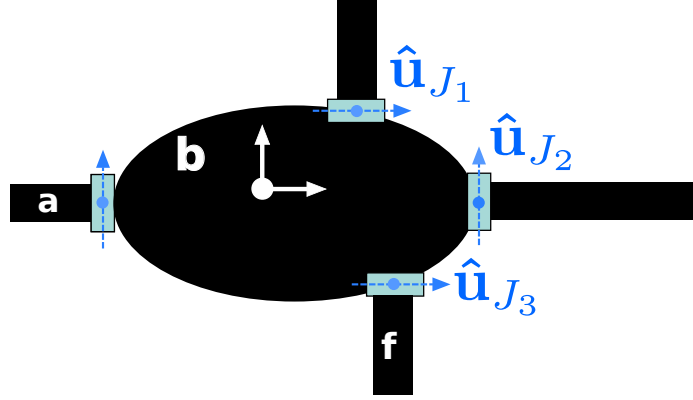
$$\mathcal{B}\hat{\mathbf{u}}_{J_i} = [\mathcal{B}R^{\mathcal{B}'}]^* \mathcal{B}'\hat{\mathbf{u}}_{J'_i} \quad (6.4)$$

- (2) (Paths) Let path $\pi_{ab} = a' \rightsquigarrow b'$ in G_2 embed edge $(a, b) \in E_1$. Let $(a', c') \in \mathcal{N}^e(a')$ be the first edge of π_{ab} . Let $K' = \gamma((a', c'))$ and $K = \gamma((a, b))$ be aligned; that is, let $\mathcal{B}\mathbf{r}_{K'/B} = \mathcal{B}\mathbf{r}_{K/B}$ and $\mathcal{B}\hat{\mathbf{u}}_{K'} = \mathcal{B}\hat{\mathbf{u}}_K$. Let $\Theta(\pi_{ab})$ be the set of angles of all joints on π_{ab} . There must exist joint angles $[\Theta(\pi_{ab})]^*$ such that:

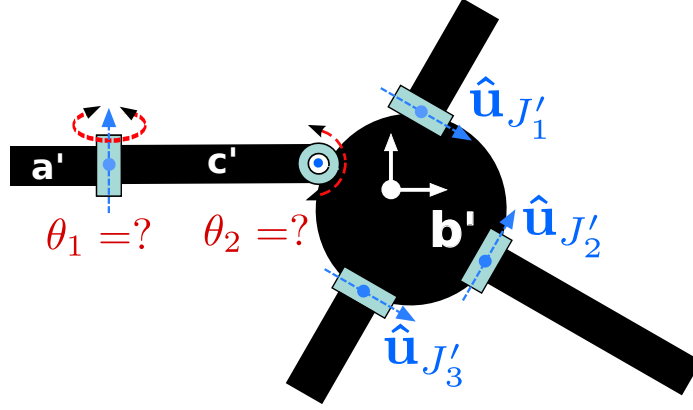
$$\mathcal{B}\mathbf{r}_{B'/B}([\Theta(\pi_{ab})]^*) = [\mathcal{B}\mathbf{r}_{B'/B}]^* \quad (6.5)$$

$$\mathcal{B}R^{\mathcal{B}'}([\Theta(\pi_{ab})]^*) = [\mathcal{B}R^{\mathcal{B}'}]^* \quad (6.6)$$

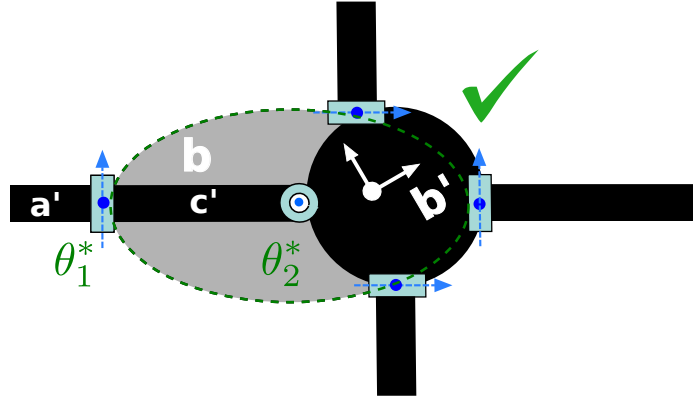
See Fig. 30 and 31 for an illustration. The condition on nodes ensures that whenever some b' embeds b , there is a special position $[\mathcal{B}\mathbf{r}_{B'/B}]^*$ and orientation $[\mathcal{B}R^{\mathcal{B}'}]^*$ for b' in which some of its child edges match with all child edges of b . The condition on paths ensures



(a) Nodes a and b in V_1



(b) Nodes a' , c' , and b' in V_2



(c) Configuration $\Theta^*(\pi_{ab})$ under which condition 2 is satisfied.

Figure 31: Def. 4, 2 (paths). For path $\pi_{ab} = (a', c', b')$ to embed edge (a, b) , there must be angles $[\Theta(\pi_{ab})]^*$ for which ${}^B\mathbf{r}_{B'/B} = [{}^B\mathbf{r}_{B'/B}]^*$ and ${}^B R^{B'} = [{}^B R^{B'}]^*$

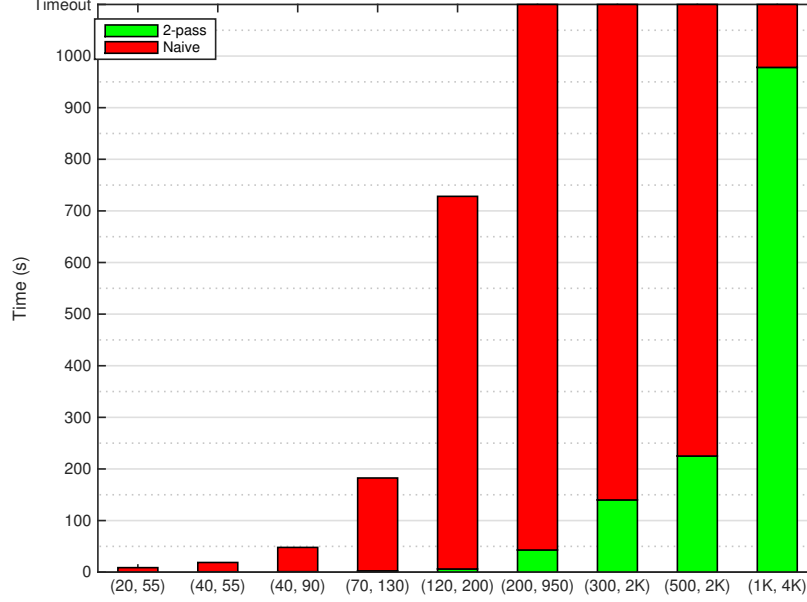


Figure 32: 2-pass against naive embedding on random trees. x -axis is benchmark size as a function of the nodes in the subdesign and the superdesign. Timeout is 2 hours.

that there is a configuration for the path connecting $f(a)$ to $f(b)$ which actually allows b' to assume this special position and orientation.

6.5.3 Checking Kinematic Admissibility

To check (1) for a modular robot system with known links and joints, we pre-compute solutions $[\mathcal{B}\mathbf{r}_{B'/B}]^*$ and $[\mathcal{B}R^{B'}]^*$ for each pair of nodes in the system by brute-force. When running the algorithm, they can be quickly found by table-lookup. (2) is checked through inverse kinematics (IK); when an edge maps to a path, we impose the additional restriction that an IK solution for this path must be found which allows its terminating node to reach $[\mathcal{B}\mathbf{r}_{B'/B}]^*$ and $[\mathcal{B}'R^{\mathcal{B}}]^*$. IK takes time exponential in the number of joints on the path π_{ab} , so in practice it is the costliest operation in our algorithm.

6.6 Experiments

The algorithms were implemented in Python on top of graph-tool [73]. All experiments were run on a single core of an Intel i7 at 2.4GHz; reported times are the average over ten repetitions.

The first experiment is a topology benchmark: designs are random trees of max degree 5, and each node is assigned one of two functionalities uniformly at random. Since the naive approach is much simpler to implement, is it worthy to adopt 2-pass from a practical viewpoint? Fig. 32 shows that naive quickly becomes infeasible, with 2-pass outperforming it and scaling really well with input size. Even large instances, where designs contain thousands of nodes, are solved in almost 15 minutes.

In another experiment, not plotted due to space limits, subdesigns are complete binary trees where each node is assigned one of two functionalities uniformly at random. Superdesigns are complete binary trees of twice the depth, with nodes at even depth assigned one



Figure 33: Grasper (left) and a walker (right) designs for the SMORES robot [17].

functionality and nodes at odd depth assigned the other one. There, 2-pass mapped a complex 127 node subdesign to a 16K node superdesign in less than 27 minutes.

Profiling the algorithm, almost 67% of the time is spent in the first pass, and 54% is consumed in generating and solving small matchings. Kinematics are definitely the bottleneck; almost ten minutes are required for designs of 200 and 500 nodes, as compared to less than 30 seconds for topology. For smaller designs with 25 and 80 nodes, kinematic embedding is detected in < 5 seconds.

6.7 Applications

We turn to practical applications of embedding, and discuss how we can perform automatic control translation from the subdesign to the superdesign once a kinematically admissible embedding has been computed.

As a first example, consider the grasper and walker robots pictured in figure 33. Both designs are built out of SMORES modules [17]. Each SMORES module has four DoF: three continuously rotating faces called *turntables* and one central hinge with a 180° range of motion. When two SMORES modules connect, the connected faces become rigidly attached; rather than representing such a connection with an edge, we fuse the faces into a single node which is then considered a member of both modules. Figure 34 shows the underlying designs and the embedding found by our algorithm in under one second.

We are now able to map behaviors from the grasper to the walker. Kinematic behaviors for a modular robot design can be specified by gait tables containing a time-series of joint-angles [105]. Given a gait table for the grasper that produces a desired behavior (like wrapping the arms around an object to immobilize it), we can use the mapping from our algorithm to translate the gait table and achieve the same desired behavior with the walker.

More importantly, the same idea can be extended accross different modular robot systems. As an illustration, our algorithm detects that a SuperBot design [81] embeds in a SMORES design 35, and in general, it can be shown that a SuperBot module always embeds in a design of two SMORES modules. Detecting such embeddings automatically and using them to translate behaviors between platforms could save time for researchers who would otherwise try to re-create behaviors manually, especially when working on complex designs where embeddability is not as straightforward.

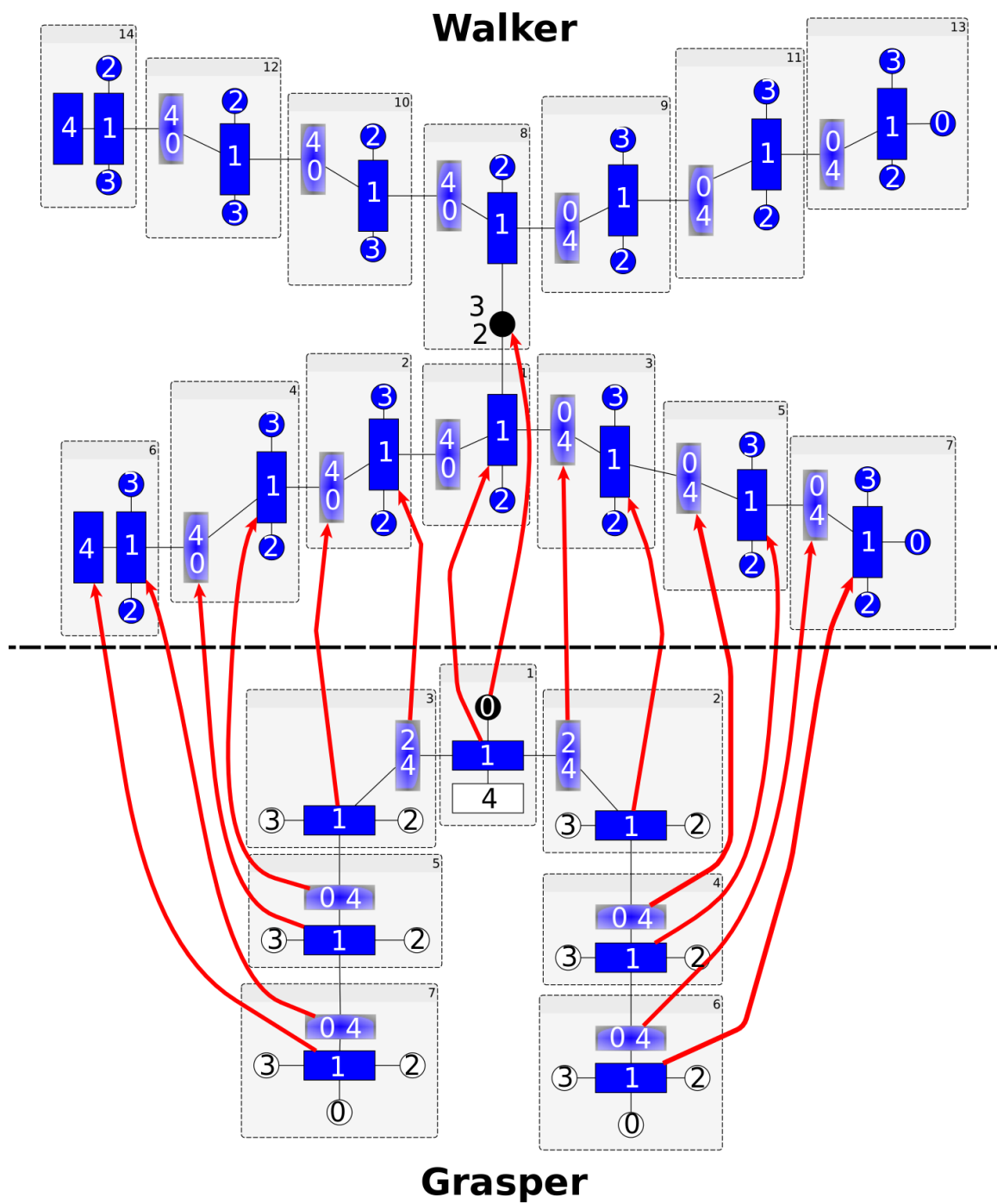


Figure 34: Walker design on top and grasper design on bottom. Red arrows show the discovered embedding.

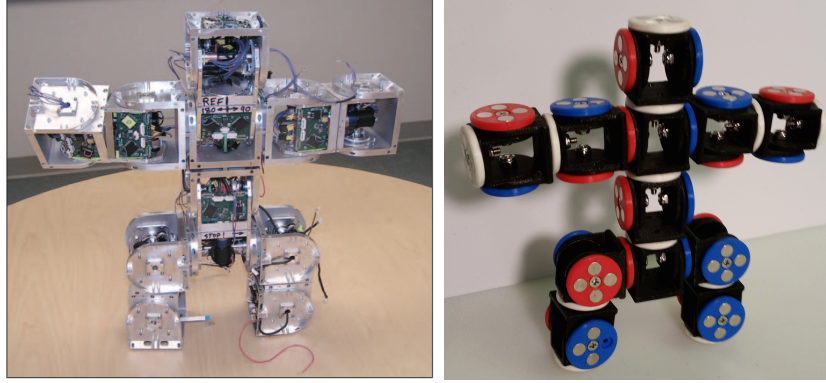


Figure 35: SuperBot subdesign [81] embeds in SMORES superdesign.

6.8 Conclusion and Future Work

We developed and implemented a poly-time algorithm to decide if a given modular robot design can be embedded into another design. The algorithm processes real-life designs in a matter of seconds and scales well with input size. We also formalized the notion of embedding, based on graph representations of modular robots, and highlighted automatic control translation as an application.

In the near future, we will look into handling designs with a small number of cycles and decreasing the runtime of kinematic checking. In the longer term, we will move from detecting embeddability to design synthesis. We believe that our embedding approach is a useful starting point for this line of research, and have obtained promising preliminary results.

Chapter 7

Accomplishing High-Level Tasks with Modular Robots

The advantage of modular self-reconfigurable robot systems is their flexibility, but this advantage can only be realized if appropriate configurations (shapes) and behaviors (controlling programs) can be selected for a given task. In this chapter, we present an integrated system for addressing high-level tasks with modular robots, and demonstrate that it is capable of accomplishing challenging, multi-part tasks in hardware experiments. The system consists of four tightly integrated components: (1) A high-level mission planner, (2) A large design library spanning a wide set of functionality, (3) A design and simulation tool for populating the library with new configurations and behaviors, and (4) modular robot hardware. This chapter excerpts heavily from the author’s work in [39] and [40]. Credit is due to collaborators Gangyuan Jing and Prof. Hadas Kress-Gazit at Cornell University, who contributed significantly to the work presented here.

7.1 An End-to-End System for Accomplishing Tasks with Modular Robots

The strength of MSRR systems lies in their flexibility. In principal, self-reconfiguration will allow modular robots to transform into designs specifically tailored to the needs of each new task they encounter, allowing them to elegantly address a wide variety of tasks by reconfiguring into a wide variety of simple solutions. However, this strategy poses an obvious challenge: given a task, is it possible to select an appropriate configuration (robot shape) and behavior (controlling program) to address it? This has been a longstanding problem in the field, and remains a significant barrier to the use of modular robots to solve real-world problems [107].

We present an end-to-end system capable of selecting appropriate modular robot configurations and behaviors to solve complex high-level tasks. Our system is library-driven: rather than attempting to generate new designs from scratch, users specify task requirements and a high-level controller retrieves designs satisfying the requirements from a library of existing designs. In addition to library management, the system integrates tools for low-level design creation, high-level mission planning, and physical modular robot hardware.

We leverage ideas from recent work on automatic controller synthesis with correctness guarantees from high-level task specification [6, 7, 42, 46, 75, 103]. These methods have



Figure 36: Six configurations from the design library

proven effective for addressing high-level tasks with traditional robots, allowing users to specify task requirements at a high level using formal languages and then automatically synthesizing low-level robot controllers with performance guarantees. Applying these methods in the context of modular robotics introduces an additional layer of complexity due to the fact that the morphology of the robot is not fixed.

Through hardware experiments, we demonstrate that our system is capable of addressing challenging multi-part tasks. This chapter presents the details of the system, discusses its strengths and weaknesses, and discusses challenges and opportunities related to applying a similar system in a real-world setting.

7.1.1 System Overview

Here, we provide a brief overview of the entire system. Figure 37 provides a visual companion to this section.

The system is built around a design library that spans a wide range of useful functionality. Library entries are configurations and behaviors for the SMORES-EP modular robot, which are designed in a physics-based simulator and design tool called VSPARC which we created for this purpose. Users build, program, and test modular robot designs through a graphical user interface, and can save their designs to a web server, allowing them to be shared with others. Any configuration or behavior created in the simulator can be directly ported to the hardware modular robot system, SMORES-EP.

Our system allows users to solve high-level tasks with modular robots. Tasks are specified in a mission planning tool using Structured English [25], a high-level language. Users do not specify which configurations and behaviors should be used to complete the task, but rather describe the required functionality. For example, the user might request that the robot perform a **drive** action in a **tunnel** environment labeled with the property $\text{max_height} = 3$.

To develop a solution to the task, the high-level mission planner fulfills each of the specified functionality by automatically selecting robot configurations and behaviors from

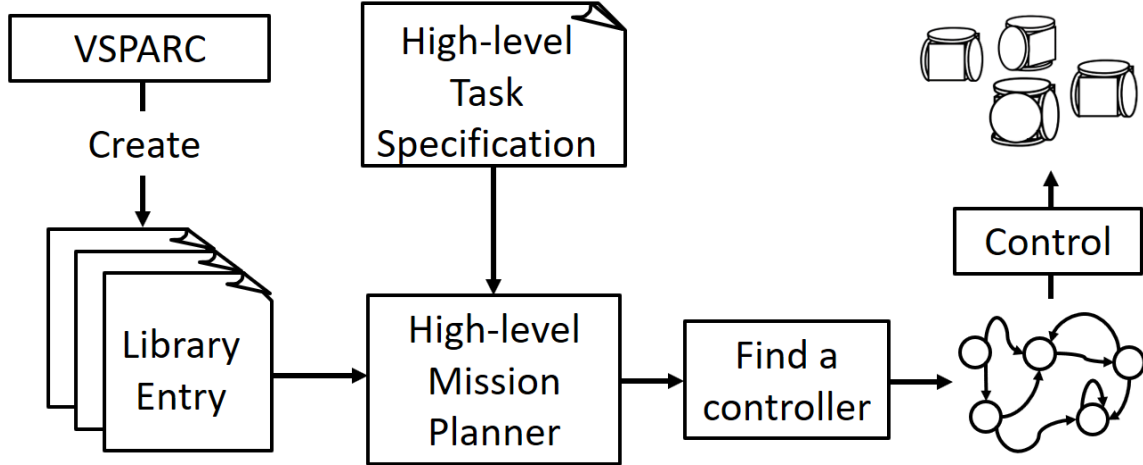


Figure 37: System flowchart

the design library, generating a controller in the form of a finite-state automaton. In the above example, the system could select any configuration that is capable of executing a **drive** behavior while maintaining a maximum height of 3 modules or less. In a sense, the high-level planner treats the entire modular robot system as a single robot with a set of capabilities defined by the library. The mission planner can then execute the controller to complete the task, directly commanding hardware SMORES-EP robots based on environment information from sensors.

7.1.2 Contributions

We present an integrated system capable of addressing high-level tasks with modular robots. The tasks it addresses are *reactive*: they require decision-making about what action to perform based on the sensed environment; *complex*: they include multiple sub-tasks with potentially very different requirements; and *high-level*: the task specification encodes the desired outcomes, and the system intelligently synthesizes a solution that results in those outcomes using the available configurations and behaviors from the library.

This system is one of the first to address these kinds of tasks with modular self-reconfigurable robots, which introduce an additional layer of complexity because they can assume many configurations. This represents a significant contribution to the field, because such systems will be necessary for modular robots to operate in realistic task scenarios. By providing this framework and demonstrating its success in the lab, we hope to lay the foundation for future modular robot systems to address tasks in the real world.

The system includes four tightly integrated components: (1) A high-level mission planner, (2) A large design library spanning a wide set of functionality, (3) A design and simulation tool for populating the library with new configurations and behaviors, and (4) modular robot hardware. Several of the subcomponents represent research contributions. Our novel design tool (VSPARC) represents a novel contribution, as does our library of 52 configurations and 97 behaviors. We also introduce a minor theoretical contribution by checking the feasibility of robot behaviors prior to controller synthesis.

7.2 Related Work

Much of the existing research in MSRR systems has focused on establishing the the fundamental capabilities that differentiate these systems from traditional robots. Notably, MSRR systems have demonstrated the ability to form a wide variety of physical morphologies capable of diverse modes of locomotion, suitable to a range of different terrains [105]. The ability to autonomously reconfigure has been demonstrated [108], and a number of reconfiguration planning algorithms have been developed [88].

Similarly, a great deal of work has been done to develop behaviors for MSRR. Many efforts focus on distributed control strategies, taking advantage of the distributed nature of MSRR hardware [99]. Distributed strategies include central pattern generators [85] and hormone-based control [80]. Genetic algorithms have been used to automatically generate both modular robot designs and behaviors [36].

It is clear that MSRR systems have demonstrated the ability to accomplish low-level tasks such as reconfiguration, locomotion, and manipulation. However, to truly live up to their promise of flexibility in real-world applications, systems must be developed that leverage these low-level capabilities to address complex, high-level, multi-part tasks.

While there is a robust body of research into addressing high-level tasks with traditional robots, little work has been done in this area with modular robots. High-level control of modular robots poses a unique challenge, because solving tasks involves selecting not only appropriate behaviors, but also appropriate configurations. This makes it all the more important to develop automated systems that can synthesize task-appropriate modular robot configurations and behaviors from high-level specifications.

In [10], Castro *et al.* introduce a high-level control framework for the CKBot modular robot. This framework lays the theoretical foundations for our high-level mission planner, one of the four major components of our system. We expand the framework into a larger system capable of addressing significantly more sophisticated tasks. In addition to the mission planner, we provide design and simulation tools for creating and testing modular robot configurations and behaviors, and a large library (52 designs, 97 behaviors, 19 properties) with designs capable of addressing a wide range of tasks. We expand the theoretical formalism introduced by Castro to include both behavior and environment properties, increasing the expressiveness of task specification, and introduce a performance improvement by grounding abstract action specifications in concrete configurations and behaviors prior to automata synthesis.

Tosun *et al.* [95] introduce a system that allows users to rapidly synthesize modular robot designs and behaviors by composition. The system includes a physics-based simulator and a hierarchically organized library of configurations and associated behaviors. The goal of this work is to aid in the selection of modular robot configurations and behaviors appropriate to complex tasks, but it takes a very different approach than our automated system, instead providing tools for users to manually create new designs by combining library entries using series and parallel composition operations.

Outside the realm of modular robotics, systems have been developed that can synthesize rapidly manufacturable robot designs from high-level user specifications [55],[54], [82]. This work is similar to ours in the sense that high-level specifications from the user are interpreted to synthesize robot designs and behaviors from elements in a design library. The goal of these systems is to allow novice users to rapidly design and build functioning robots at

low cost, using fabrication techniques such as 3D printing [55],[54] and origami folding [82]. Consequently, the scope of the tasks they address is very different from ours. In these systems, library entries are electromechanical components such as motors, motor drivers, and microcontrollers, and a high level task might be “Create a robot that can walk and turn.” In contrast, library entries in our system are whole robots with associated behaviors, and we address complex, multi-part tasks such as “Climb on top of the table, and move any debris you find into the trash bin.”

7.3 Background

In this section, we formalize modular robot systems and provide background on controller synthesis techniques.

7.3.1 Modular Robot Systems

Definition 5 (Module). A module is the fundamental unit of a modular robot system. Each module is a small robot that can receive and respond to commands, move, and connect to other modules. In this work, we consider only homogeneous modular robot systems, meaning that all modules in the system are identical.

We define a module as $m = (J, A)$. $J = \{J_1, \dots, J_d\}$ is the set of joints of the module with d degrees of freedom. $A = \{A_1, \dots, A_k\}$ is the set of *attachment points* where the module can connect to other similar modules. Each attachment point can only connect to one other module at a time. We denote the attachment point A_i of module m as $m.A_i$.

Definition 6 (Configuration). A configuration is a connected set of modules that acts together as a single robot. The smallest configuration is a single module. A configuration is denoted as $\mathcal{C} = (M, E)$, where $M = \{m_1, \dots, m_q\}$ is the set of connected modules that form the configuration and E is the set of connections between modules, represented as pairs of attachment points. $(m_i.A_{a_1}, m_j.A_{a_2}) \in E$, where $m_i, m_j \in M$, and $m_i \neq m_j$. cannot have two disconnected sets of modules that move independently).

Definition 7 (Joint Command). Joint commands are used to control the joints of the modules. A command to a joint J_i is defined as $u_{J_i} = (\alpha, V, t)$, where $\alpha \in \{\text{Position, Velocity}\}$ is the type of command, $V \in \mathbb{R}$ is the value of the command, and $t \in \mathbb{R}$ is the time duration of the command. For example, $u_{J_i} = (\text{Position}, \frac{\pi}{2}, 2)$ commands joint J_i to hold the angle $\theta = \frac{\pi}{2}$ rad for 2 seconds. Similarly, $u_{J_i} = (\text{Velocity}, \pi, 3)$ will drive joint J_i with angular velocity of $\dot{\theta} = \pi \frac{\text{rad}}{\text{sec}}$ for 3 seconds. We assume there are low-level controllers (*e.g.* PID controllers) that can drive the corresponding joint to satisfy the command u_{J_i} .

Definition 8 (Behavior). For a configuration \mathcal{C} , we define a behavior $B_{\mathcal{C}} = \{b_1, \dots, b_n\}$ as a sequence of behavior states. Each behavior state is defined as $b_i = (U, T)$, where U is the set of joint commands for all joints of all modules in the configuration. The time duration T of each behavior state is equal to longest duration of its joint commands U , ensuring that behavior execution will move on to the next state only once all joint commands in the current behavior state have completed.

7.3.2 Controller Synthesis

In this work, we utilize existing work on controller synthesis [25, 46] to generate high-level controllers for modular robot systems. The process of controller synthesis consists of three

main steps: (1) representing the robot and the environment using a discrete abstraction, (2) expressing desired robot tasks with a formal specification language, (3) searching for a control strategy that satisfies the given task specification, or determining that such a strategy does not exist.

Robot and Environment Abstraction: To represent the continuous environment state and robot actions as discrete models, we abstract the environment events and robot capabilities into sets of boolean variables. The value of each variable represents the sensed environment state or the current robot actions. For example, the environment variable **Cup** is **True** if and only if the robot is currently sensing a cup with its camera. Similarly, the robot variable **Push** is **True** if and only if the robot currently performing a pushing action.

Robot Task Specification: A wide range of robot tasks can be defined using a formal language called Linear Temporal Logic (LTL). In [25], authors introduce a tool called LTLMoP that allows users who are unfamiliar with LTL to specify robot tasks in a formal language called Structured English, which is closer to natural language. LTLMoP then automatically translates Structured English specifications into LTL formulas. The following is an example of a robot task specification written in the Structured English:

- visit **Classroom**
- if the robot senses **Student** then do **Greet**
- do **Pickup** if and only if the robot senses **Trash**

In these examples, **Classroom**, **Greet**, **Pickup** are robot action variables and **Student**, **Trash** are environment variables.

Controller Synthesis and Execution: Authors of [46] introduce a framework to automatically generate a high-level robot controller to satisfy a task specification, or decide such controller does not exist. The synthesized controller is a finite-state automaton, and specifies robot actions that satisfy the task. Each state in the controller is labeled with robot variables, and each transition is labeled with environment variables.

To accomplish a tasks, the synthesized controller is implemented continuously by mapping each robot variable to a low-level robot controller, and mapping each environment variable to a robot sensing function. With these mappings, the robot is able to detect the environment and perform desired actions to satisfy the task specification.

7.4 System

7.4.1 Modular Robot Hardware - SMORES-EP Robot

Our system is built around the SMORES-EP modular robot, but could be extended to other modular robot hardware systems. Because individual modules have no way of sensing their environment, localization is provided by AprilTag markers [68] mounted to modules and objects of interest, tracked by an overhead camera. The AprilTag tracker, high-level planner, and module control software run with a control loop time of about $4Hz$ on a laptop a $2.4GHz$ processor and $4GB$ of RAM.

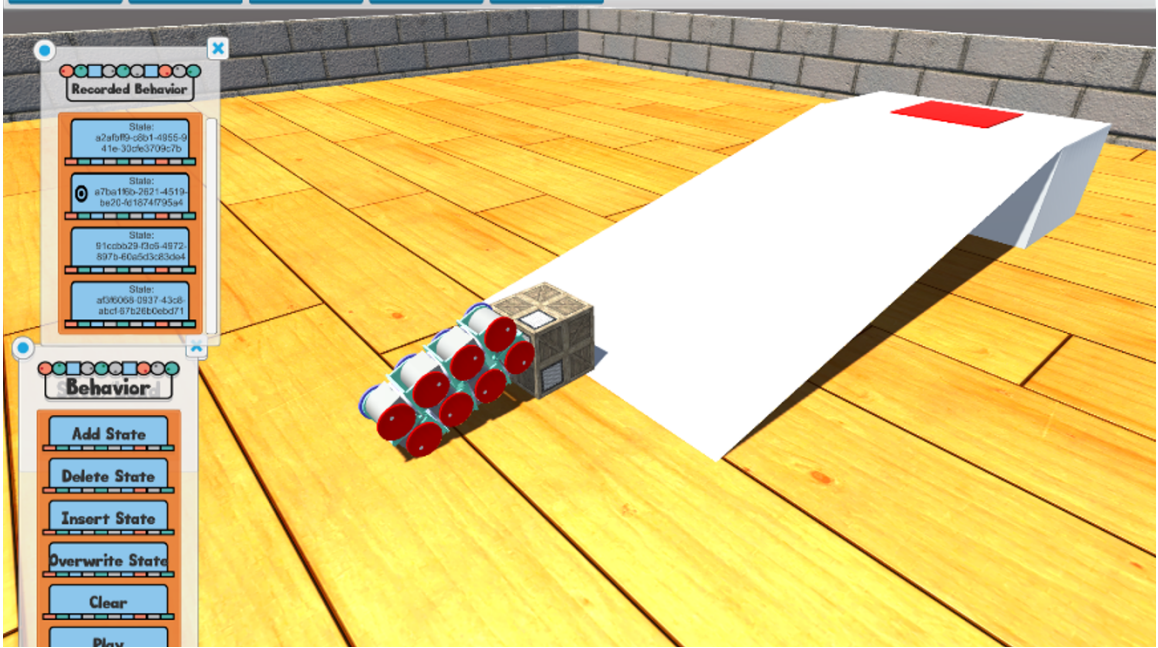


Figure 38: VSPARC user interface

7.4.2 Design and Simulation Tool: VSPARC

VSPARC, which stands for **V**erification, **S**imulation, **P**rogramming And **R**obot **C**onstruction, is our interactive design tool that allows users to design configurations and behaviors for SMORES-EP robots, and simulate them with a real-time physics engine. As shown in Figure 38, the graphical user interface, powered by the Unity3D Engine [1], allows users with little background in robotics to design and test different robot configurations and behaviors. The ability to control each joint of each module grants more experienced users the possibility to create complex designs.

VSPARC provides physical modeling of SMORES-EP, taking into consideration factors such as the connector and actuator force limits. This allows users to test and verify behaviors before running them with physical modules and receive early warning if, for example, their behavior would likely cause the connection between two modules to break.

VSPARC is available for free online at www.vsparc.org, and enables users to save and share their designs to a central server, allowing a large number of users to contribute to our design library. VSPARC's main features are listed below:

- Design configurations with unlimited number of modules and visualize the design in a 3D environment.
- Command positions or velocities for each joint of all modules.
- Design behaviors for any configuration by creating a sequence of joint commands.
- Simulate the performance of any behavior in a physics engine.

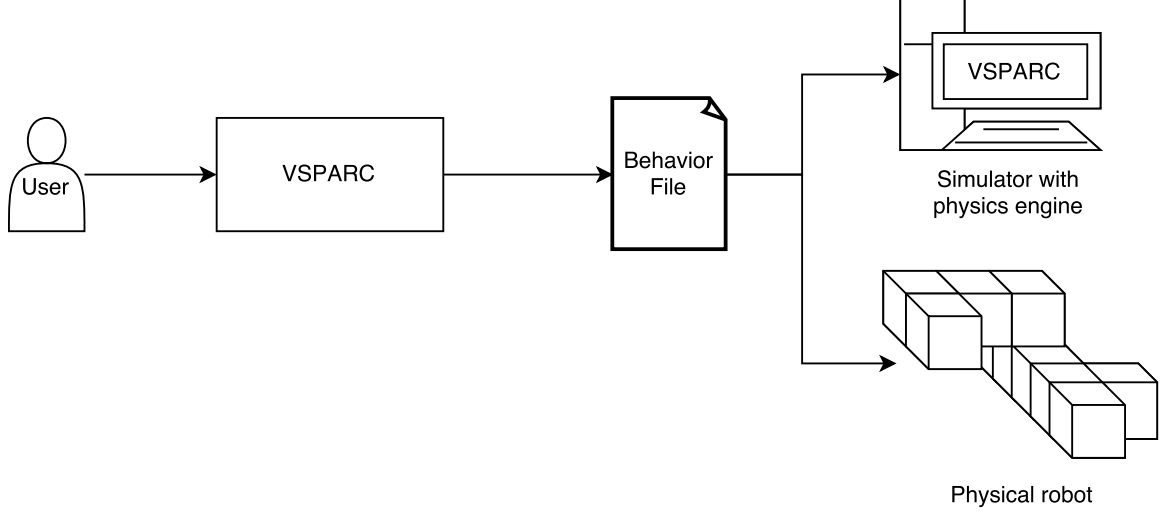


Figure 39: The same behavior file can be used by both the simulator and the physical robot.

- Create and share designs online. Test and improve other users' designs.

As shown in Figure 39, behaviors designed in VSPARC can be exported as XML files and then run on SMORES-EP modules, providing seamless translation of behaviors from the simulator to physical robots.

7.4.3 Design Library

In this section, we introduce a library-driven framework to organize configurations and behaviors created in VSPARC. We introduce the notion of *properties*, which specify the functionality and constraints of behaviors, and the *robot design library*, which can be searched to find configurations and behaviors with desired properties.

Definition 9 (Property). Properties provide high-level descriptions of the intended effects of a behavior, as well as the environment in which the behavior is appropriate. We define a property as $p = (p_n, \Omega)$, where p_n is the name of the property (i.e. a description title, in English) and Ω is the set of values of the property. For example, a behavior with the property $p = (\text{Action}, \{\text{Move}, \text{Push}\})$ can perform both *Move* and *Push* actions. Properties are also used to describe the environmental conditions required for the behavior to run as expected. For example, the property $p = (\text{ObjectWeight}, [2, 5])$ indicates that the behavior can appropriately interact with an object if its weight is between 2 and 5 module-weights. In this case, the property is a quantitative description of the environment. We say a property $p_1 = (p_{n_1}, \Omega_1)$ *satisfies* a property $p_2 = (p_{n_2}, \Omega_2)$ if and only if $p_{n_1} = p_{n_2}$ and $\Omega_1 \subseteq \Omega_2$.

Properties connect tasks with behaviors that are appropriate to address them. In Section 7.4.4, we discuss how correct behaviors for a task can be automatically selected based on requirements over property values. Table 4 lists some examples of environment and behavior properties that might be used for common robot tasks.

Table 4: Examples of property names

Properties for Robot Behavior	Properties for Environment
Speed	Box_Mass
Width	Stair_Height
Height	Ground_Roughness
Action	Tunnel_Height

Definition 10 (Robot Design Library). The design library is a collection of modular robot configurations and behaviors labeled with environment and robot behavior properties. The library \mathcal{L} consists of a set of library entries, $\mathcal{L} = \{l_1, l_2, \dots\}$. Each library entry is defined as $l = (C, B_C, P_e, P_r)$, where C is the configuration and B_C is a behavior associated with C . P_e and P_r are sets of properties that describe the environment conditions and robot behavior functionality, respectively.

As an example, the library entry:

$$l = (C = \text{snake}, B_C = \text{climb}, P_e, P_r)$$

where : $P_e = \{(\text{Ledge_Height}, [2, 3])\}$
and : $P_r = \{(\text{Action}, [\text{Climb}]), (\text{Speed}, [1])\}$

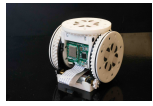



represents a **snake** shape configuration with a **climb** behavior that can climb a ledge with a height of two to three module-lengths, with the speed of 1 module-length per second. Moreover, we say a library entry l *satisfies* a property p if there exist a property $p' \in P_e \cup P_r$ such that p' satisfies p .

To populate the library with different configurations and behaviors designs, we made our design tool available online at www.vsparc.org and distributed the tool to undergraduate and graduate student volunteers, hosting three hackathons in which participants created designs for various robot tasks. Currently, the library includes 52 configurations and 97 behaviors contributed by 20 volunteers. Since the full library is too large to list here, we provide a representative sampling of configurations, behaviors, and properties in Table 5. The unit for length is the side length of a single SMORES-EP module. The unit for mass is the mass of a single SMORES-EP module.

Environmentally Adaptive Parametric Behaviors

As explained in Section 7.3, standard behaviors are defined as a series of joint angles or joint velocities for the modular robot cluster. These are discrete, open-loop actions can be sequenced by the high-level mission planner to complete tasks. Here, we present *environmentally-adaptive parametric behaviors* (EAP behaviors) which provide additional functionality, allowing low-level behaviors to directly respond to sensed conditions in sophisticated ways. These behaviors are *parametric* because they take input arguments, called parameters, which allow them to produce a continuous range of motions. They are *environmentally adaptive* because their parameters are intelligently assigned as a function of the state of the robot and environment.

Definition 11 (Environmentally Adaptive Parametric Behavior). We define an Environmentally-Adaptive Parametric Behavior as $B_C^{EAP} = (\{b_1, b_2, \dots, b_n\}, \mathbf{p}, f)$, where $\{b_1, b_2, \dots, b_n\}$ is

Configuration Name	Single module	Rolling Loop	DoubleDriver	Stair Climber
Number of modules	1	8	7	4
				
Locomotion				
Max robot height	1	2.5	1.5	2
Max robot width	1	1	3	1
Max robot length	1	5	3	3.5
Terrain - Smooth	✓	✓	✓	✓
Terrain - Rough		✓		✓
Terrain - Sloped		✓	✓	✓
Driving - Straight	✓	✓	✓	✓
Driving - Differential drive	✓		✓	
Driving - Holonomic				
Ledge ascent - Max height		0.25		0.75
Ledge ascent - # modules lifted		all		all
Ledge descent - Max height		1		1.5
Ledge descent - # modules lowered		all		all
Manipulation				
Attachment - Push	✓	✓	✓	✓
Attachment - Magnetic	✓		✓	✓
Attachment - Carry				
Workspace size	$X : [-\text{inf}, \text{inf}]$ $Y : [-\text{inf}, \text{inf}]$ $Z : [0, 1]$	$X : [-\text{inf}, \text{inf}]$ $Y : [0, 1]$ $Z : [0, 2.5]$	$X : [-\text{inf}, \text{inf}]$ $Y : [-\text{inf}, \text{inf}]$ $Z : [0, 1.5]$	$X : [-\text{inf}, \text{inf}]$ $Y : [-\text{inf}, \text{inf}]$ $Z : [0, 2]$
Payload mass	1	2	4	2




Configuration Name	Swerve Lifter	backhoe	snake7
Number of modules	9	9	7
			
Locomotion			
Max robot height	2	4	4
Max robot width	4	4	1
Max robot length	3	7	7
Terrain - Smooth	✓		✓
Terrain - Rough			
Terrain - Sloped			✓
Driving - Straight	✓		✓
Driving - Differential drive	✓		
Driving - Holonomic			
Ledge ascent - Max height			3
Ledge ascent - # modules lifted			4
Ledge descent - Max height			3
Ledge descent - # modules lowered			all
Manipulation			
Attachment - Push	✓	✓	✓
Attachment - Magnetic		✓	✓
Attachment - Carry	✓		
Workspace size	$X : [-\text{inf}, \text{inf}]$ $Y : [0, 1]$ $Z : [0, 2]$	$X : [-\text{inf}, \text{inf}]$ $Y : [-3, 3]$ $Z : [0, 4]$	$X : [-3, 3]$ $Y : [-3, 3]$ $Z : [0, 4]$
Payload mass	3	1	1

Table 5: Matrix of designs and properties. Length and mass units are module-lengths and module-masses.

a sequence of behavior states, $\mathbf{p} \in \mathbb{R}^m$ is a vector of parameters, and $f : \mathbb{R}^n \rightarrow \mathbf{p}$ is the controller function.

Like standard behaviors, EAP behaviors consist of a sequence of behavior states $\{b_1, b_2, \dots, b_n\}$. However, some of the joint commands of these states are *parametric*: instead of encoding fixed joint angles or velocities, they introduce a variable (called a *parameter* of the behavior) that can be assigned their value whenever the behavior is called. Additionally, we associate with each EAP behavior a *controller function* $f : \mathbb{R}^n \rightarrow \mathbf{p}$, which takes as input information about the robot and environment and produces as output the parameters of the behavior. This function is a feedback controller which lets the behavior adapt to environment conditions.

EAP behaviors expand the capabilities of our system. For example, consider a single SMORES-EP module, which can drive on smooth terrain using its two wheels. Using VSPARC, we can create a parametric **Drive** behavior that commands it to turn its wheels, assigning the wheel velocities to two parameters, e.g. $\mathbf{p} = \{V_{\text{left}}, V_{\text{right}}\}$. Using Python, we can now write a controller function for path following, taking as input the current location of the module and producing as output appropriate parameter values (wheel velocities) to drive the module along the path. In Section 7.5.2, we demonstrate how a similar **Drive** behavior and a path planner are used to direct a module to explore different regions on a tabletop.

As another example, consider the **Backhoe** configuration in Table 5. Using VSPARC, we can create a behavior that assigns parameters to the angles of all pan and tilt joints of the arm, providing access to the 7-DOF forward kinematics of the robot. For the controller function, we can write code that takes the position of an object as input and solves an inverse kinematics problem, providing output joint angles that cause the arm to touch an object.

As the above examples imply, EAP behaviors have a two-step design process. First, a parametric behavior is created using VSPARC, which we have extended to allow users to assign any joint value to a parameter rather than a fixed value. This process is no more difficult than creating a non-parametric behavior. Next, a controller function is written, to provide the mapping from sensor data to parameter values. Controller functions can be quite sophisticated (examples include motion planners and feedback controllers) and are typically written in Python by an expert user. However, if existing controller functions are available, novice users can re-use them to produce new EAP behaviors. For example, the path-following controller developed for a single module could be re-used by a novice user to create a similar behavior for the **DoubleDriver** configuration (Table 5), which is also capable of differential drive.

7.4.4 Reactive Controller Synthesis and Execution with the Library

In this section, we describe how our high-level mission planner synthesizes and executes controllers capable of accomplishing tasks using configurations and behaviors from the design library. This process has three parts: (1) matching library entries with boolean variables, (2) generating additional LTL constraints imposed by mapping, and (3) executing the controller. This framework is illustrated in Figure 40, and described in the following subsections.

Matching library entries with boolean variables

In our high-level mission planner, users specify tasks using robot and environment variables that abstract robot actions and environmental conditions, as discussed in Section 7.3.2. They label each variable with behavior and environment properties, to encode the desired functionality and constraints. Our system searches the design library for a set of library entries that satisfy the properties, and maps them to the corresponding boolean variable. Consider an example robot task specification:

if the robot senses **Cup** then do **Push**.

The robot variable **Push** might be described with:

$$P = \{ \{(\text{Cup_Mass}, [1, 3])\}, \\ \{(\text{Action}, [\text{Drive}]), (\text{Speed}, [1])\} \}$$

indicating that robot needs to be able to drive with speed of 1 with a cup that weights 1 to 3 module-weights. With this specification, we can search through the robot design library to find a set of library entries $L_y = \{l_1, \dots, l_k\}$ that satisfies all properties in the set P .

Generating additional LTL formulas imposed by matching

During the matching process, additional necessary LTL constraints are automatically created among the robot variables. Consider a set of robot boolean variables \mathcal{Y} used in a task specification. We define a mapping relation $\lambda : \mathcal{Y} \rightarrow 2^{\mathcal{L}}$ that maps each variable $y \in \mathcal{Y}$ to a set of library entries L_y that satisfies the user specified set of properties P for y . We say a library entry l can *implement* a variable y if $l \in \lambda(y)$. For any $y \in \mathcal{Y}$, if $\lambda(y) = \emptyset$, we need to make sure variable y is never **True**, because no library entry can implement y . For any $y, y' \in \mathcal{Y}$, if $\lambda(y) \cap \lambda(y') = \emptyset$, we need to make sure variable y and y' can never be **True** at the same time, because there does not exist a library entry that can implement both y and y' . To encode the mutual exclusion between robot variables into the task specification, we specify them in the form of LTL formulas that are used together with the original task specification to generate robot controllers during synthesis.

Controller Execution

The synthesized finite-state automaton can be used to control simulated or the physical robots. If synthesis fails, possibly due to lack of library entries that implement some robot variables, LTLMoP will notify the user, who can then design suitable configurations and behaviors with VSPARC.

A synthesized controller is executed by running behaviors based on the value of each robot action variable. If a variable maps to a non-parametric behavior, the behavior is simply executed when the variable becomes **True**. A behavior is stopped when the corresponding variable becomes **False**.

To execute an environmentally-adaptive parametric behavior, the values of all parametric joint commands are decided during execution by calling the controller function each time the behavior is executed. For example, if the robot variable **Explore** matches with the EAP **Drive** behavior of the **Single Module** configuration, the behavior will be executed whenever **Explore** is **True**. A path planner function computes values of parameters in **Drive** in order

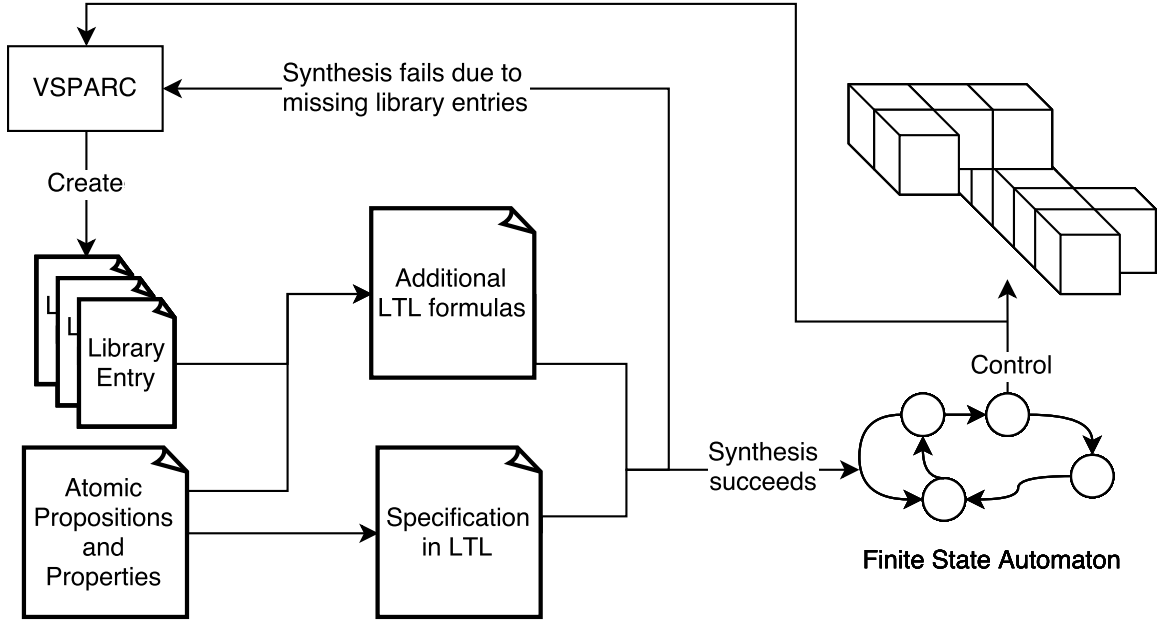


Figure 40: Controller synthesis and execution

to control the robot as a two-wheel differential-drive car.

If two consecutive behaviors must be satisfied by two different configurations, reconfiguration is required. To reduce overall mission time, when multiple behaviors match with a robot boolean variable, we avoid unnecessary reconfiguration by biasing towards the behavior that requires no reconfiguration.

7.5 Experimental Results

We validate the capabilities of our system through experiments in simulation and hardware, illustrated in Figures 43, 44, and 45, as well as the attached video. Faced with various task requirements, the system responds by synthesizing appropriate solutions. The simulation experiments demonstrate how the high-level mission planner can automatically synthesize and execute solutions to tasks using configurations and behaviors from the library. The hardware experiments validate that the system is capable of accomplishing complex physical tasks, such as carrying objects and climbing ledges.

7.5.1 Simulated Task Scenarios

We present two simulated task scenarios. A straightforward task is matched with a simple solution that uses one configuration, while a more complex task is addressed by reconfiguring between three different configurations, to leverage their wide-ranging capabilities.

Scenario 1

In Scenario 1, our system must solve a multi-part task in the environment shown at the top of Figure 41. The environment includes a button, a lightweight block, a gap in the ground, and a ramp, all in a straight line. Pressing the button causes the block to drop to the

Action Definition	Properties
pushButton:	type = Manipulation_Push height = 1.5
pushBox:	type = Manipulation_Push payload = 2 distance_x = 3
climb:	type = Locomotion drive = Straight terrain = Sloped

Table 6: High-level Action Definitions for Scenario 1

ground, where it can be pushed into the gap, forming a bridge between the flat region and ramp. When the task begins, the robot is initially positioned in front of the button. The objective is to reach a goal area at the top of the ramp. The high-level action definitions for this task are provided in Table 6.

After searching the library, the high-level mission planner discovers that the **rollingLoop** configuration has behaviors that satisfy the requirements of all three actions needed for this task (See Table 5). To complete the task, the mission planner synthesizes a controller that commands the loop to press the button, push the block into the gap, and ascend the ramp, as shown in Figure 41.

In response to this straightforward task, our system produces a simple solution. As discussed in Section 7.4.4, the system attempts to minimize reconfiguration when completing a task, and so will opt to solve the entire task with a single configuration whenever possible.

Scenario 2

Like Scenario 1, Scenario 2 requires the robot to move from a starting position to a goal position. However, several small changes have been made to the environment that makes the task more difficult. The button has been moved to the side of the map, and floats at a height of 4 module-lengths above the ground. The box is twice as heavy, weighing 4 module-weights rather than 2. The ramp has been replaced with stairs with a step height of 0.75 module-lengths. Table 7 provides the high-level action definitions for this scenario.

These changes make it impossible for the **rollingLoop** to complete the task - it can't reach the button, it's not strong enough to push the block, and it can't ascend steps more than 0.25 module-lengths high. Instead, the high-level planner compiles a more complicated controller that uses behaviors from three different configurations in the library, shown in Figure 41. To push the button, the planner selects the **backhoe**, because it is the only configuration with a large enough vertical workspace. To push the block into the gap, the robot reconfigures into the **doubleDriver**, which is capable of driving, turning, and pushing objects as heavy as 5 module-weights. To climb the stairs, the robot reconfigures into the **stairClimber**, which can easily ascend 0.75 module-length steps.

This scenario demonstrates how our system leverages the flexibility of modular robots. This challenging task requires the diverse capabilities provided by all three configurations, and could not be accomplished by any one of them alone. Note that for the purposes of this work we do not provide strategies to autonomously perform self-reconfiguration. Instead,

Action Definition	Properties
<code>pushButton:</code>	type = Manipulation_Push height = 4
<code>pushBox:</code>	type = Manipulation_Push payload = 4 distance_x = 3
<code>climb:</code>	type = Locomotion drive = Straight ledge height = 0.75

Table 7: High-level Action Definitions for Scenario 2

we assume that the robot can self-reconfigure between any two configurations as long as the initial configuration has an equal or greater number of modules than the final configuration. This does not fundamentally limit the power of our system: techniques for autonomous self-reconfiguration with SMORES-EP have been recently developed, and are being published in parallel.

7.5.2 Hardware Experiments

Our hardware experiments demonstrate that our system can accomplish a complex physical task using physical SMORES-EP robot modules. The robot is required to clean the top of a table, operating in the environment shown in Figure 42. To do so, the robot must first move a waste bin from its initial location (labelled “Pickup”) to a target location next to the table (labelled “Dropoff”). Then, the robot must climb to the top of the table and explore the surface. Whenever it encounters an object, it must react appropriately: if it is garbage, it should push it off the table and into the waste bin, and if not, it should notify a human to remove it.

This experiment showcases the seamless translation of behaviors from the VSPARC simulator to hardware, and the ability to use the LTLMoP high-level planner to create mission plans that can be directly executed by the modules. AprilTags tracked by an overhead camera provide information about the position of modules and objects in the environment, serving as sensory feedback for the high-level planner.

This experiment also demonstrates how the design library is continually expanded as users develop designs to address new tasks. While the library encompasses a wide range of functionality, it is by no means complete: when a high-level specification was first created for this experiment, the mission planner reported that it could not be satisfied using existing elements in the library. Consequently, two new configurations (the `swerveLifter` and `snake7` configurations) were created, and low-level behaviors were iteratively developed to fulfill the needs of each component of the task. Once these configurations and behaviors were made available in the library, the high-level planner was able to successfully synthesize and execute controllers to accomplish the tasks.

Moving the Waste Bin

The robot begins its task in region `Start1`, and must move the waste bin from `Pickup` to `Dropoff`, a distance of 10 module lengths. Once the waste bin is in place beside the table,

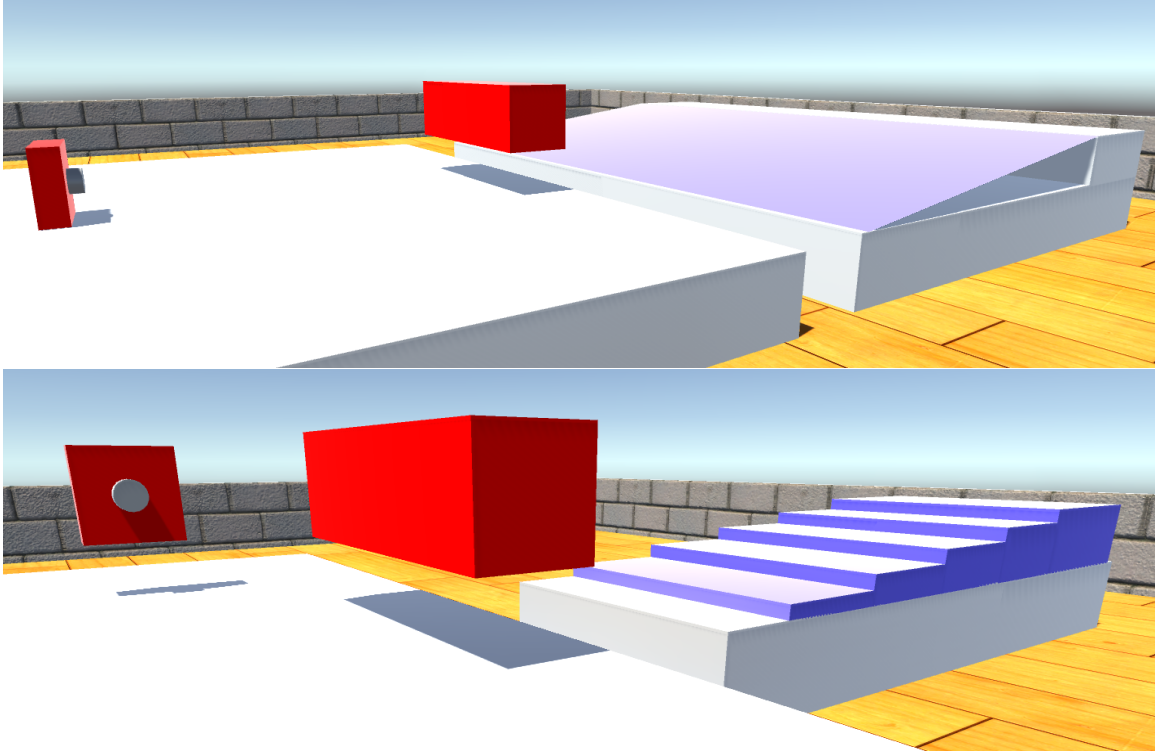


Figure 41: Environments for Scenarios 1 (top) and 2 (bottom) in the simulator.

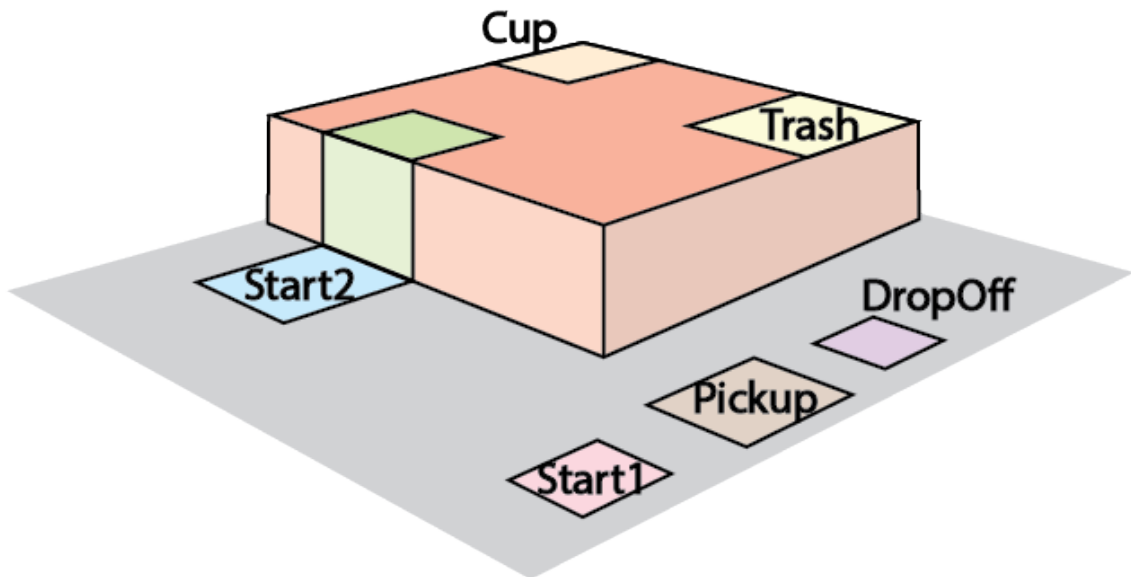


Figure 42: Map of the hardware demo

the robot must travel to the edge of the table (**Start2**), where it can begin the next phase of the task (exploring the tabletop).

The waste bin is a box supported by four legs, making it impossible for any design less than two module-heights tall to push it. This constraint rules out most car-like configurations in the library. The 10-module distance over which the bin must be transported imposes a workspace requirement that rules out all stationary manipulators. Consequently, the **swerveLifter** configuration was designed to meet all the criteria. The **swerveLifter** uses four SMORES-EP modules as powered caster wheels, allowing omnidirectional movement (sometimes called *swerve drive*). It can also raise and lower, enabling it to lift and carry objects by driving underneath them.

The high-level description of this phase of the task is shown in Specification 1, and Figure 44 shows how the robot completes it. The task is reactive: the robot waits until it senses the waste bin before beginning the **pickup** action (Line 3 of Specification 1). Once the waste bin appears (i.e. the AprilTag marking it comes the camera view), the robot lowers itself, drives beneath the waste bin, and carries it to the **Dropoff** region. It then moves back out from beneath the waste bin, and executes a series of omnidirectional driving behaviors to travel to the edge of the table.

Specification 1 Moving the Wastebin

1. **carry** is set on **pickup** and reset on false
 2. **dropped** is set on **drop** and reset on false
 3. do **pickup** if and only if you were sensing **wasteBin**
and you are not activating **carry**
 4. do **goToTable** if and only if you are activating **dropped**
 5. do **drop** if and only if you were activating **carry**
and you are not activating **dropped**
-

Table Exploration

With the waste bin in place, the robot begins the second phase of the task: cleaning the top of the table. The robot needs to climb to the tabletop, explore, and react to what it finds. The **snake7** configuration was designed to be capable to do this. As shown in Table 5, the **snake7** configuration can use its **climbup** and **climbdn** behaviors to ascend and descend ledges up to 3 module-heights tall. However, it is unable to lift its entire body up to the tabletop, and even if it could, it would be too large to effectively explore. Instead, the robot reconfigures, detaching the front module of the snake to act as a **module1** configuration that can use its EAP behavior **differentialDrive** to explore the tabletop, and its **spin**, and **push** behaviors to clean.

Specification 2 provides the high-level task description, and Figure 45 shows the robot completing the task. The robot begins in the **snake7** configuration, positioned at the edge of the table in the **ground** region. An AprilTag is fixed to the front module of the snake, allowing the mission planner to determine its location at all times. Sensing that it is in the **ground** region, the **snake7** executes **climbup** (line 8 of Specification 2). After climbing, the mission planner senses that the head of the snake has reached the **dock** region at the edge of the tabletop, and executes the **undock** behavior to detach the head module from

the snake (line 6), allowing it to operate on its own as a `module1`.

The module then uses `differentialDrive` to visit two regions of interest on the tabletop (`loc1` and `loc2`). `differentialDrive` is an EAP behavior that allows the robot to explore its environment in a continuous fashion. The driving behavior and its parameters are the same as the driving behavior presented as an example in Section 7.4.3: two parameters specify the left and right wheel velocities. The controller function is a potential field path planner that maps the robot’s current position (sensed via AprilTag) to a desired linear and angular velocity, which are converted to wheel velocities.

When it reaches `loc1`, the robot senses a coffee mug (marked with an AprilTag), and responds by executing a `spin` behavior to notify a nearby human that it should be removed (line 1). When it reaches `loc2`, it senses a piece of trash, and it correctly responds by performing a `push` to move it off the table and into the waste bin. Having fully explored the table, the module returns to the dock point and re-attaches to the body of the snake (line 5). The snake then executes `climbdown` to descend back to the floor, completing its mission.

Specification 2 Cleaning the Tabletop

1. if you are sensing `mug` then do `spin`
 2. if you are sensing `trash` then do `push`
 3. `loc1visited` is set on `loc1` and reset on false
 4. `loc2visited` is set on `loc2` and reset on false
 5. do `docking` if and only if you were in `dock` and you are activating (`loc1visited` and `loc2visited`)
 6. do `undock` if and only if you were in `dock` and you are not activating (`loc1visited` or `loc2visited`)
 7. do `climbdown` if and only if you were in `dock` and you activated (`loc1visited` and `loc2visited`)
 8. do `climbup` if and only if you were in `ground` and you are not activating (`loc1visited` or `loc2visited`)
 9. infinitely often do `docking`
-

Challenges

In general, the hardware experiment was successful, with the high-level planner successfully executing library behaviors to complete this task. While running the experiment, several notable challenges were encountered. During the first phase (moving the waste bin), achieving accurate steering with the `swerveLifter` proved difficult. The `swerveLifter` steers by aligning four caster wheels in the same direction, a process that is sensitive to encoder calibration errors across modules. Recently, more sophisticated calibration procedures for the SMORES-EP encoders have been developed, and encoder performance has been improved [93].

During the second phase of the experiment (exploring the tabletop), careful initial positioning was required for the open-loop `climbUp` behavior to succeed - in several trials, the snake was started too close to the ledge, causing it to collide with the corner of the table and break. This problem could be alleviated by developing an EAP behavior allowing the robot to autonomously drive to the appropriate distance before beginning to climb.

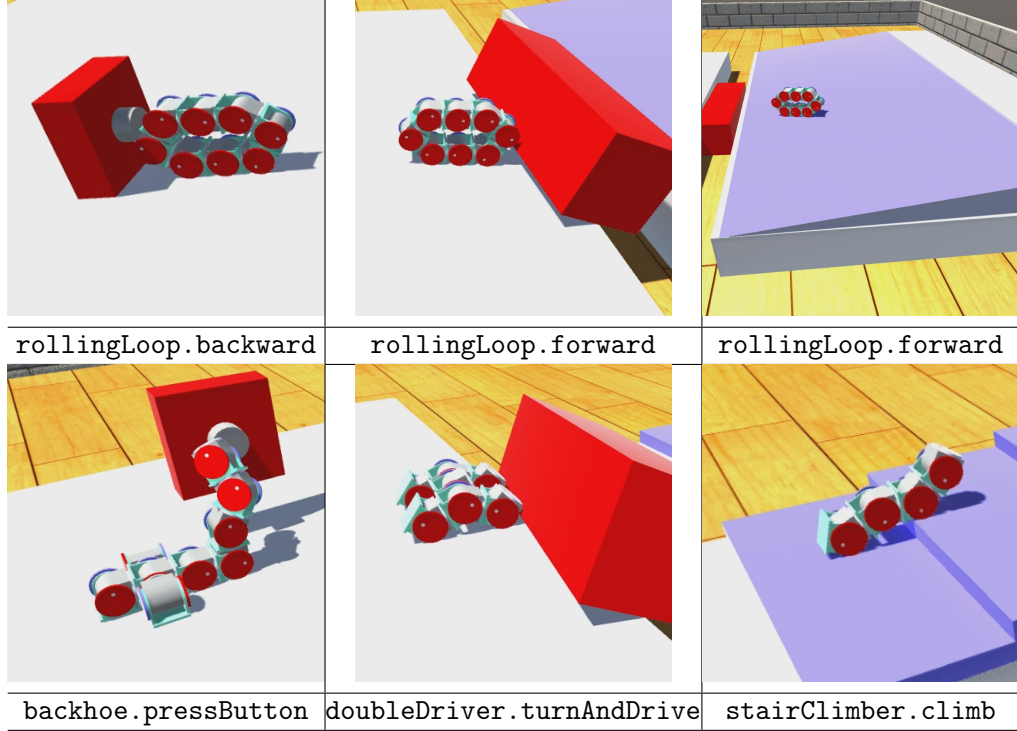


Figure 43: Simulated Demo

In both phases of the experiment, limited magnetic connector strength between modules presented a significant challenge. The `swerveLifter` configuration had to be constructed with a passive cube in its center in order to perform its raising and lowering behaviors without breaking. During descent from the table, bending forces experienced at the center of the `snake7` configuration would sometimes cause connections between modules to break.

The limited strength of the magnetic connectors can be viewed as a trade-off for ease of reconfiguration. Connection and disconnection between the head and body of the snake takes very little time, and the forgiving area-of-acceptance of the connector [92] makes it possible to dock the head of the snake to the body even though the exact position of the body is not known (only the head module had an AprilTag). Autonomous docking succeeded about 25% of the time. This performance could be improved by applying more recently developed techniques for autonomous self-reconfiguration with SMORES-EP.

7.6 Discussion and Future Work

7.6.1 Simulator-to-hardware translation

Translation of behaviors from VSPARC to the hardware was largely successful, and the ability to prototype designs and behaviors in a simulator resulted in significant time savings over prototyping in hardware. Disparities between performance in the simulator and hardware tended to arise from real-world phenomena the simulator did not model accurately. For example, variability in magnetic connector strength (which differs from module to module [92]) sometimes resulted in connections breaking unexpectedly, and encoder calibration

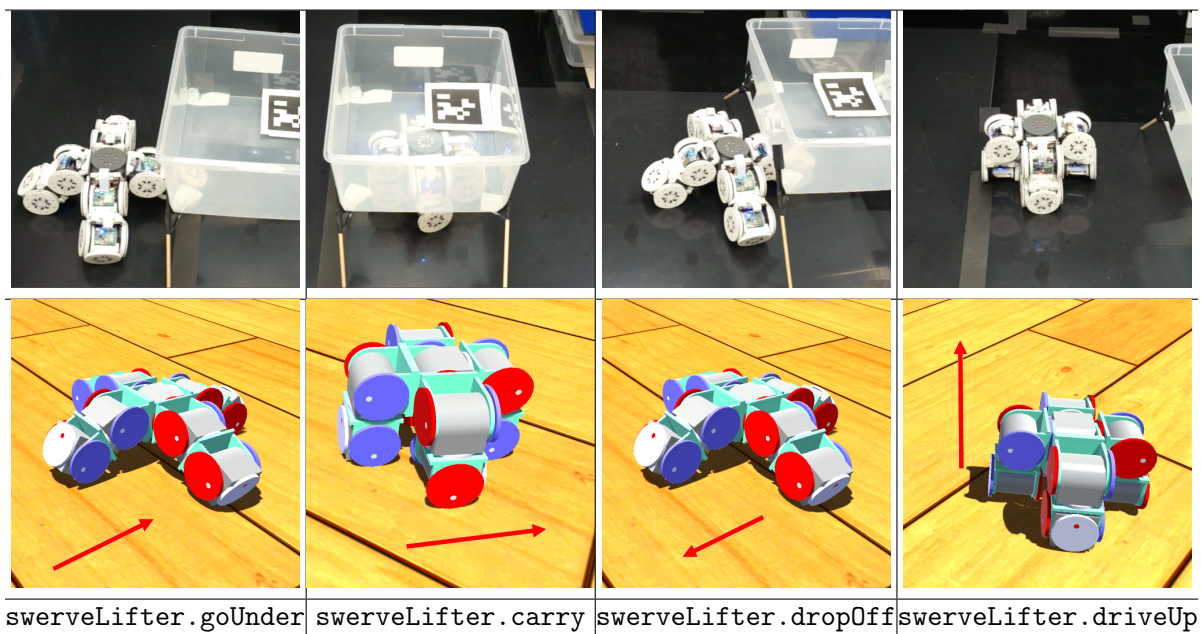


Figure 44: Moving the Waste Bin

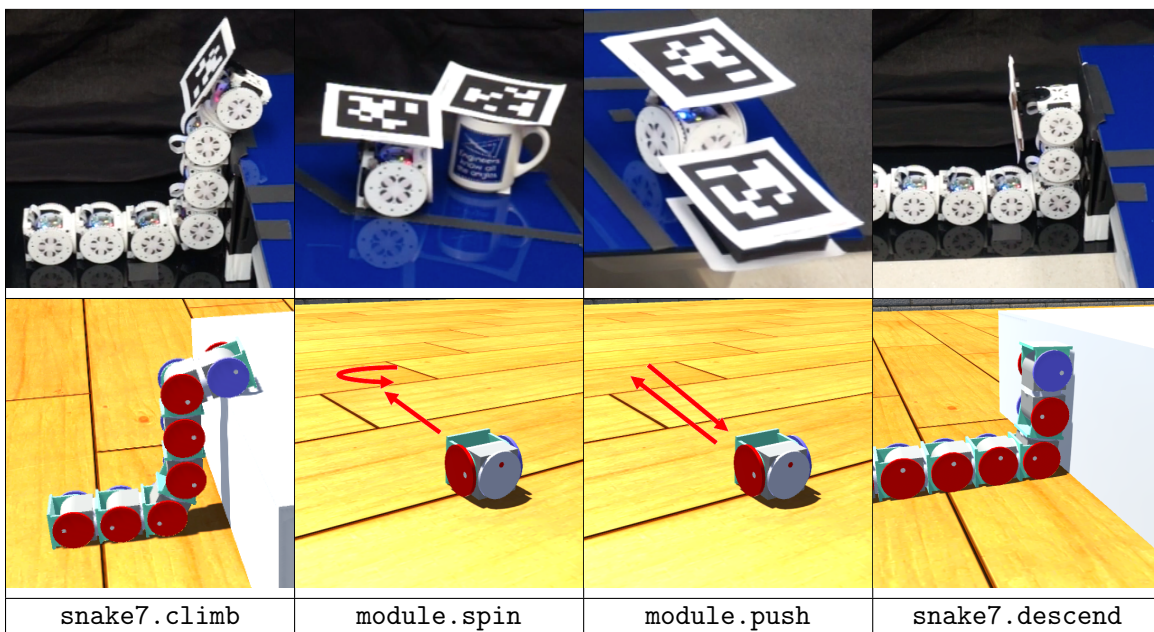


Figure 45: Cleaning the Table

errors could cause behaviors requiring very precise position control to perform poorly.

Incorporation of on-board sensing will allow our system to operate autonomously in unknown environments. At the time of writing, a “brain module” has been developed that allows SMORES-EP clusters to carry an RGB-D camera and computer unit. In the future, VSPARC could be expanded to include simulated sensing capabilities, making it easier to develop closed-loop EAP behaviors.

7.6.2 Library Creation: Lessons Learned

Early on in the development of this system, we intended to populate our design library through crowdsourcing, using a system such as Amazon Mechanical Turk where a large number of online users could create configurations and behaviors using VSPARC. We quickly realized that this strategy would not produce high-quality designs: developing sophisticated designs and behaviors in the simulator requires skill and experience. Holding hackathons with undergraduate engineers proved to be a much more effective strategy, because participants would become significantly more adept at creating designs and behaviors through hours of practice. Newcomers would typically spend about an hour creating a useful behavior, where well-practiced users would spend about twenty minutes.

Interestingly, users spent significantly more time creating behaviors than configurations. Most users required only a few minutes to build a new configuration and conceive of the fundamental motions they wanted it to perform. The majority of the design time was spent coding joint trajectories to achieve the desired motion while maintaining balance and avoiding connector strength overload. In the future, we hope to incorporate motion planning tools within VSPARC that will allow novice users to specify desired motions without explicitly coding joint angles. Evolutionary techniques will also be explored to generate behaviors automatically.

An existing algorithm for modular robot design embedding detection could also be used to automatically generate behaviors for new configurations [52]. This algorithm can automatically detect when one a subset of the joints of one configuration can be used to replicate the kinematics of another (a condition known as *embedding*), and generates a mapping that can be used to transfer behaviors originally developed for one configuration to any other configuration that embeds it. This could also allow behaviors developed for SMORES-EP to be ported to other modular robot systems, or vice-versa.

7.6.3 Composing Library Elements to Complete Missions

Environment and behavior properties provide an expressive way for the user to specify the requirements of a task. However, the fact that a behavior is labeled with a specific property does not guarantee it will perform as intended in all circumstances. Adapting behaviors to environments different from the one in which they were designed can cause them to fail, as evidenced by the problems in establishing proper initial robot position for the `climbUp` behavior in the table cleaning scenario. Development of more closed-loop parametric behaviors will help address this issue.

Methods for automatically analyzing tasks and environments are actively being researched [89]. Determining optimal sets of environment factors and integrating methods for automatic task analysis and would be an interesting avenue for future work.

It’s worth noting that some behaviors are much more tolerant to varying environments than others. In our hardware experiments with the `stairClimber` configuration, we found

that a single open-loop gait was able to climb steps of several varying sizes with no problems. Establishing confidence bounds on behavior success as a function of environment parameters and including this information in the library is future work.

7.7 Conclusion

We presented a system for addressing high-level tasks with modular self-reconfigurable robots. We demonstrated how our physics-based simulator allows SMORES-EP configurations and behaviors to be easily created and stored in the design library, and how our framework for labeling each entry in the library with descriptive properties allows them to be organized by functionality. Integration with a high-level mission planner allowed users to provide high-level task specifications, which were used to synthesize reactive controllers that use configurations and behaviors from the library. The capabilities of our system are validated through experiments in simulation and with physical modular robots. We also expanded the system by introducing environmentally-adaptive parametric behaviors, which allowed sophisticated motion planners and feedback controllers to be used within our framework.

Chapter 8

Autonomy

The theoretical ability of modular robots to reconfigure in response to complex tasks in *a priori* unknown environments has frequently been cited as an advantage, but never demonstrated. Today, this vision remains a major motivator for work in the field.

This chapter presents the first modular robot system capable of autonomously completing high-level tasks by reactively reconfiguring to meet the needs of a perceived, *a priori* unknown environment. The system integrates perception, high-level planning, and modular hardware, and is validated in three hardware demonstrations. Based on a high-level task specification, a modular robot autonomously explores an unknown environment, decides when and how to reconfigure, and manipulates objects to complete its task. The system architecture balances distributed mechanical elements with centralized perception, planning, and control. By providing a clear example of how a modular robot system can be designed to leverage reactive reconfigurability in unknown environments, we have begun to lay the groundwork for MSRR systems to address tasks in the real world.

This chapter excerpts heavily from the author’s work in [16]. Much credit is due to co-authors Jonathan Daudelin, Gangyuan Jing, Mark Campbell, and Hadas Kress-Gazit, all from Cornell University, who contributed significantly to this work.

8.1 Introduction

Since the field of modular robotics was in its nascence, researchers have presented a vision of flexible, reactive systems operating in unknown environments. Modular self-reconfigurable robots would be able to enter unknown environments, assess their surroundings, and self-reconfigure to take on a form suitable to the task and environment at hand [105]. Today, this vision remains a major motivator for work in the field [107].

Continued research in MSRR has resulted in substantial advancement. Existing research has demonstrated MSRR self-reconfiguring, assuming interesting morphologies, and exhibiting various forms of locomotion, as well as methods for programming, controlling, and simulating modular robots [12, 22, 26, 39, 52, 60, 62, 72, 76–78, 102, 105, 108]. However, achieving autonomous operation of a self-reconfigurable robot in unknown environments requires the ability to explore, gather information about the environment, consider the requirements of a high-level task, select configurations whose capabilities match the requirements of task and environment, transform, and perform actions (like manipulating objects) to complete tasks. Existing systems provide partial sets of these capabilities. Many systems have demonstrated

limited autonomy, relying on beacons for mapping [20, 29] and human input for high-level decision making [21, 59]. Others have demonstrated swarm self-assembly to address basic tasks like hill-climbing and gap-crossing [31, 70]. While these existing systems all represent advancements, none have demonstrated fully autonomous, reactive self-reconfiguration to address high-level tasks. A more detailed overview of existing systems can be found in Section 8.5.

This chapter presents a novel system allowing modular robots to complete complex high-level tasks autonomously. The system automatically selects appropriate behaviors to meet the requirements of the task and constraints of the perceived environment. Whenever the task and environment require a particular capability, the robot autonomously self-reconfigures to a configuration that has that capability. The success of this system is a product of our choice of system architecture, which balances distributed and centralized elements. Distributed, homogeneous robot modules provide flexibility, reconfiguring between morphologies to access a range of functionality. Centralized sensing, perception, and high-level mission planning components provide autonomy and decision-making capabilities. Tight integration between the distributed low-level and centralized high-level elements allows us to leverage advantages of distributed and centralized architectures.

The system is validated in three hardware demonstrations, showing that, based on a high-level task specification, the robot autonomously explores an unknown environment, decides if, when, and how to reconfigure, and manipulates objects to complete its task. By providing a clear example of how a modular robot system can be designed to leverage reactive reconfigurability in unknown environments, we have begun to lay the groundwork for reconfigurable systems to address tasks in the real world.

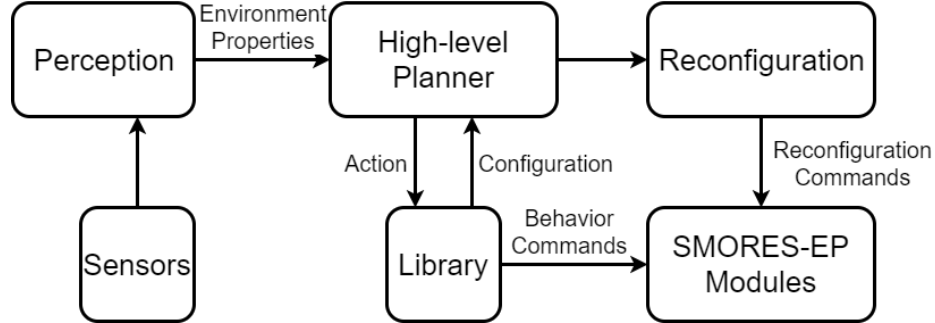


Figure 46: System Overview Flowchart

8.2 Results

We demonstrate an autonomous, perception-informed, modular robot system that reactively adapts to unknown environments via reconfiguration in order to perform complex tasks. The system hardware consists of a set of **robot modules** (that can move independently and dock with each other to form larger morphologies), and a **sensor module** that contains multiple cameras and a small computer for collecting and processing data from the environment. Software components consist of a **high-level planner** to direct robot actions and reconfiguration, and **perception algorithms** to perform mapping, navigation, and classi-



Figure 47: Sensor Module with labelled components. UP board and battery are inside the body.

fication of the environment. Our implementation is built around the SMORES-EP modular robot [92], but could be adapted to work with other modular robots.

Our system is the first to demonstrate high-level decision-making in conjunction with reconfiguration in an autonomous setting. In three hardware demonstrations, the robot explores an *a priori* unknown environment, and acts autonomously to complete a complex task. Tasks are specified at a high level: users do not explicitly specify which configurations and behaviors the robot should use; rather, tasks are specified in terms of *behavior properties*, which describe desired effects and outcomes [40]. During task execution, the high-level planner gathers information about the environment and reactively selects appropriate behaviors from a design library, fulfilling the requirements of the task while respecting the constraints of the environment. Different configurations of the robot have different capabilities (sets of behaviors). Whenever the high-level planner recognizes that task and environment require a behavior the current robot configuration cannot execute, it directs the robot to reconfigure to a different configuration that can execute the behavior.

Figure 48 shows the environments used for each demonstration, and Figure 49 shows snapshots during each of the demonstrations. A video of all three demonstrations is available as part of the supplementary material.

In Demonstration I, the robot must find, retrieve, and deliver all pink- and green-colored metal garbage to a designated drop-off zone for recycling, which is marked with a blue square on the wall. The demonstration environment contains two objects to be retrieved: a green soda can in an unobstructed area, and a pink spool of wire in a narrow gap between two trash cans. Various obstacles are placed in the environment to restrict navigation. When performing the task, the robot first explores using the “Car” configuration. Once it locates the pink object, it recognizes the surrounding environment as a “tunnel” type, and the high-

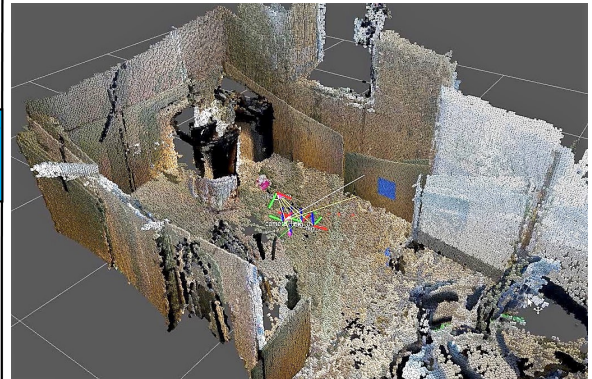
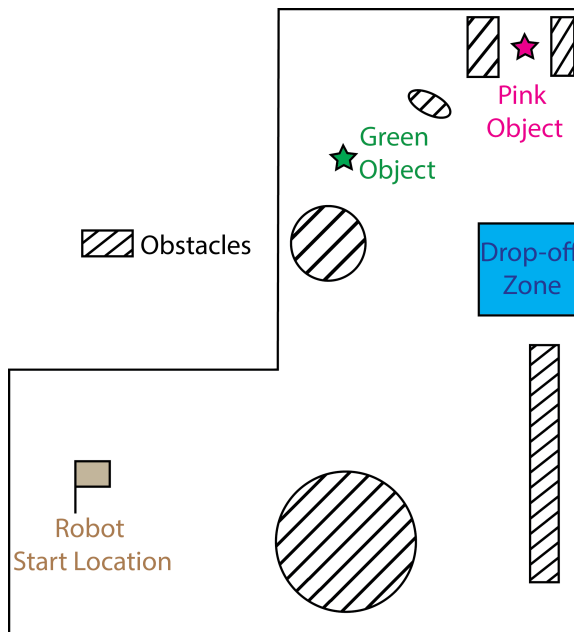
level planner reactively directs the robot to reconfigure to the “Proboscis” configuration, which is then used to reach in between the trash cans and pull the object out in the open. The robot then reconfigures to the “Car,” retrieves the object, and delivers it to the drop-off zone which the system had previously seen and marked during exploration. Figure 48b shows the resulting 3D map created from SLAM during the demonstration.

For Demonstrations II and III, the high-level task specification is the following: start with an object, explore until finding a delivery location, and deliver the object there. Each demonstration uses a different environment. For Demonstration II, the robot must place a circuit board in a mailbox (marked with pink-colored tape) at the top of a set of stairs with other obstacles in the environment. For Demonstration III, the robot must place a postage stamp high up on the box that is sitting in the open.

For Demonstration II, the robot begins exploring in the “Scorpion” configuration. Shortly, the robot observes and recognizes the mailbox, and characterizes the surrounding environment as “stairs.” Based on this characterization, the high-level planner directs the robot to use the “Snake” configuration to traverse the stairs. Using the 3D map and characterization of the environment surrounding the mail bin, the robot navigates to a point directly in front of the stairs, faces the bin, and reconfigures to the “Snake” configuration. The robot then executes the stair climbing gait to reach the mail bin, and drops the circuit successfully. It then descends the stairs and reconfigures back to the “Scorpion” configuration to end the mission.




For Demonstration III, the robot begins in the “Car” configuration, and cannot see the package from its starting location. After a short period of exploration, the robot identifies the pink square marking the package. The pink square is unobstructed, but is approximately 25cm above the ground; the system correctly characterizes this as the “high”-type environment, and recognizes that reconfiguration will be needed to reach up and place the stamp on the target. The robot navigates to a position directly in front of the package, reconfigures to the “Proboscis” configuration, and executes the “highReach” behavior to place the stamp on the target, completing its task.

It should be noted that all experiments were run using the same software architecture, same SMORES-EP modules, and system described in this chapter. The library of behaviors was extended with new entries for Demonstrations II and III to expand the system abilities for the new challenges presented in the new environments. Minor adjustments to motor speeds, SLAM parameters, and the low-level reconfiguration controller. In addition, Demonstrations II and III used a newer, improved 3D sensor, and therefore a different sensor driver was used.



(b) Volumetric map of environment 1 built by visual SLAM

(a) Diagram of Demonstration I environment

Environment Setup	Task Description
	Demonstration I: Explore environment to find all pink or green objects and blue dropoff zone. Deliver all objects to dropoff zone.
	Demonstration II: Explore environment to find mailbox, then deliver a circuit to the box.
	Demonstration III: Explore environment to find package, then place a stamp on the package.

(c) Environments and tasks for hardware demonstrations

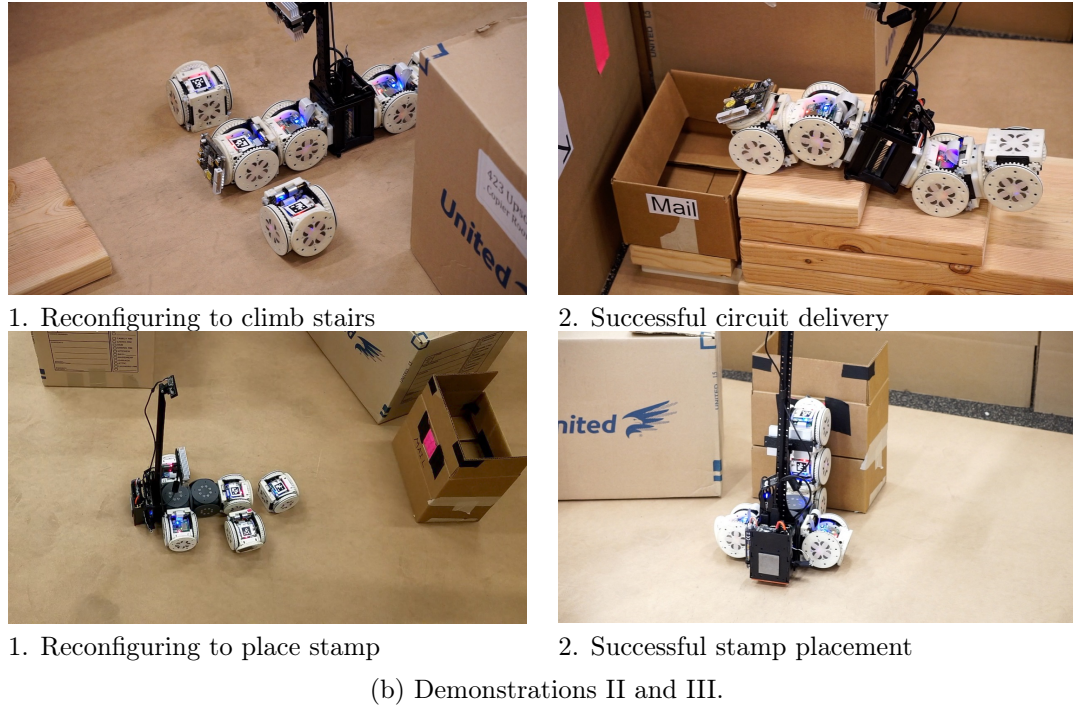
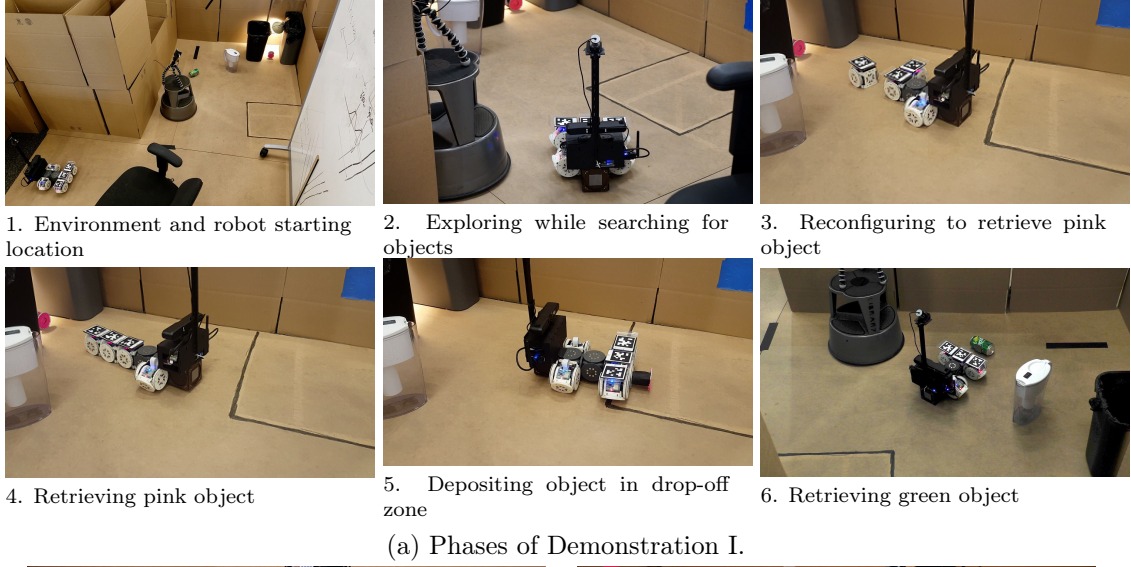


Figure 49: Demonstrations 1, 2, and 3

8.3 Discussion

Modular self-reconfigurable robots are by their nature mechanically distributed, and as a result lend themselves naturally to distributed planning, sensing, and control. Most past systems have used entirely distributed frameworks [20, 59, 62, 70, 77, 108]. Our system is designed differently. It is distributed at the low level (hardware), but centralized at the high level (planning and perception), leveraging the advantages of both design paradigms.

The three scenarios in the demonstrations showcase a range of different ways SMORES-

EP can interact with environments and objects: movement over flat ground, fitting into tight spaces, reaching up high, climbing over rough terrain, and manipulating objects. This broad range of functionality is only accessible to SMORES-EP by reconfiguring between different morphologies.

The high-level planner, environment characterization tools, and library work together to allow tasks to be represented in a flexible and reactive manner. For example, at the high level, Demonstrations II and III are the same task: deliver an object at a point of interest. However, after characterizing the different environments (“High” in II, “Stairs” in III), the system automatically determines that different configurations and behaviors are required to complete each task: the Proboscis to reach up high, and the Snake to climb the stairs. Similarly, in Demonstration I there is no high-level distinction between the green and pink objects - the robot is simply asked to retrieve all objects it finds. The sensed environment once again dictates the choice of behavior: the simple problem (object in the open) is solved in a simple way (with the Car configuration), and the more difficult problem (object in tunnel) is solved in a more sophisticated way (by reconfiguring into the Proboscis). Achieving this level of sophistication in control and decision-making through a distributed architecture would have been significantly more difficult.

Centralized sensing and control during reconfiguration, provided by AprilTags and a centralized path planner, allowed our implementation to transform between configurations more rapidly than previous distributed systems. Each reconfiguration action (a module disconnecting, moving, and reattaching) takes about one minute. In contrast, past systems that utilized distributed sensing and control required 5-15 minutes for single reconfiguration actions [62, 77, 108], which would prohibit their use in the complex tasks and environments that our system demonstrated.

8.3.1 Challenges and Limitations

Through the hardware demonstrations performed with our system, we observed several challenges and opportunities for future improvement with autonomous perception-informed modular systems. All SMORES-EP body modules are identical, and therefore interchangeable for the purposes of reconfiguration. However, the sensor module has a significantly different shape than a SMORES-EP body module, which introduces heterogeneity in a way that complicates motion planning and reconfiguration planning. Configurations and behaviors must be designed to provide the sensor module with an adequate view, and to support its weight and elongated shape. Centralizing sensing also limits reconfiguration: modules can only drive independently in the vicinity of the sensor module, preventing the robot from operating as multiple disparate clusters.

Our high-level planner assumes all underlying components are reliable and robust, so failure of a low-level component can cause the high-level planner to behave unexpectedly, and result in failure of the entire task. Table 8 shows the causes of failure for 24 attempts of Demonstration II (placing the stamp on the package). Nearly all failures are due to an error in one of the low-level components the system relies upon, with 42% of failure due to hardware errors and 38% due to failures in low-level software (object recognition, navigation, environment characterization). This kind of cascading failure is a weakness of centralized, hierarchical systems: distributed systems are often designed so that failure of a single unit can be compensated for by other units, and does not result in global failure.

This lack of robustness represents a challenge, but steps can be taken to address it. Un-

surprisingly, open-loop behaviors (like stair-climbing and reaching up to place the stamp) were vulnerable to small hardware errors and less robust against variations in the environment. For example, if the height of stairs in the actual environment is higher than the property value of the library entry, the stair-climbing behavior is likely to fail. Closing the loop using sensing made exploration and reconfiguration significantly less vulnerable to error. Future systems could be made more robust by introducing more feedback from low-level components to high-level decisions making processes, and by incorporating existing high-level failure-recovery frameworks [51]. Distributed repair strategies could also be explored, to replace malfunctioning modules with nearby working ones on the fly [91].

To implement our perception characterization component, we assumed a simplified set of environment types and implemented a simple characterization function to distinguish between them. This function does not generalize very well to completely unstructured environments and also is not very scalable. Thus, to expand the system to work well for more realistic environments and to distinguish between a large number of environment types, a more general characterization function should be implemented.

This chapter presents the first modular robot system to autonomously complete high-level tasks by reactively reconfiguring in response to its perceived environment and task requirements. In addition, putting the entire system to the test in hardware demonstrations revealed several opportunities for future improvement in such systems.

Reason of failure	Number of times	Percentage
Hardware Issues	10	41.7%
Navigation Failure	3	12.5%
Perception-Related Errors	6	25%
Network Issues	1	4.2%
Human Error	4	16.7%

Table 8: Reasons for demonstration failure.

8.4 Methods and Materials

The following sections discuss the role of each component within the general system architecture. Inter-process communication between the many software components in our implementation is provided by the Robot Operating System (ROS)¹. Figure 46 gives a flowchart of the entire system. For more details of the implementation used in the demonstrations see the Supplementary Materials.

8.4.1 Hardware - Sensor Module

SMORES-EP modules have no sensors that allow them to gather information about their environment. To enable autonomous operation, we introduce a *sensor module*, designed to work with SMORES-EP as shown in Figure 47. The body of the sensor module is a 90mm × 70mm × 70mm box with thin steel plates on its front and back that allow SMORES-EP modules to connect to it. Computation is provided by an UP computing board with an Intel Atom 1.92 GHz processor, 4 GB memory, and a 64 GB hard drive. A USB WiFi adapter

¹<http://www.ros.org>

provides network connectivity. A front-facing Orbecc Astra Mini camera provides RGB-D data, enabling the robot to explore and map its environment and recognize objects of interest. A thin stem extends 40cm above the body, supporting a downward-facing webcam. This camera provides a view of a $0.75\text{m} \times 0.5\text{m}$ area in front of the sensor module, and is used to track AprilTag [68] fiducials for reconfiguration. A 7.4V, 2200mAh LiPo battery provides about one hour of running time.

A single sensor module carried by the cluster of SMORES-EP modules provides centralized sensing and computation. Centralizing sensing and computation has the advantage of facilitating control, task-related decision making, and rapid reconfiguration, but has the disadvantage of introducing physical heterogeneity, making it more difficult to design configurations and behaviors. The shape of the sensor module can be altered by attaching lightweight cubes, which provide passive structure to which modules can connect. Cubes have the same 80mm form factor as SMORES-EP modules, with magnets on all faces for attachment.

8.4.2 Perception and Planning for Information

Completing tasks in unknown environments requires the robot to explore and gain information about its surroundings, and use that information to inform actions and reconfiguration. Our system architecture includes active perception components to perform SLAM, choose waypoints for exploration, and recognize objects and regions of interest. It also includes a framework to characterize the environment in terms of robot capabilities, allowing the high-level planner to reactively reconfigure the robot to adapt to different environment types. Implementations of these tools should be selected to fit the MSRR system being used and types of environments expected to be encountered.

Environment characterization is done using a discrete classifier (using the 3D occupancy grid of the environment as input) to distinguish between a discrete set of environment types corresponding to the library of robot configurations and gaits. To implement our system for a particular MSRR, the classification function must be defined by the user to classify the desired types of environments. For our proof-of-concept hardware demonstrations, we assumed a simplified set of possible environment types around objects of interest. We assumed the object of interest must be in one of four environment types shown in Figure 50e: “tunnel” (the object is in a narrow corridor), “stairs” (the object is at the top of low stairs), “high” (the object is on a wall above the ground), and “free” (the object is on the ground with no obstacles around). Our implemented function performs characterization as follows: When the system recognizes an object in the environment, the characterization function evaluates the 3D information in the object’s surroundings. It creates an occupancy grid around the object location, and denotes all grid cells within a robot-radius of obstacles as unreachable (illustrated in Figure 50f). The algorithm then selects the closest reachable point to the object within 20° of the robot’s line of sight to the object. If the distance from this point to the object is greater than a threshold value and the object is on the ground, the function characterizes the environment as a “tunnel”. If above the ground, it function the environment as a “stairs” environment. If the closest reachable point is under the threshold value, the system assigns a “free” or “high” environment characterization, depending on the height of the colored object.

Based on the environment characterization and target location, the function also returns a waypoint for the robot to position itself to perform its task (or to reconfigure, if necessary).

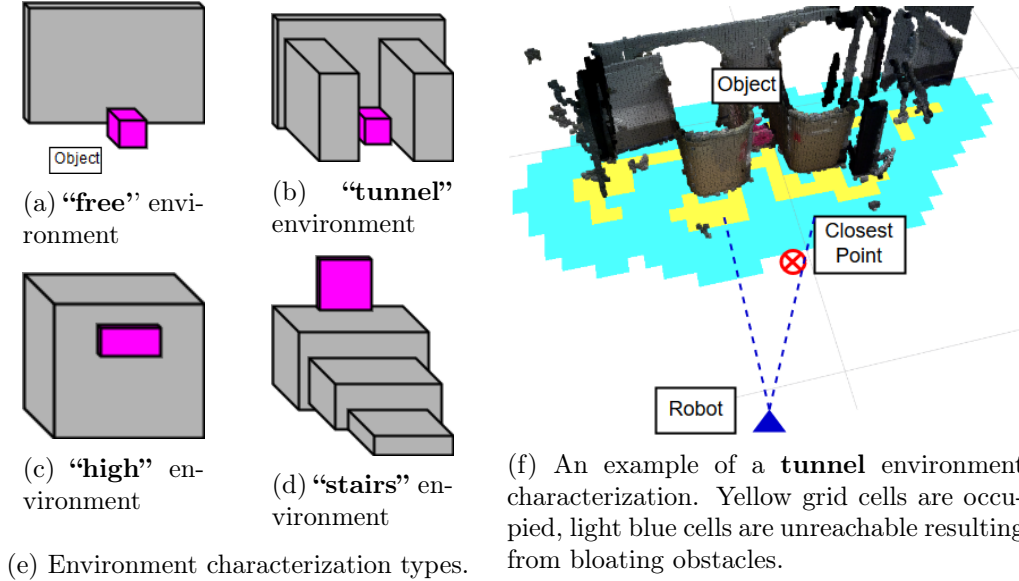


Figure 50: Environment Characterization

In Demonstration II, the environment characterization algorithm directs the robot to drive to a waypoint at the base of the stairs, which is the best place for the robot to reconfigure and begin climbing the stairs.

Our implementation for other components of the perception architecture use previous work and open-source algorithms. The RGB-D SLAM software package RTAB-MAP[48] provides mapping and robot pose. The system incrementally builds a 3D map of the environment and stores the map in an efficient octree-based volumetric map using Octomap[37]. The Next Best View algorithm by Daudelin et. al.[15] enables the system to explore unknown environments by using the current volumetric map of the environment to estimate the next reachable sensor viewpoint that will observe the largest volume of undiscovered portions of objects (the Next Best View). In the example object delivery task, the system begins the task by iteratively navigating to these Next Best View waypoints to explore objects in the environment until discovering the dropoff zone.

To identify objects of interest in the task (such as the dropoff zone), we implemented our system using color detection and tracking. The system recognizes colored objects using CMVision², and tracks them in 3D³ using depth information from the onboard RGB-D sensor. Although we implement object recognition by color, more sophisticated methods could be used instead, under the same system architecture.

8.4.3 High-Level Planning and Library

The system employs the high-level planning framework introduced in Chapter 7, allowing the robot to automatically select appropriate configurations and behaviors to satisfy task requirements specified by the user. As before, the full set of capabilities of the robot is encoded in a design library of configurations and behaviors, which are labeled with properties

²CMVision: <http://www.cs.cmu.edu/~jbruce/cmvision/>

³Lucas Coelho Figueiredo: <https://github.com/lucascoelho91/ballFollower>

describing their capabilities. Table 9 lists ten library entries for the four configurations are used in this work. Compared to the library presented in Chapter 7, the library used in this work is much smaller. Autonomous operation introduces a higher bar for robustness: since the high-level planner is free to choose any configuration or behavior from the library, all entries need to operate predictably and without errors under any conditions in which the high-level planner might select them. Consequently, using a small library of rigorously tested entries proved to be the most effective strategy for this work. For example, when climbing the stairs, noise in localization could sometimes cause the perceived position of the robot to deviate from reality by several centimeters, so the stair-climbing behavior was designed to be tolerant to this uncertainty in initial conditions

As before, the high-level planner selects configuration and behaviors to achieve the specified goals of the task while obeying the constraints of the current environment. In Chapter 7, these environment properties were “sensed” based on the proximity of fiducial markers on the robot and objects. As described in the previous section, we now extract environment properties directly from 3D sensor information. While the environment characterization system is limited to differentiating between environments with only four properties, the system is still able to complete fairly complex tasks.

Configuration	Behavior properties	Environment Types
Car	pickUp	“free”
	drop	“free”
	drive	“free”
Proboscis	pickUp	“tunnel” or “free”
	drop	“tunnel” or “free”
	highReach	“high”
Scorpion	drive	“free”
Snake	climbUp	“stairs”
	climbDown	“stairs”
	drop	“stairs” or “free”

Table 9: A library of robot behaviors

8.4.4 Reconfiguration

When the high-level planner decides to use a new configuration during a task, the robot must reconfigure. We have implemented tools for mobile reconfiguration with SMORES-EP, taking advantage of the fact that individual modules can drive on flat surfaces.

Determining the relative positions of modules during mobile self-reconfiguration is an important challenge. In this work, the localization method is centralized, using a camera carried by the robot to track AprilTag fiducials mounted to individual modules. As discussed in Section 8.4.1, the camera provides a view of a $0.75\text{m} \times 0.5\text{m}$ area on the ground in front of the sensor module. Within this area, the localization system provides pose for any module equipped with an AprilTag marker to perform reconfiguration.

Given an initial configuration and a goal configuration, the reconfiguration controller commands a set of modules to disconnect, move and reconnect in order to form the new

topology of the goal configuration. Figure 51 shows reconfiguration from the “Car” to the “Proboscis” during Demonstration 1. The robot first takes actions to establish the conditions needed for reconfiguration by confirming that the reconfiguration zone is a flat surface free of obstacles (other than the modules themselves). The robot then sets its joint angles so that all modules that need to detach have both of their wheels on the ground, ready to drive. Then the robot performs operations to change the topology of the cluster by detaching a module from the cluster, driving, and re-attaching at its new location in the goal configuration, as shown in Figure 51. Currently, reconfiguration plans from one configuration to another are created manually and stored in the library. However the framework can work with existing assembly planning algorithms ([83, 100]) to generate reconfiguration plans automatically. Because the reconfiguration zone is free of obstacles, the controller compute collision-free paths offline and store them as part of the reconfiguration plan. Once all module movement operations have completed and the goal topology is formed, the robot sets its joints to appropriate angles for the goal configuration to continue performing desired behaviors.

We developed several techniques to ensure reliable connection and disconnection during reconfiguration. When a module disconnects from the cluster, the electro-permanent magnets on the connected faces are turned off. To guarantee a clean break of the magnetic connection, the disconnecting module bends its tilt joint up and down, mechanically separating itself from the cluster. During docking, accurate alignment is crucial to the strength of the magnetic connection [92]. For this reason, rather than driving directly to its final docking location, a module instead drives to a pre-docking waypoint directly in front of its docking location. At the waypoint, the module spins in place slowly until its heading is aligned with the dock point, and then drives in straight to attach. To guarantee a good connection, the module intentionally overdrives its dock point, pushing itself into the cluster while firing its magnets.

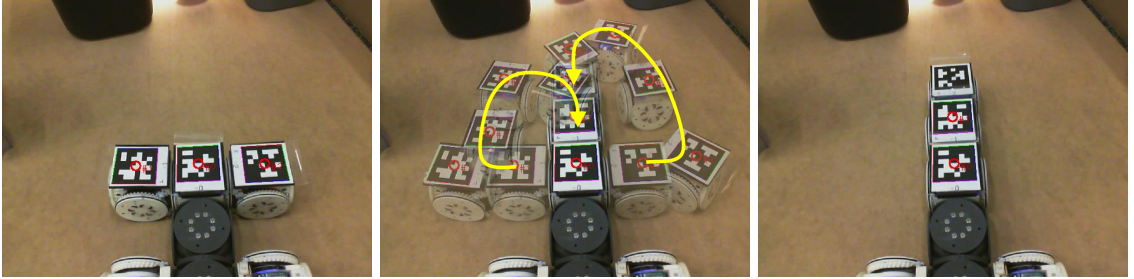
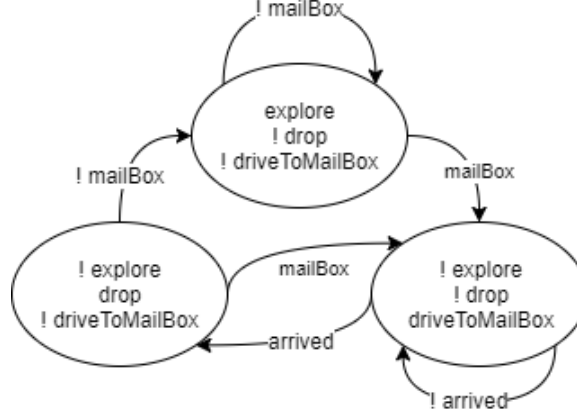


Figure 51: Module movement during reconfiguration. Left: initial configuration (“Car”). Middle: module movement, using AprilTags for localization. Right: final configuration (“Proboscis”).

Specification 1 Drop an object in the mailbox

do **explore** if and only if the robot is not sensing **mailBox**
do **driveToMailBox** if and only if the robot is sensing (**mailBox** and not **arrived**)
do **drop** if and only if the robot is sensing (**arrived** and **mailBox**)

(a) Specification for dropping an object in the mailbox.



(b) The synthesized controller. A proposition with “!” has a value of **False**, and **True** otherwise.

Figure 52: A task specification with the synthesized controller.

8.5 Additional Commentary on Related Work

Here we provide a more detailed overview of prior work in MSRR systems. These systems provide partial sets of the capabilities of our system.

The Millibot system demonstrated mapping when operating as a swarm. Certain members of the swarm are designated as “beacons,” and have known locations. The autonomy of the Millibot swarm is limited: a human operator makes all high-level decisions, and is responsible for navigation using a GUI [29].

The Swarm-Bots system has been applied in exploration [20] and collective manipulation [59] scenarios. Like the Millibots, some members of the swarm act as “beacons” that are assumed to have known location during exploration. In a collective manipulation task, Swarm-Bots have limited autonomy, with a human operator specifying the location of the manipulation target and the global sequence of manipulation actions.

In [70], Swarm-Bots demonstrate swarm self-assembly to climb a hill. Robots exhibit phototaxis, with the goal of moving toward a light source. When robots detect the presence of a hill (using tilt sensors), they aggregate to form a random connected structure to collectively surmount the hill. A similar strategy is employed to cross holes in the ground. In each case, the swarm of robots is loaded with a single self-assembly controller specific to an *a priori* known obstacle type (hill or hole). The robots do not self-reconfigure between specific morphologies, but rather self-assemble, beginning as a disconnected swarm and coming together to form a random connected structure. In our work, a modular robot completes high-level tasks by autonomously self-reconfiguring between specific morphologies

with different capabilities. Our system differentiates between several types of environments using RGB-D data, and may choose to use different morphologies to solve a given high-level task in different environments.

The swarmanoid project (successor to the swarm-bots), uses a heterogeneous swarm of ground and flying robots (called “hand-”, “foot-”, and “eye-” bots) to perform exploration and object retrieval tasks [21]. Robotic elements of the swarmanoid system connect and disconnect to complete the task, but the decision to take this action is not made autonomously by the robot in response to sensed environment conditions. While the location of the object to be retrieved is unknown, the method for retrieval is known and constant.

Self-reconfiguration has been demonstrated with several other modular robot systems. CKbot, Conro, and MTRAN have all demonstrated the ability to join disconnected clusters of modules together [62, 77, 108]. In order to align, Conro uses infra-red sensors on the docking faces of the modules, while CKBot and MTRAN use a separate sensor module on each cluster. In all cases, individual clusters locate and servo towards each other until they are close enough to dock. These experiments do not include any planning or sequencing of multiple reconfiguration actions in order to create a goal structure appropriate for a task. Additionally, modules are not individually mobile, and mobile clusters of modules are limited to slow crawling gaits. Consequently, reconfiguration is very time consuming, with a single connection requiring 5-15 minutes.

Other work has focused on reconfiguration planning. Paulos et al. present a system in which self-reconfigurable modular boats self-assemble into prescribed floating structures, such as a bridge [72]. Individual boat modules are able to move about the pool, allowing for rapid reconfiguration. In these experiments, the environment is known and external localization is provided by an overhead AprilTag system.

Our system goes beyond existing work by using self-reconfiguration capabilities of an MSRR system to take autonomy a step further. The system uses perception of the environment to inform the choice of robot configuration, allowing the robot to adapt its abilities to surmount challenges arising from *a priori* unknown features in the environment. Through hardware demonstrations, we show that autonomous self-reconfiguration allows our system to adapt to the environment to complete complex tasks.

Part II

Environment Augmentation

Chapter 9

Environment Augmentation

In this chapter and the one that follows, we consider how robots can augment their environments to gain advantage, adding physical structures to problematic unstructured environments to establish conditions under which they can more easily move and accomplish tasks. We expand the physical capabilities of SMORES-EP by introducing building blocks that let it construct bridges and ramps. Building on our prior work in Chapter 7 addressing tasks with modular robots, we show how structure-building can be incorporated into our high-level decision-making framework in a natural way: rather than reconfiguring the body of the robot to match the properties of the environment, we reconfigure the environment (by building structures) to make its properties match the capabilities of the robot. Likewise, we leverage perception tools and environment characterization tools similar to those presented in Chapter 8 to autonomously identify build structures in response to the sensed environment.

This chapter heavily excerpts the author’s work in [94], which is again a collaboration with Gangyuan Jing, Jonathan Daudelin, Prof. Hadas Kress-Gazit, and Prof. Mark Campbell from Cornell University.

9.1 Introduction

Employing structures to accomplish tasks is a ubiquitous part of the human experience: to reach an object on a high shelf, we place a ladder near the shelf and climb it, and at a larger scale, we construct bridges across wide rivers to make them passable. The fields of collective construction robotics and modular robotics offer examples of systems that can construct and traverse structures out of robotic or passive elements [63, 74, 76, 90], and assembly planning algorithms that allow arbitrary structures to be built under a variety of conditions [83, 100]. This existing body of work provides excellent contributions regarding the generality and completeness of these methods: some algorithms are provably capable of generating assembly plans for arbitrary volumetric structures in 3D, and hardware systems have demonstrated the capability to construct a wide variety of structures.

Less work is available regarding ways that robots could deploy structures as a means of completing an extrinsic task, the way a person might use a ladder to reach a high object. This chapter presents hardware, perception, and high-level planning tools that allow structure-building to be deployed by a modular robot to address high-level tasks.

We introduce novel passive block and wedge modules that SMORES-EP can use to form

ramps and bridges in its environment. Building structures allows the robot to surmount large obstacles that would otherwise be very difficult or impossible to traverse, and therefore expands the set of tasks the robot can perform. This addresses a common weakness of modular robot systems, which often struggle with obstacles much larger than a module.

We expand on the framework for high-level tasks presented in Chapter 7. In this work, the high-level planner not only decides when to reconfigure the robot, but also when to augment the environment by assembling a passive structure. To inform these decisions, we introduce a novel environment characterization algorithm that identifies candidate features where structures can be deployed to advantage. Together, these tools comprise a novel framework to automatically identify when, where, and how the robot can augment its environment with a passive structure to gain advantage in completing a high-level task.

We integrate our tools into the system for perception-driven autonomy presented in Chapter 8, and validate them in two hardware experiments. Based on a high-level specification, a modular robot reactively identifies inaccessible regions and autonomously deploys ramps and bridges to complete locomotion and manipulation tasks in realistic office environments.

9.2 Related Work

Our work complements the well-established field of collective robotic construction, which focuses on autonomous robot systems for building activity. While we use a modular robot to create and place structures in the environment, our primary concern is not assembly planning or construction of the structure itself, but rather its appropriate placement in the environment to facilitate completion of an extrinsic high-level task.

Petersen et al. present Termes [74], a termite-inspired collective construction robot system that creates structures using blocks co-designed with a legged robot. Similarly, our augmentation modules are designed to be easily carried and traversed by SMORES-EP. Where the TERMES project focused on collective construction of a goal structure, we are less concerned with efficient building of the structure itself and more concerned with the application and placement of the structure in the larger environment as a means of facilitating a task unrelated to the structure itself.

Werfel et al. present algorithms for environmentally-adaptive construction that can build around obstacles in the environment [100]. A team of robots senses obstacles and builds around them, modifying the goal structure if needed to leave room for immovable obstacles. An algorithm to build enclosures around preexisting environment features is also presented. As with Termes, the goal is the structure itself; while the robots do respond to the environment, the structure is not built in response to an extrinsic high-level task.

Napp et al. present hardware and algorithms for building amorphous ramps in unstructured environments by depositing foam with a tracked mobile robot [63, 64]. Amorphous ramps are built in response to the environment to allow a small mobile robot to surmount large, irregularly shaped obstacles. Our work is similar in spirit, but places an emphasis on autonomy and high-level locomotion and manipulation tasks rather than construction.

Our work extends the SMORES-EP hardware system by introducing passive pieces that are manipulated and traversed by the modules. Terada and Murata [90], present a lattice-style modular system with two parts, structure modules and an assembler robot. Like many lattice-style modular systems, the assembler robot can only move on the structure modules,

and not in an unstructured environment. Other lattice-style modular robot systems create structures out of the robots themselves. M-blocks [76] form 3D structures out of robot cubes which rotate over the structure. Paulos et al. present rectangular boat robots that self-assemble into floating structures, like a bridge [72].

Magnenat et al [50] present a system in which a mobile robot manipulates specially designed cubes to build functional structures. The robot explores an unknown environment, performing 2D SLAM and visually recognizing blocks and gaps in the ground. Blocks are pushed into gaps to create bridges to previously inaccessible areas. In a “real but contrived experimental design” [50], a robot is tasked with building a three-block tower, and autonomously uses two blocks to build a bridge to a region with three blocks, retrieving them to complete its task. Where the Magnenat system is limited to manipulating blocks in a specifically designed environment, our work presents hardware, perception, and high-level planning tools that are more general, providing the ability to complete high-level tasks involving locomotion and manipulation in realistic human environments.

9.3 Approach

9.3.1 Environment Characterization

To successfully navigate its environment, a mobile robot must identify traversable areas. One simple method for wheeled robots is to select flat areas large enough for the robot to fit. However, MSRR systems can reconfigure to traverse a larger variety of terrains. The augmentation abilities we introduce extend MSRR navigation even further; the robot can build structures to traverse otherwise-impossible terrains. For autonomous operation, we need an algorithm to locate and label features in the environment that can be augmented. We present a probabilistic, template-based environment characterization algorithm that identifies augmentable features from a 2.5D elevation map of the robot’s environment.

The characterization algorithm searches for a desired feature \mathcal{F}_n using a template consisting of a grid of likelihood functions $l_i(h)$ for $1 \leq i \leq M$ where M is the number of grid cells in the template, and h is a height value. The size of grid cells in the template is variable and need not correspond to the resolution of the map. In addition, features of different size can be searched for by changing the cell size of the template to change the scale. Figure 53 shows an example of a template used to characterize a “ledge” feature, consisting of Gaussian and logistic likelihood functions. Any closed-form likelihood function may be used for each grid cell, enabling templates to accommodate noisy data and variability in possible geometric shapes of the same feature. To determine if the feature exists at a candidate pose \mathcal{X} in the map, a grid of height values is taken from the map corresponding to the template grid centered and oriented at the candidate pose, as illustrated in Figure 53. Then, the probability that each grid cell c_i belongs to the feature is evaluated using the cell’s likelihood function from the template.

$$P(c_i \in \mathcal{F}_n) = l_i(h_i) \quad (9.1)$$

The likelihood of the feature existing at that location is calculated by finding the total probability that all grid cells belong to the feature. Making the approximate simplifying assumption that grid cells are independent, this probability is equivalent to taking the product over the feature likelihoods of all grid cells in the template:

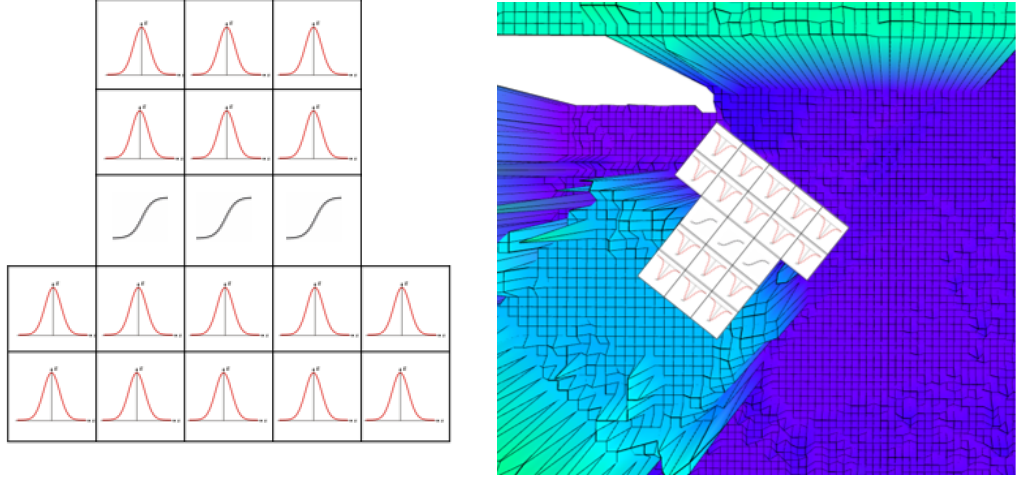


Figure 53: *Left*: Example template used to characterize a “ledge” feature. *Right*: Example template overlaid on elevation map (top view) to evaluate candidate feature pose.

$$P(\mathcal{X} \in \mathcal{F}_n) = \prod_{i=1:M} l_i(h_i) \quad (9.2)$$

The feature is determined to exist if the total probability is higher than a user-defined threshold, or $P(\mathcal{X} \in \mathcal{F}_n) > \alpha^M$, where α represents the minimum average probability of each grid cell forming part of the feature. In our experiments we use $\alpha = 0.95$. This formulation normalizes the threshold with respect to the number of grid cells in the template.

To characterize an environment, the algorithm takes as inputs an elevation map of the environment and a list of feature templates. Before searching for features, the algorithm preprocesses the elevation map by segmenting it into flat, unobstructed regions that are traversable without augmentation. It then grids the map and exhaustively evaluates each candidate feature pose from the grid, using a grid of orientations for each 2D location. In addition to evaluation with the template, candidate poses are only valid if the ends of the feature connect two traversable regions from the preprocessing step, thereby having potential to extend the robot’s reachable space. Once the search is complete, the algorithm returns a list of features found in the map, including their locations, orientations, and the two regions they link in the environment. Figure 54 shows an example of a characterized map. Each long red cell represents a detected “ledge” feature, with a corresponding small pink cell demonstrating the orientation of the feature (and the bottom of the ledge). Note that, in this example, several features are chosen close to each other. Since all connect the same regions, any one is valid and equivalent to be selected for augmentation.

The algorithm scales linearly with the number of grid cells in the 2D environment map, and linearly with the number of features being searched for. Characterization of the environment shown in Figure 54 took approximately 3 seconds to run on a laptop with an Intel Core i7 processor.

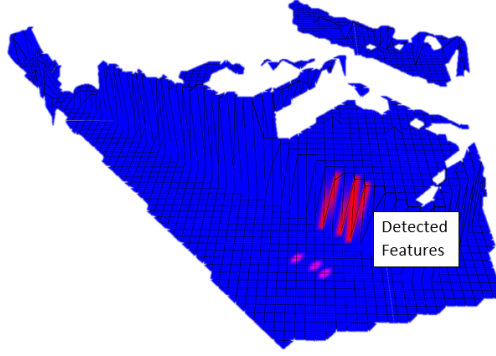


Figure 54: Characterization of an environment with a “ledge” feature. Red indicates a detected feature, pink indicates the start of the feature, demonstrating orientation.

9.3.2 Hardware: Augmentation Modules

Our system is built around the SMORES-EP modular robot. Large obstacles, like tall ledges or wide gaps in the ground, are often problematic for modular robot systems. One might expect that a modular system could scale, addressing a large-length-scale task by using many modules to form a large robot. In reality, modular robots don’t scale easily: adding more modules makes the robot bigger, but not stronger. The torque required to lift a long chain of modules grows quadratically with the number of modules, quickly overloading the maximum torque of the first module in the chain. Consequently, large systems become cumbersome, unable to move their own bodies. Simulated work in reconfiguration and motion planning has demonstrated algorithms that handle hundreds of modules, but in practice, fixed actuator strength has typically limited these robots to configurations with fewer than 40 modules.

We address this issue by extending the SMORES-EP hardware system with passive elements called *environment augmentation modules*. We use the *wedge* and *block* augmentation modules shown in Figure 55. Like the construction blocks used by Termes [74], wedge and block modules are designed to work synergistically with SMORES-EP, providing features that use the best modes of locomotion (driving), manipulation (magnetic attachment), and sensing (AprilTags) available to SMORES-EP.

Blocks are the same size as a module (80mm cube), and wedges are half the size of a block (an equilateral right triangle with two 80mm sides). Both are made of lightweight laser-cut medium-density fiberboard (blocks are 162g, wedges are 142g) and equipped with a steel attachment point for magnetic grasping. Neodymium magnets on the back faces of wedges, and the front and back faces of blocks, form a strong connection in the horizontal direction. Interlocking features on the top and bottom faces of the blocks, and the bottom faces of the wedges, allow them to be stacked vertically. Wedges provide a 45-degree incline with a high-friction rubber surface, allowing a configuration of 3 or more modules to drive up them. Side walls on both the wedges and blocks ensure that SMORES-EP modules stay aligned to the structure and cannot fall off while driving over it. The side walls of wedges are tapered to provide a funneling effect as modules drive onto them, making them more tolerant to misalignment. Each wedge and ramp has unique AprilTag fiducials on its

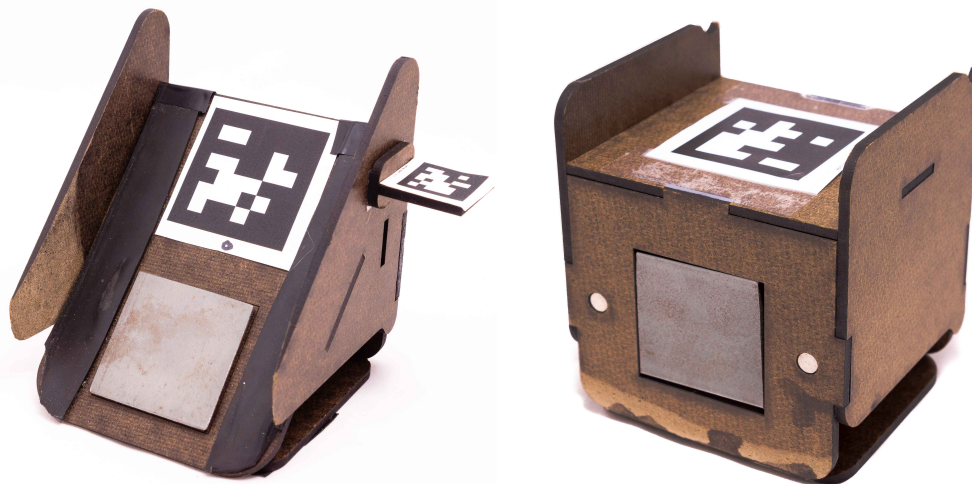


Figure 55: Wedge and Block Augmentation Modules

faces, allowing easy identification and localization during construction and placement in the environment.

Wedges and blocks allow a SMORES-EP cluster to autonomously construct bridges or ramps that allow it to reach higher heights and cross wider gaps than it could with robot modules alone (Figure 56). Provided enough time, space, and augmentation modules are available, there is no limit to the height of a ramp that can be built. Bridges have a maximum length of 480mm (longer bridges cannot support a load of three SMORES-EP modules in the center).

9.3.3 High-Level Planner

For task-related decision-making, we extend the high-level planning framework introduced in Chapter 7. As before, users do not explicitly specify configurations and behaviors for each task, but rather define goals and constraints for the robot. The high-level planner chooses robot configurations and behaviors from the design library based on the task specifications, and executes them to satisfy the tasks.

Consider the example controller shown in Figure 57. The robot tasks are to look for a pink drawer, open the drawer, and then climb on top of it. Each state is labeled with a desired robot action, and each transition is labeled with perceived environment information; for example, the “climb drawer” action is specified to be any behavior from the library with properties *climb* in a *ledge* environment. In our previous framework, the high-level planner could choose to reconfigure the robot whenever needed to satisfy the required properties of the current action and environment.

In this work, the high-level planner can choose not only to change the abilities of the robot (reconfiguration), but also the properties of the environment (environment augmentation). We expand the library of robot designs with feature templates for recognizing augmentable environments (Section 9.3.1). Associated with each template is a controller for manipulating augmentation modules to build the desired structure. Assembly plans are currently created

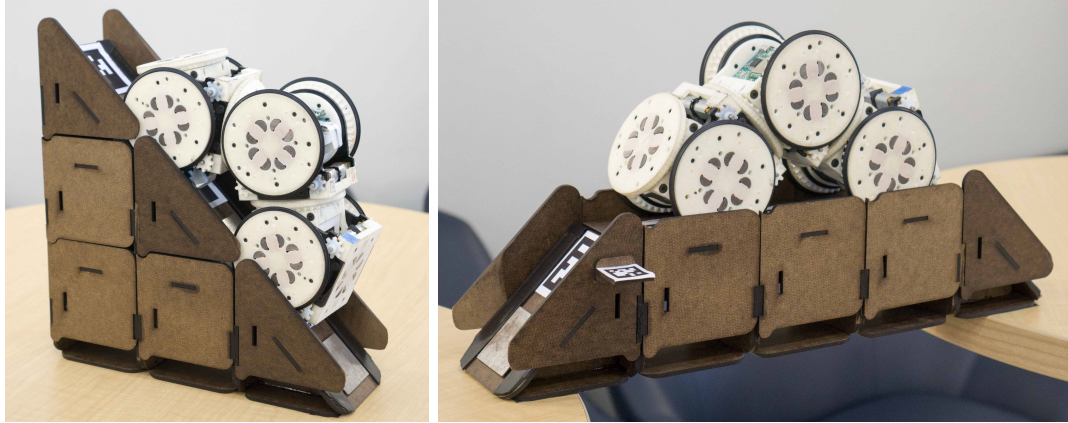


Figure 56: Bridge and Ramp

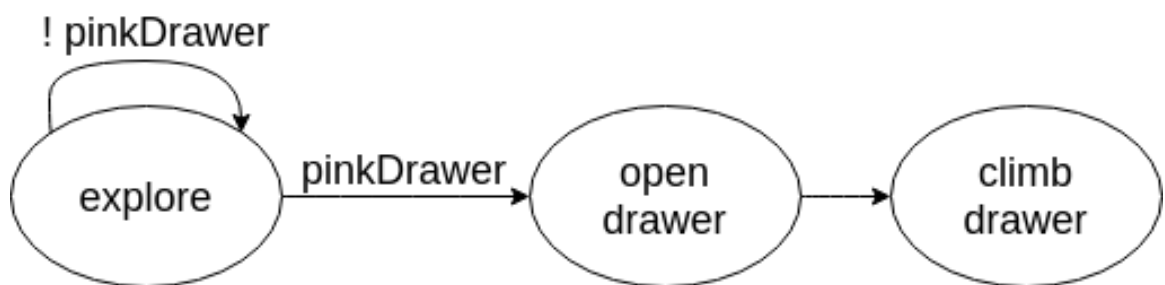


Figure 57: An example of synthesized robot controller

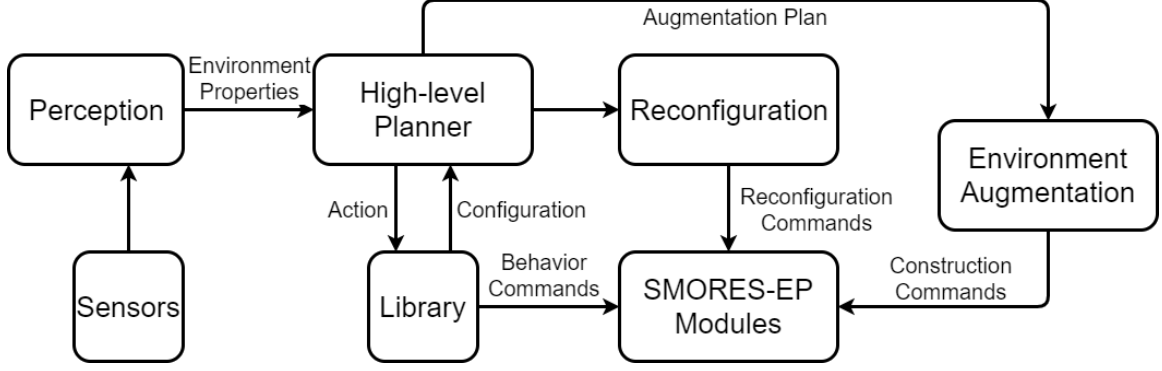


Figure 58: System Overview Flowchart

manually, but could be automatically generated [83, 100].

For the example in Figure 57, if no behavior in the library satisfies the “climb drawer” action, the high-level planner will query the environment characterization subsystem (Section 9.3.1) for possible environment augmentations. The high-level planner passes a set of feature templates to the environment characterization subsystem. Once the environment characterization subsystem recognizes the augmentable environment, it provides the high-level planner a list of matched environment features, and two lists of regions $R^1 = \{r_0^1, r_1^1, \dots\}$, $R^2 = \{r_0^2, r_1^2, \dots\}$ that the features connect.

The high-level planner uses the features to find a list of augmentation plans $A = \{a_0, a_1, \dots\}$ from the library. An augmentation plan a_i consists of i) a set augmentation modules used for structure construction, ii) the position and orientation of the structure, and iii) the construction controller for the robot. To choose the best augmentation plan from the list A , the high-level planner considers the available augmentation modules in the current environment, current robot configuration, and distance from the structure to the robot goal position. The high-level planner then commands the robot to construct the structure based on the best augmentation plan. After the robot constructs the structure for augmentation a_i , the high-level planner considers regions r_i^1 and r_i^2 to be connected and traversable by the robot.

9.4 System Integration

We integrate our environment augmentation tools into the system for autonomy introduced in Chapter 8, as shown in Figure 58. The high-level planner automatically converts user defined task specifications to controllers from a robot design library. It executes the controller by reacting to the sensed environment, running appropriate behaviors from the design library to control a set of hardware robot modules. Active perception components perform simultaneous localization and mapping (SLAM), and characterize the environment in terms of robot capabilities. Whenever required, the reconfiguration subsystem controls the robot to change configurations.

The system used the Robot Operating System (ROS)¹ for a software framework, networking, and navigation. SLAM was performed using RTAB-MAP[48], and color detection

¹<http://www.ros.org>

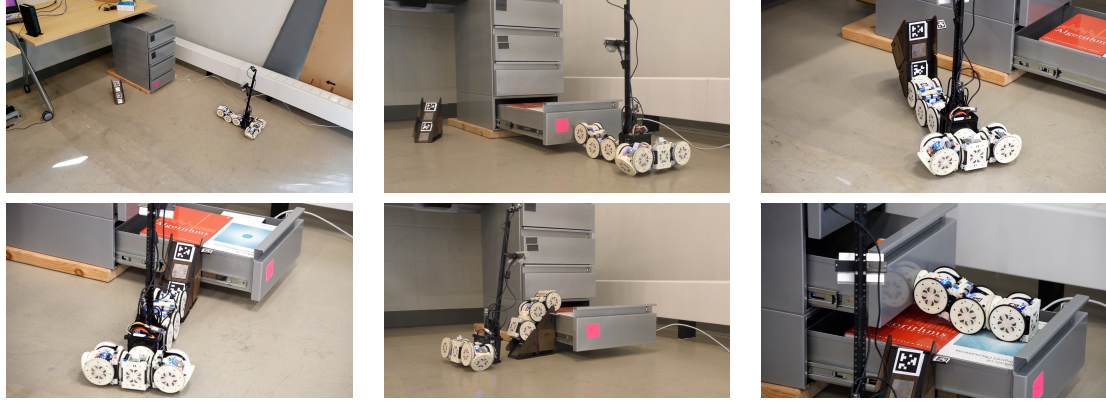


Figure 59: Snapshots throughout Experiment I. From left to right, top to bottom: i) Experiment start ii) Opening first drawer iii) Picking up ramp iv) Placing ramp next to open drawer. v) Reconfiguring and climbing ramp vi) Opening second drawer

was done using CMVision².

As SMORES-EP modules have no sensors that allow them to gather information about their environment, we again utilize the sensor module introduced in Chapter 8 to allow autonomous operation (See Figure 47).

9.5 Experiment Results

Our system can generalize to arbitrary environment augmentations and high-level tasks. We validate our system in two hardware experiments that require the same system to perform tasks requiring very different environment augmentations for successful completion. In both experiments, the robot autonomously perceives and characterizes each environment, and synthesizes reactive controllers to accomplish the task based on the environment. Videos of the full experiments are available online at <https://youtu.be/NKj-xulsxco>.

9.5.1 Experiment I

For the first experiment, the robot is tasked with inspecting two metal desk drawers in an office. If it cannot open both drawers, it should indicate recognition of the situation by self-disassembling (breaking apart into pieces). Although the robot can pull the bottom drawer by attaching to it with its magnetic connectors, it is unable to reach and open the second drawer from the ground. Thus, it can only open the second drawer if it can first open the bottom drawer and then climb on top of the things inside it.

Figure 59 shows snapshots throughout the robot’s autonomous performance of Experiment I. After recognizing and opening the first drawer, the robot characterizes the environment with the opened drawer and identifies the side of the drawer as a “ledge” feature. The high-level planner recognizes that the ledge is too high for the current configuration to climb, and furthermore that there is no other configuration in the library to which the robot can transform that could climb the ledge, leaving environment augmentation as the only strategy that can complete the task. Observing a ramp structure in the environment, the

²CMVision: <http://www.cs.cmu.edu/~jbruce/cmvision/>

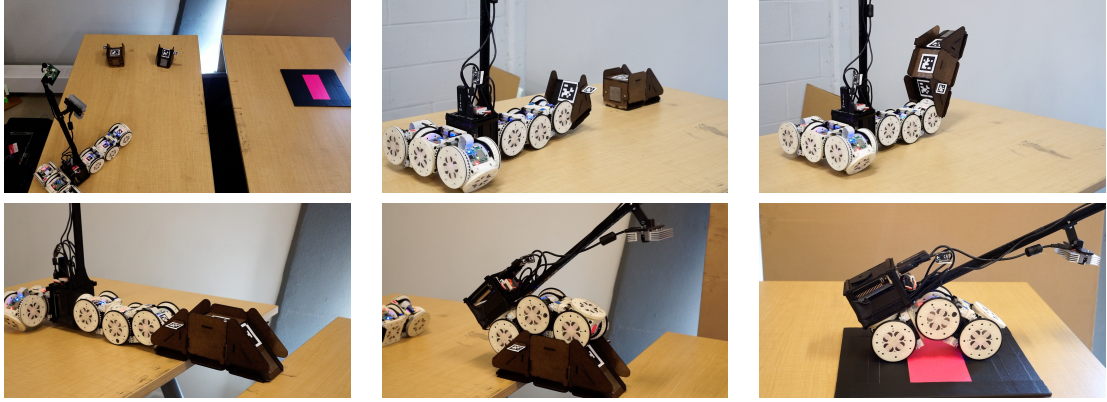


Figure 60: Snapshots throughout Experiment II. From left to right, top to bottom: i) Experiment start ii) Assembling bridge iii) Transporting bridge iv) Placing bridge over gap. v) Reconfigure and cross bridge. vi) Arrive at the target zone.

high-level planner commands the robot to acquire the ramp, place it at the “ledge” feature detected by the characterization algorithm, climb the drawer, and complete the mission.

In a second version of the same experiment, the first drawer is empty. When the robot characterizes the environment containing the drawer, it identifies no “ledge” features, since the drawer no longer matches the requirements of the feature. As a result, it recognizes that environment augmentation is not possible, and the mission cannot be completed. To indicate this, it self-disassembles.

9.5.2 Experiment II

The environment for Experiment II consists of two tables separated by a 16 cm gap. The robot begins the experiment on the left table with two wedges and one block. To complete its mission, the robot must cross the gap to reach a pink destination zone on the right table.

Figure 60 shows snapshots throughout Experiment II. This time, characterization of the environment identifies that the pink goal zone is in a separate region from the robot, and also identifies several “gap” features separating the two regions. Recognizing that the gap is too wide for any configuration in the design library to cross unassisted, the high-level planner concludes it must build a bridge across the gap to complete its mission. It begins searching for materials, and quickly identifies the three available augmentation modules, which it autonomously assembles into a bridge. It then places the bridge across the gap and crosses to complete its mission.

9.6 Discussion

Block and wedge modules demonstrably expand the physical capabilities of SMORES-EP, allowing the system to climb to a high ledge and cross a wide gap to complete tasks that would have been very difficult with the SMORES-EP modules alone. Perception tools accurately characterize augmentable features in the environment. High-level reasoning tools identify when environment augmentation is necessary to complete a high-level task, and reactively sequence locomotion, manipulation, construction, and reconfiguration actions to accomplish the mission. The presented work represents the first time that modular robots

Outcome	Exp 1	Exp 2
Success	2 (25.0%)	3 (37.5%)
Perception-Related Failure	2 (25.0%)	2 (25.0%)
Navigation Failure	1 (12.5%)	0 (0.0%)
Hardware Failure	3 (37.5%)	2 (25.0%)
Setup Error	0 (0.0%)	1 (12.5%)

Table 10: Outcomes for Experiments 1 and 2

have successfully augmented their environment by deploying passive structures to perform high-level tasks.

As with our previous work with autonomous modular robots in Chapter 8, robustness proved challenging. Out of 8 test runs of Experiment I, the robot successfully completed the entire task once. Table 10 shows the outcomes of 8 runs of each experiment. The largest source of error was due to hardware failures such as slight encoder miscalibration or wireless communication failure. Creating more robust hardware for modular robots is challenging due to the constrained size of each module, and the higher probability of failure from higher numbers of components in the system.

Perception-related errors were another frequent cause of failure. These were due in part to mis-detections by the characterization algorithm, or because the accuracy in finding location and orientation of features was not high enough for the margin of error of the robot when placing structures. Finally, navigation failures occurred throughout development and experiments due to cumulative SLAM errors as the robot navigates the environment. We found that it was important to minimize in-place rotation of the robot, and to avoid areas without many features for visual odometry to use.

9.6.1 Conclusion

To conclude, this chapter presents tools that allow a modular robot to autonomously deploy passive structures as a means to complete high-level tasks involving locomotion, manipulation, and reconfiguration. This work expands the physical capabilities of the SMORES-EP modular robot, and extends our existing frameworks for addressing high-level tasks with modular robots by allowing both the robot morphology and the environment to be altered if doing so allows the task to be completed. We validate our system in two hardware experiments that demonstrate how the hardware, perception tools, and high-level planner work together to complete high-level tasks through environment augmentation.

Chapter 10

Optimal Structure Synthesis

The previous chapter demonstrates how SMORES-EP can augment its environment to accomplish high-level tasks. The system relied on a human-made library with a limited number of structures: deploying a “**height-2 ramp**” structure required a library entry devoted to that particular structure, along with a “**height-2 ledge**” classifier to identify where it could be used.

A more general approach would attempt to synthesize traversable structures directly from model of the robot’s movement capabilities and a representation of the environment. Consider a scenario where a small robot (like SMORES-EP) must move through an environment filled with objects much larger than itself. The robot is unable to cross over the objects or traverse large gaps. As a result, entire portions of the environment may be inaccessible. This task presents a planning problem: Given an environment, a robot, and a supply of building blocks, can we find a set of structures that could be added to the environment to make it fully accessible to the robot (i.e. there exists a navigable path between any pair of points)? Furthermore, since structure-building is a time-consuming process, can we find such a set of structures which uses a minimum number of building blocks? We refer to this as the *optimal structure synthesis* problem.

This chapter presents a mathematical formalism for optimal synthesis of structures made of discrete building blocks and shows that the problem is NP-Hard. We present a complete, optimal algorithm that will find a minimum-cost set of structures to make any input environment traversable, if such a set exists. The algorithm solves practical problems efficiently using a branch-and-bound strategy, typically exploring a tiny fraction of the exponentially-large solution space before finding the optimal solution. In our experiments, we show that the algorithm finds optimal solutions in about one minute for 3D maps of real indoor environments, and demonstrate that the structures selected by the algorithm do indeed allow the robot to traverse the environments.

This chapter excerpts heavily from the author’s work in [96]. Credit is due to co-authors Prof. Cynthia Sung (Penn) and Colin McCloskey (Yale), who contributed significantly to this work.

10.1 Related Work

The fields of collective construction robotics and modular robotics offer examples of systems that build and traverse structures. Petersen et al. present Termes [74], a termite-inspired

collective construction robot system that creates structures using blocks co-designed with a legged robot. Werfel et al. present algorithms for environmentally-adaptive construction: a team of robots senses obstacles and builds around them, modifying the goal structure if needed to leave room for immovable obstacles [100]. Terada and Murata [90] present a lattice-style modular system with two parts, structure modules and an assembler robot. Like many lattice-style modular systems, the assembler robot can only move on the structure modules, and not in an unstructured environment. M-blocks [76] form structures out of robot cubes which rotate over the structure, and can reconfigure between arbitrary 3D shapes, except those containing certain inadmissible sub-configurations [88]. Other related work in manipulation planning allows robots to carry out multi-step procedures to assemble furniture [45] or rearrange clutter surrounding a primary manipulation task [19].

There is also some work showing that robots can deploy structures to enhance their ability to move through an environment. Napp et al present a distributed algorithm for adaptive ramp building with amorphous materials [63]. Using local information, the algorithm controls one or more robots to deposit amorphous material (like foam or sandbags) on their environment to make a goal point accessible [79]. Our work addresses a similar, complementary problem: assuming a known map of the entire environment, we generate a *globally optimal* set of structures making *every* point accessible.

10.2 Problem Formulation

10.2.1 Preliminaries

We consider the problem of a ground robot traversing an environment \mathcal{E} , which is a discretized height map $\mathcal{E} : \mathbb{Z}^2 \rightarrow \mathbb{R}^+$. The robot moves over the surface of \mathcal{E} , and its goal is to be able to access every location on the map. The environment can be represented as a grid graph $G(V, E)$, where V is a set of nodes corresponding to each of the grid cells, and edges $E \subseteq V \times V$ connect neighboring cells. The robot’s movement are treated as discrete transitions along edges.

The robot is subject to certain physical limitations, which determine its ability to actually traverse this graph. We model the robot’s ability to move using *traversability* criteria, which we define for both nodes and edges.

The first criterion is on edges, or transitions between nodes. A ground robot is typically not able to traverse a sharp rise or dropoff (or “*cliff*”). We therefore identify single edges across which the difference in height is above a threshold ΔZ_{cliff} to be non-traversable.

Definition 12 (Non-Traversable Edge). An edge $e = (v_i, v_j)$ is non-traversable if $|\mathcal{E}(v_i) - \mathcal{E}(v_j)| > \Delta Z_{cliff}$. We refer to these edges as cliffs.

The second criterion is on the slope of the environment. In some cases, a ground robot will not be able to traverse ground that is too steep. We find these “*steep*” areas by identifying nodes at which the average gradient (within a window of radius d about the node) is higher than a threshold K_{steep} . To prevent cliffs from influencing the apparent steepness of nearby nodes (due to the windowed average), we remove the contribution of cliffs from the gradient

before identifying steep nodes. Let:

$$\text{cliffs}(\mathcal{E}(v)) = \begin{cases} |\nabla \mathcal{E}(\vec{v})| & \text{if } |\nabla \mathcal{E}(v)| > \Delta Z_{\text{cliff}} \\ 0 & \text{otherwise} \end{cases} \quad (10.1)$$

$$\text{steep}(\mathcal{E}(v)) = \text{ma}_d(|\nabla \mathcal{E}(v)| - \text{cliffs}(\mathcal{E}(v))) \quad (10.2)$$

where ma_d denotes a windowed-moving-average with window-size d . In our implementation, $d = 8$ cm.

Definition 13 (Non-Traversable Node). A node $v \in V$ is non-traversable if $\text{steep}(\mathcal{E}(v)) > K_{\text{steep}}$.

For SMORES-EP, $\Delta Z_{\text{cliff}} = 4$ cm and $K_{\text{steep}} = 1$.

We define the *traversable environment* to be G_T , the subgraph of G containing only traversable nodes and edges. If G_T is connected, then the robot is able to reach every traversable grid cell. However, it is possible that G_T is not connected. In this case, the environment is split into *regions*, which are the connected components of G_T .

Definition 14 (Region). A region $r_i \in R$ is a connected component in $G_T(V, E)$. By definition, a path exists between any two nodes in a single region, and no path exists between any two nodes in different regions.

Figure 61 shows an example. On the left is an environment with a table and chair, and on the right is the corresponding height map \mathcal{E} . Different colors on this map correspond to different regions.

10.2.2 Structures

We seek to add structures to the environment which will allow the robot to move between regions. Structures are composed of building blocks, which the robot is able to carry, place, and drive over. In this work, the goal is to select the shapes, positions, and orientations of structures that could be added to the environment to make it globally traversable. We do not attempt to determine what actions the robot could take to actually build the structure - to do so, we could use an existing algorithm for robotic assembly planning (See Section 10.1).

A *structure* is a line of columns of stacked building blocks. We consider structures made of building blocks similar to the *block* and *wedge* shown in Figure 55. In general we assume building blocks have a square footprint with side length L_B ; for our building blocks, $L_B = 8\text{cm}$. Importantly, we assume the robot can only move over the structure along a straight line - this is a physical constraint imposed by the building blocks, which have side walls. We identify building blocks with type labels $t \in \{\text{block}, \text{wedge}_f, \text{wedge}_b\}$, with “f” and “b” denoting the two possible orientations (forward and backward) of wedges with respect to the structure. For each type, we define a surface function $s : [0, L_B] \rightarrow \mathbb{R}^+$ describing its shape: $\text{block} : s(x) = L_B$, $\text{wedge}_f : s(x) = x$, and $\text{wedge}_b : s(x) = L_B - x$.

Viewed from above, the footprint of a structure in the xy plane is a linear array of n contiguous squares which we call *structure cells* (Figure 61); sitting atop each structure cell is a column of stacked blocks which comprise the structure. Structures are not locked to the grid - rather, the build point \mathbf{b} defines the position of the first structure cell in the plane,

and the orientation vector $\hat{\mathbf{u}}$ defines the line along which all cells lie, and the direction along which the robot may move.

Definition 15 (Structure). A structure $T = \langle \mathbf{b}, \hat{\mathbf{u}}, C \rangle$ has build point $\mathbf{b} \in \mathbb{R}^2$, orientation vector $\hat{\mathbf{u}} \in \mathbb{R}^2 : |\hat{\mathbf{u}}| = 1$, and cells $C = \{c_1 \dots c_n\}$. Each cell $c_i = \langle \lambda_i, h_i, t_i \rangle$, $i \in \{1 \dots n\}$ has corners $\lambda_i \in \mathbb{R}^{4 \times 2}$, column height $h_i \in \mathbb{Z}^+$, and surface block type t_i .

The *height* and *surface block type* of a structure cell respectively specify the number of blocks stacked on that cell and the type of building block at the top of the stack, which determines the shape of the structure surface over which the robot will drive. To fully define a structure, it is sufficient to provide \mathbf{b} , $\hat{\mathbf{u}}$, and lists of cell heights $\{h_1, h_2 \dots, h_n\}$ and surface block types $\{t_1, t_2 \dots, t_n\}$; assuming L_B is known, cell corners λ can be computed from \mathbf{b} and $\hat{\mathbf{u}}$. The *cost* of a structure is the number of blocks it contains, $\text{cost}(T) = \sum_{i=1 \dots n} h_i$.

For a structure to be considered valid, the environment surface between each structure cell must be suitable to support the column of blocks that sit on top of it in a stable way. For our structures, we define *buildability* in terms of the flatness of the underlying environment surface. Letting $\mathcal{E}(c_i) = \text{median}(\mathcal{E}(v)) \forall v \in c_i$, cell c_i is buildable iff $\mathcal{E}(v) - \mathcal{E}(c_i) < \alpha L_B \forall v \in c_i$. In our implementation, $\alpha = 0.4$.

For a structure to be useful, it must be both buildable and traversable. Since each block is individually traversable, we determine traversability of a structure by evaluating the cliff condition at the boundaries between neighboring cells c_i and c_{i+1} :

$$Z_i = \mathcal{E}(c_i) + h_i * L_B + s_i(L_B) \quad (10.3)$$

$$Z_{i+1} = \mathcal{E}(c_{i+1}) + h_{i+1} * L_B + s_{i+1}(0) \quad (10.4)$$

$$|Z_{i+1} - Z_i| < \Delta Z_{cliff} \quad (10.5)$$

The boundary is traversable if Equation 10.5 is satisfied.

In addition to moving between blocks on the structure, the robot must also be able to transition from the structure to the surrounding ground surface at both ends. To capture these transitions, we require the first and last cell of traversable structures to have zero height (i.e. to simply be the environment surface), and exempt them from the buildability condition. Additionally, when moving to the first or last structure cell involves crossing a region boundary (e.g. moving from the structure to the top of a cliff), the region boundary must be flat, and its surface normal must align with the structure orientation $\hat{\mathbf{u}}$, so that structure presses flat up against the cliff (Figure 61).

Definition 16 (Valid Structure). A structure T is valid if: $\forall c_i, c_{i+1} \in C$, (c_i, c_{i+1}) is traversable, and $\forall c_i \in \{c_2 \dots c_{n-1}\}$, c_i is buildable.

10.2.3 Conflicts

Introducing a valid structure allows the robot to move between its the endpoint cells, so if those cells are in different regions, the structure creates a path them. Our goal, then, is to synthesize a set of structures $\mathcal{T} = \{T_1, T_2 \dots, T_k\}$ on a world graph with regions R such that there is a path between every pair of regions. Of course, because structures occupy physical space in the environment, even if a structure is valid in isolation, it may generate conflicts when combined with others. There are two kinds of *conflicts* that can

exist between structures and regions. A pair of structures conflict if they collide. A set of structures conflicts with a region if they divide the region into disconnected pieces.

Definition 17 (Structure-Structure Conflict). A pair of structures conflict if any of their cells intersect.

Definition 18 (Region-Structure Conflict). Consider region r_c and set of structures $\mathcal{T}_c = \{T_1, T_2, \dots, T_k\}$. Let V_{r_c} be the set of nodes in region r_c . Let $V_{\mathcal{T}_c}$ be the set of nodes in the cells of \mathcal{T}_c . Let G_{r_c} be the subgraph of $G(V, E)$ induced by $V_{r_c} \setminus V_{\mathcal{T}_c}$. There is a region-structure conflict between r_c and \mathcal{T}_c if G_{r_c} is disconnected.

Structure-structure conflicts result from collisions between structures and represent a situation that is physically impossible, so a set \mathcal{T} which contains a structure-structure conflict is considered invalid. Region-structure conflicts occur when a set of structures \mathcal{T}_c cut through a region r_c in such a way that it is no longer fully connected. Consequently, r_c needs to be treated as multiple regions when \mathcal{T}_c are present.

10.2.4 Problem Statement

Consider an input environment \mathcal{E} , represented as grid-graph $G(V, E)$, with regions R . Our objective is to find a min-cost, conflict-free set of valid structures which make the entire environment traversable. More precisely, let $G_R(R, \mathcal{T})$ be a graph in which nodes represent regions and edges represent valid structures connecting regions. We seek a set of structures $\mathcal{T}^* = \{T_1, T_2, \dots, T_k\}$ such that $G_R(R, \mathcal{T}^*)$ is fully connected and conflict-free, and $\text{cost}(\mathcal{T}^*) = \sum_{T_i \in \mathcal{T}^*} \text{cost}(T_i)$ is minimized.

10.2.5 Approach Summary

Our approach consists of two major steps: (1) generating the set \mathcal{T} of all potentially-useful structures, and (2) forming graph $G_R(R, \mathcal{T})$, and identifying a subset of edges $\mathcal{T}^* \subseteq \mathcal{T}$ which form a conflict-free minimum-spanning tree.

10.3 Waterfall Algorithm - Generating All Useful Structures

A valid structure may be placed at any position and orientation in the environment, but we observe that it is only *useful* if it connects two different regions, so we ought to look for structures that bridge the boundaries between regions. Regions are bounded by sloped areas (where nodes are removed) and sharp cliffs (where edges are removed), but because sloped areas violate the conditions for buildability, we consider only cliffs as candidate locations for useful, valid structures. We define the set of *build points* $B = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$ to be the coordinates of the cliff edges. At each build point, we attempt to synthesize a structure connecting the top of the cliff to another region.

For the robot to move from the cliff surface onto the structure, the structure's orientation $\hat{\mathbf{u}}$ should be nearly perpendicular to the cliff surface (Definition 16). In our implementation, the cliff surface normal at \mathbf{b} is estimated by selecting all nodes with a radius of L_B of \mathbf{b} and training a linear classifier (specifically an SVM) using node coordinates as features and node regions as labels; the resulting classification plane provides a suitable estimate of the surface normal. The error rate of the classifier provides a measure of surface flatness: if more than 10% of training points are incorrectly classified, the boundary is considered non-flat and the build point is rejected.

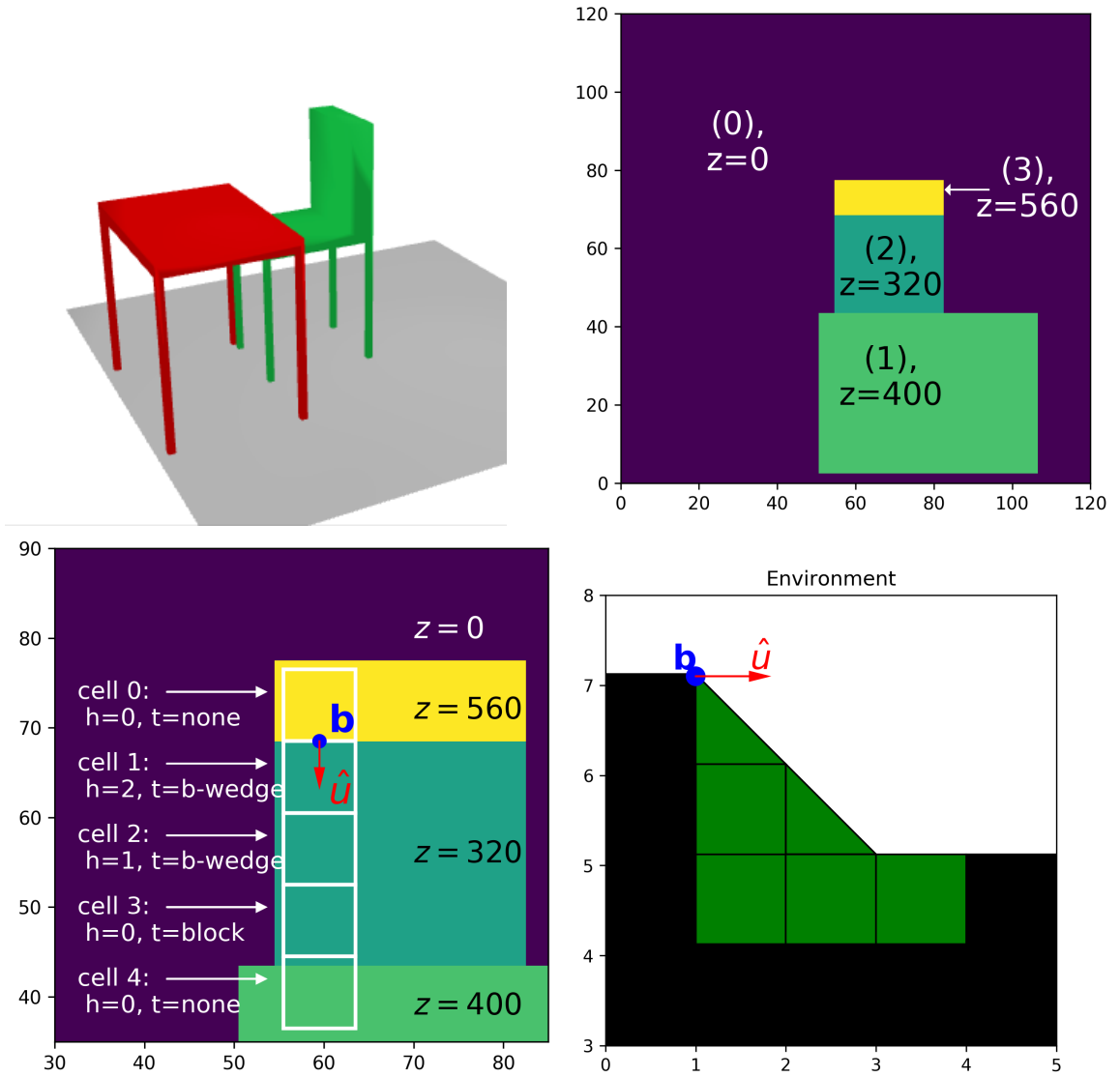


Figure 61: Example environment and structure. Top: Example table-and-chair environment (left), and corresponding height field world (right), with labeled regions. Bottom: Example structure X-Y view (left) and \hat{u} -Z view (right).

10.3.1 Algorithm

Given build point \mathbf{b} and orientation $\hat{\mathbf{u}}$, we synthesize valid structures by solving the constraint satisfaction problem imposed by the conditions for structure validity (Definition 16). Using parameters $L_B = 8cm$ and $\Delta Z_{cliff} = 4cm = L_B/2$ for our robot and building blocks, the constraint imposed by Equations 10.3-10.5 imposes upper and lower bounds on the cell heights h_i (measured in number of blocks) as a function of the bounds on its neighbors:

$$h_{max}(i)_j = \left\lceil h_{max}(j) + \frac{\mathcal{E}(c_j) - \mathcal{E}(c_i)}{L_B} + 1.5 \right\rceil \quad (10.6)$$

$$h_{min}(i)_j = \left\lfloor h_{min}(j) + \frac{\mathcal{E}(c_j) - \mathcal{E}(c_i)}{L_B} - 1.5 \right\rfloor \quad (10.7)$$

The brackets $\lceil x \rceil$ and $\lfloor x \rfloor$ denote the ceiling and floor functions, which round x up or down to the nearest integer (respectively). The above equations enforce the constraints imposed on c_i by c_j , where $j = i \pm 1$.

The WATERFALL algorithm (Algorithm 1) solves for structures by propagating these bounds outwards from \mathbf{b} . It begins by placing the first structure cell at the top of the cliff, assigning it height $h = 0$. It then marches cells into the lower region, outwards from \mathbf{b} in the direction of $\hat{\mathbf{u}}$ (Fig. 61). For each new cell, it calculates h_{min} and h_{max} , propagating the constraints forward. If it encounters a cell that is unbuildable, or for which $h_{max} < 0$, it returns False (no structure can be built here). If it finds a cell for which $h_{min} < 0$, it records this cell as the endpoint of the structure, and assigns it height $h = 0$.

The algorithm then marches back to \mathbf{b} , assigning cell heights. At each cell c_i , it checks the h_{min} constraint in the backwards direction (imposed on i by $i + 1$), and assigns h_i to be the maximum of the lower-bounds from enforcing the constraint in both directions.

The algorithm performs a final pass to assign terminators. At each cell i , it considers the difference in height Z between its neighbors $i + 1$ and $i - 1$. If magnitude of the difference is less than $L_B/2$, the terminator is a block (flat); otherwise, the terminator is a wedge oriented so that its slope matches the sign of ΔZ .

10.3.2 Proofs and Analysis

Optimality

WATERFALL finds the cheapest valid structure (if it exists) that can be built at \mathbf{b} which will connect the region at the top of the cliff to another region. Valid structures must begin and end with zero-height cells (i.e. at the environment surface). In its first pass, WATERFALL finds the closest cell to \mathbf{b} at which a valid structure could terminate, and selects this cell as the end of the structure. At each cell between the terminators, WATERFALL computes the lower bound on structure height imposed by propagating the traversability constraints forward from the start cell and backwards from the end cell, and selects the minimum structure height satisfying both constraints.

Completeness

For the purposes of minimal structure construction, we need not consider any structures that could be built at \mathbf{b} other than the one produced by WATERFALL.

To show this, consider if it were not the case. Say a longer structure T_L (connecting \mathbf{b}

to a different region) is required as part of the minimum spanning tree of structures. This structure would cut through multiple regions. Based on the buildability conditions, at each point on a region boundary that this structure crosses through there is another potential build point, with its own potential minimal structure identified by the WATERFALL algorithm in the $\hat{\mathbf{u}}$ direction. All of these minimal structures must cost less than T_L , and their footprints are fully contained within the footprint of T_L . Call the set of these structures \mathcal{T}_L .

Now consider if T_L were removed from the minimum spanning tree. The environment would then be separated into two separate components. At least one of the minimal structures in \mathcal{T}_L must bridge these components. If this structure is added back to the minimum spanning tree, then the environment is again fully traversable, and the new set of structures costs less than the original. So the original spanning tree of structures was not minimal. We therefore only need to search through the set of structures discovered through WATERFALL.

Runtime and Generalization

In general, the structure synthesis problem can be formulated as an integer linear program (ILP) – the variables (cell heights) are integer valued, while the constraints (traversability) and objective (number of blocks used) are linear. Well-established algorithms can solve general ILP problems, however ILP is known to be NP-Complete. For the particular case of our building blocks, we are able to solve this ILP efficiently using the WATERFALL algorithm, which synthesizes structures with N cells in $O(N)$ time. For other kinds of building blocks, the WATERFALL algorithm could be modified, or a general ILP solver could be used.

10.4 BB-MST Algorithm – Solving for the Minimum Spanning Tree of Structures

Given the set of structures \mathcal{T} generated by running WATERFALL at every build point, form graph $G_R(R, \mathcal{T})$ with a nodes representing regions and edges representing structures connecting regions, and assign edges weights equal to the cost of their structures. We seek $M_{R, \mathcal{T}}^*$, a conflict-free min-cost spanning tree of $G_R(r, \mathcal{T})$. When we say that $M_{R, \mathcal{T}}^*$ is conflict free, we mean that (1) it may contain no structure-structure conflicts, and (2) if it contains any region-structure conflicts, $M_{R, \mathcal{T}}^*$ must span the split regions. We refer to this problem as *struct-MST*.

10.4.1 NP-Hardness of struct-MST

Kruskal’s algorithm computes minimum-spanning trees of graphs in $O(|E| \log |V|)$ time. However, structure-structure and region-structure conflicts (Section 10.2.3) impose constraints that make struct-MST a much more difficult problem to solve efficiently. Structure-structure conflicts create *pairwise negative disjunctive constraints* between edges in G_R , that is, pairs of edges that cannot both be present in the solution. These constraints may be represented in terms of a *conflict graph* with vertices corresponding to edges in the original graph, and edges corresponding to constraints. It has been shown that deciding the existence of a spanning tree (as well as finding the min-cost spanning tree) of a graph is strongly \mathcal{NP} -hard under negative disjunctive constraints, unless the conflict graph has a maximum path length less than two [14]. In general this is not the case for struct-MST (because any structure could conflict with multiple others), making struct-MST at least \mathcal{NP} -hard.

Algorithm 1 Waterfall Algorithm

Require: build point \mathbf{b} and direction $\hat{\mathbf{u}}$

hn , hx , h , and t are lists that resize automatically.

```
1: function WATERFALL( $\mathbf{b}, \hat{\mathbf{u}}$ )
2:    $end \leftarrow \infty$  ,  $hx[0] \leftarrow 0$  ,  $hn[0] \leftarrow 0$ 
3:   for  $i \leftarrow 1$ ;  $i < \infty$ ;  $i \leftarrow i + 1$  do                                ▷ Pass 1: Endpoint
4:     if  $\neg buildable(i)$  then return False
5:     end if
6:      $hn[i] \leftarrow h_{min}(i)_{i-1}$ 
7:      $hx[i] \leftarrow h_{max}(i)_{i-1}$ 
8:     if  $hx[i] < 0$  then return False
9:     end if
10:    if  $hn[i] < 0$  then
11:       $end \leftarrow i$  ,  $hx[i] \leftarrow 0$  ,  $hn[i] \leftarrow 0$ 
12:      break
13:    end if
14:  end for
15:   $h[0] \leftarrow 0$  ,  $h[end] \leftarrow 0$                                           ▷ Pass 2: Heights
16:  for  $i \leftarrow end - 1$ ;  $i > 0$ ;  $i \leftarrow i - 1$  do
17:     $hn[i] \leftarrow \max(h_{min}(i)_{i+1}, hn[i])$ 
18:     $h[i] \leftarrow hn[i]$ 
19:  end for
20:   $t[0] \leftarrow none$  ;  $t[end] \leftarrow none$                                     ▷ Pass 3: Terminators
21:  for  $i \leftarrow 1$ ;  $i < end$ ;  $i \leftarrow i + 1$  do
22:     $\Delta Z \leftarrow \mathcal{E}(c_{i-1}) - \mathcal{E}(c_{i+1}) + L_B(h[i - 1]) - h[i + 1])$ 
23:    if  $|\Delta Z| \leq 0.5 * L_B$  then  $t[i] \leftarrow block$ 
24:    else if  $\Delta Z < 0$  then  $t[i] \leftarrow wedge_f$ 
25:    else if  $\Delta Z > 0$  then  $t[i] \leftarrow wedge_b$ 
26:    end if
27:  end for
28:  return  $\langle h, t \rangle$ 
29: end function
```

10.4.2 Algorithm

We present BB-MST, a branch-and-bound algorithm to solve struct-MST. Branch-and-bound is one of the most general algorithmic techniques for solving combinatorial optimization problems, and has in particular been employed as a practical technique to solve a number of \mathcal{NP} -hard problems [49]. The method solves cost-minimization problems through an intelligently structured search of the space of all feasible solutions: it repeatedly partitions (“branches”) the solution space into subsets and computes a lower bound on the cost of the solutions within each subset. After each branching step, the lower-bound cost of each subset is compared to the cost of the best-yet feasible solution, and those with cost bounds exceeding the best-yet solution are ignored. Employing branch-and-bound does not change the *worst-case* runtime of an \mathcal{NP} -hard problem, and for struct-MST the worst-case runtime remains exponential in the number of potential conflicts. However, branch-and-bound can lead to a significant speedup of the *average* runtime, allowing many practical problems to be solved efficiently.

Algorithm 2 provides pseudocode for BB-MST. Each time BB-MST is called, it uses Kruskal’s algorithm to solve for $M_{R,\mathcal{T}}$, the MST of the regions and structures passed in as arguments, and compares it to the best-yet solution M^* . Regardless of whether $M_{R,\mathcal{T}}$ is a valid solution, if $\text{cost}(M_{R,\mathcal{T}}) \geq \text{cost}(M^*)$, BB-MST returns M^* , terminating search of the branch. If $M_{R,\mathcal{T}}$ is conflict-free and $\text{cost}(M_{R,\mathcal{T}}) < \text{cost}(M^*)$, M^* is updated and $M_{R,\mathcal{T}}$ is returned.

If $M_{R,\mathcal{T}}$ includes a conflict, BB-MST recursively branches, solving two or more child problems in which some of the conflicting edges or nodes have been removed, and returns the cheapest of the child solutions. Structure-structure and region-structure conflicts are handled as follows:

Structure-Structure Conflicts

If there is conflicting pair of structures $\{T_1, T_2\}$ in $M_{R,\mathcal{T}}$, form two child problems, removing one conflicting structure from each: $M_1 = \text{BB-MST}(R, \mathcal{T} \setminus T_1)$ and $M_2 = \text{BB-MST}(R, \mathcal{T} \setminus T_2)$. Return the cheaper of the two solutions.

Region-Structure Conflicts

Region-structure conflicts are somewhat more complex. Let $\mathcal{T}(M_{R,\mathcal{T}})$ be the set of edges of $M_{R,\mathcal{T}}$. $\forall (T, r) \in \mathcal{T}(M_{R,\mathcal{T}}) \times R$, check whether T has more than one structure cell containing a boundary node of r ; if so, T might split r . Let r_c be one such region, and $\mathcal{T}_c = \{T_1, T_2, \dots, T_k\}$ be the set of structures meeting this condition. Let $V_{\mathcal{T}_c}$ be the set of nodes in the grid-graph $G(V, E)$ occupied by \mathcal{T}_c , V_{r_c} be the set of nodes in r_c , and G_{r_c} be the subgraph of $G(V, E)$ induced by $V_{r_c} \setminus V_{\mathcal{T}_c}$. Compute R_{r_c} , the set of connected components of G_{r_c} : if R_{r_c} has more than one element, r_c has been split and is in conflict with \mathcal{T}_c .

We handle this by forming $k + 1$ branches, where k is the number of structures in \mathcal{T}_c . In each of the first k branches, we remove one conflicting structure: $M_i = \text{BB-MST}(R, \mathcal{T} \setminus T_i) \forall T_i \in \mathcal{T}_c$. In the final branch, we keep all conflicting structures, but split the region r_c into multiple sub-regions: Letting $R_{\text{split}} = (R \setminus r_c) \cup R_{r_c}$, we have $M_{\text{split}} = \text{BB-MST}(R_{\text{split}}, \mathcal{T})$. We return the cheapest of all solutions $\{M_1, M_2, \dots, M_k, M_{\text{split}}\}$.

Algorithm 2 BB-MST Algorithm

Require: Regions R and candidate structures \mathcal{T}

```
1: Initialize  $M^* \leftarrow \emptyset$ 
2: function BB-MST( $R, \mathcal{T}$ )
3:   Form  $G_R(R, \mathcal{T})$ 
4:   if  $G_R$  is not a connected graph then return  $\emptyset$ 
5:   end if
6:    $M_{R,\mathcal{T}} \leftarrow \text{kruskal}(G_R)$ 
7:   if  $\text{cost}(M_{R,\mathcal{T}}) \geq \text{cost}(M^*)$  then return  $M^*$ 
8:   end if
9:   if  $\text{cost}(M_{R,\mathcal{T}}) < \text{cost}(M^*)$  and  $M_{R,\mathcal{T}}$  is valid then
10:     $M^* \leftarrow M_{R,\mathcal{T}}$ 
11:    return  $M_{R,\mathcal{T}}$ 
12:   end if
13:   if  $M_{R,\mathcal{T}}$  has a structure conflict  $\{T_1, T_2\}$  then
14:    return BRANCHEDGES( $\{T_1, T_2\}, R, \mathcal{T}$ )
15:   else if  $M$  has a region conflict  $\{r_c, \mathcal{T}_c\}$  then
16:     $M_{\text{edges}} \leftarrow \text{BRANCHEDGES}(\mathcal{T}_c, R, \mathcal{T})$ 
17:     $M_{\text{split}} \leftarrow \text{BRANCHREGION}(r_c, \mathcal{T}_c, R, \mathcal{T})$ 
18:    return the cheaper of  $M_{\text{edges}}, M_{\text{split}}$ 
19:   end if
20: end function

21: function BRANCHEDGES( $\mathcal{T}_c, R, \mathcal{T}$ )
22:    $M \leftarrow \emptyset$ 
23:   for all  $T \in \mathcal{T}_c$  do
24:     $M_T \leftarrow \text{BB-MST}(R, \mathcal{T} \setminus T)$ 
25:    if  $\text{cost}(M_T) < \text{cost}(M)$  then  $M \leftarrow M_T$ 
26:    end if
27:   end for
28:   return  $M$ 
29: end function

30: function BRANCHREGION( $r_c, \mathcal{T}_c, R, \mathcal{T}$ )
31:    $R_{r_c} \leftarrow \text{split\_into\_subregions}(r_c, \mathcal{T}_c)$ 
32:    $R_{\text{split}} \leftarrow (R \setminus r_c) \cup R_{r_c}$ 
33:   for all  $T \in \mathcal{T}$  do
34:    if  $T$  connected to  $r_c$  then
35:     Reassign  $T$  to connect to appropriate  $r \in R_{\text{split}}$ 
36:    end if
37:   end for
38:   return BB-MST( $R_{\text{split}}, \mathcal{T}$ )
39: end function
```

10.4.3 Proof

Lemma 5 (Child Bounding). Let $M_{R,\mathcal{T}}$ (returned by Kruskal’s algorithm, Algorithm 2 line 6) contain one or more conflicts. $\text{cost}(M_{R,\mathcal{T}})$ is a lower bound on the cost of the solution to any child problem formed by branching on a conflict in $M_{R,\mathcal{T}}$.

Proof. Whenever BB-MST branches, it either eliminates one structure, or it splits one region into multiple regions. Consider the case where structure T has been eliminated. It is clear that $\text{cost}(M_{R,\mathcal{T}}) \leq \text{cost}(M_{R,\mathcal{T} \setminus T})$: $M_{R,\mathcal{T} \setminus T}$ is optimal with respect to $\mathcal{T} \setminus T$, so making T available could only decrease cost. Consider the case where region $r_c \in R$ has been split into two sub-regions r_1, r_2 , resulting in a new set of regions $R_{\text{split}} = (R \setminus r_c) \cup \{r_1, r_2\}$. $M_{R_{\text{split}},\mathcal{T}}$ is the MST which spans R_{split} . Form a new graph \mathcal{M} identical to $M_{R_{\text{split}},\mathcal{T}}$ except that nodes r_1, r_2 have been merged together to form node r_c . We may remove one edge from this graph to form a tree, which we will denote M' . By construction, $\text{cost}(M_{R_{\text{split}},\mathcal{T}}) = \text{cost}(\mathcal{M}) \geq \text{cost}(M')$. M' spans the same set of nodes as $M_{R,\mathcal{T}}$, but $M_{R,\mathcal{T}}$ is the MST, so $\text{cost}(M_{R,\mathcal{T}}) \leq \text{cost}(M') \leq \text{cost}(M_{R_{\text{split}},\mathcal{T}})$. \square

We prove by induction that each recursive call of $\text{BB-MST}(R, \mathcal{T})$ returns either $M_{R,\mathcal{T}}^*$ or M^* (the best-yet solution), whichever is cheaper.

Base Case

There are three conditions under which BB-MST returns without branching. **(1) Null:** G_R is disconnected, so no spanning tree can be found and BB-MST returns \emptyset . **(2) Shortcut:** $\text{cost}(M_{R,\mathcal{T}}) \geq \text{cost}(M^*)$, so BB-MST returns M^* ; by Lemma 5, this branch cannot contain a solution cheaper than the current best-yet solution M^* , so we stop exploring it. **(3) Success:** $\text{cost}(M_{R,\mathcal{T}}) < \text{cost}(M^*)$ and $M_{R,\mathcal{T}}$ is conflict-free, so we set $M^* \leftarrow M_{R,\mathcal{T}}$ and return $M_{R,\mathcal{T}}$. In this case, it is clear that $M_{R,\mathcal{T}} = M_{R,\mathcal{T}}^*$ since Kruskal’s algorithm produces a min-cost spanning tree, which is explicitly verified as conflict-free.

Induction Step

When $M_{R,\mathcal{T}}$ has a conflict, BB-MST forms two or more child branches and returns the cheapest solution among them. Assuming recursive calls of $\text{BB-MST}(R, \mathcal{T})$ return $M_{R,\mathcal{T}}^*$ for the set of regions and structures passed down to them, we prove that the optimal solution to the parent problem must be the cheapest of its children.

Let $M_{R,\mathcal{T}}$ contain structure-structure conflict set $\{T_1, T_2\}$. Since $\{T_1, T_2\}$ cannot be in the solution, we know $M_{R,\mathcal{T}}^*$ must be a subset of either $\mathcal{T} \setminus T_1$ or $\mathcal{T} \setminus T_2$, since $\mathcal{P}(\mathcal{T} \setminus T_1) \cup \mathcal{P}(\mathcal{T} \setminus T_2)$ is the set of all subsets of \mathcal{T} that do not contain $\{T_1, T_2\}$. Therefore, $M_{R,\mathcal{T}}^*$ must be the cheaper of $M_{R,\mathcal{T} \setminus T_1}^*$ and $M_{R,\mathcal{T} \setminus T_2}^*$.

By similar reasoning, let $M_{R,\mathcal{T}}$ contain a region-structure conflict set $\{r_c, \mathcal{T}_c\}$. We know that either $M_{R,\mathcal{T}}^* = M_{R,\mathcal{T} \setminus \mathcal{T}_c}^*$ for some $\mathcal{T}_c \in \mathcal{T}_c$, or $M_{R,\mathcal{T}}^* = M_{R_{\text{split}},\mathcal{T}}^*$, since in each of these cases a single member of the conflict set has been removed. BB-MST returns the cheapest of these options by comparing M_{edges} and M_{split} , returned by `BRANCHEDGES` and `BRANCHREGION`.

10.4.4 Runtime

BB-MST resolves conflicts by recursively exploring each possible conflict-free subset of the conflict set, so in the worst case it will explore an exponential number of branches before

finding a solution. This is to be expected: struct-MST is NP-Hard, and BB-MST solves the problem exactly.

In practice, BB-MST typically prunes many branches and explores a tiny fraction of this space. Additionally, once a feasible solution is found, BB-MST has an *anytime* property: it can be terminated at any time and return the best-yet feasible solution. To identify that no conflict-free spanning solutions exists, the algorithm must explore each branch until G_R becomes disconnected, which can be very time-consuming.

It is worth noting that relaxing the optimality requirement of struct-MST would not improve the worst-case runtime, because deciding the existence of a (non-minimal) spanning tree of structures is also \mathcal{NP} -hard.

10.5 Results

10.5.1 Examples and Experiments

Our Python implementation can solve for optimal sets of structures for the SMORES-EP robot and building blocks given a height-field representation of an environment. The implementation accepts CAD models and 3D maps of real environments as inputs, and we show that it can solve for optimal sets of structures in real-world indoor environments.

CAD Example – Table-and-Chair

Figure 62 shows optimal solutions for two configurations of the table-and-chair CAD example environment from Figure 61. In Example (A) -“Chair Pulled Out”, the optimal solution uses one large ramp from the floor to the chair seat, and two smaller structures connecting the seat to the tabletop and tabletop to chair top. Notice the position the structure connecting the tabletop to the chair top: the structure crosses over the chair seat, and has been placed at the far left edge of the chair seat to avoid creating a region-structure conflict (which would have required a fourth structure). In Example B, the chair has been pushed in further, and there is no longer enough space to place structures on the chair seat. The algorithm is forced to select a more expensive solution using three large ramps from the floor.

Real-World Experiments

Figure 63 shows solutions generated from two indoor environments. The video accompanying this work¹ shows the SMORES-EP robot moving through these environments using structures placed in the locations selected by the algorithm.

To solve for structures in these environments, 3D occupancy grid maps were created with the Octomap library [37] using point cloud data collected with a Microsoft Kinect RGB-D sensor. A similar RGB-D sensor can be carried by SMORES-EP to autonomously map its environment [16]. Occupancy grids were converted to height fields (2D grayscale images), which were smoothed (median filter, window of 3 pixels) and segmented (K-means, K=150) to reduce noise before running the algorithm to generate structures. Real environments often contain regions that are too small for the robot to occupy them, even if it could access them (for example, the arms and back of the chair in Figure 63). To account for this, we test whether each region can fit an inscribed 8cm square (the size of one SMORES-EP robot module) anywhere within it, and remove small regions from the set of regions R before running the algorithm.

¹<https://youtu.be/B9WM557NP44>

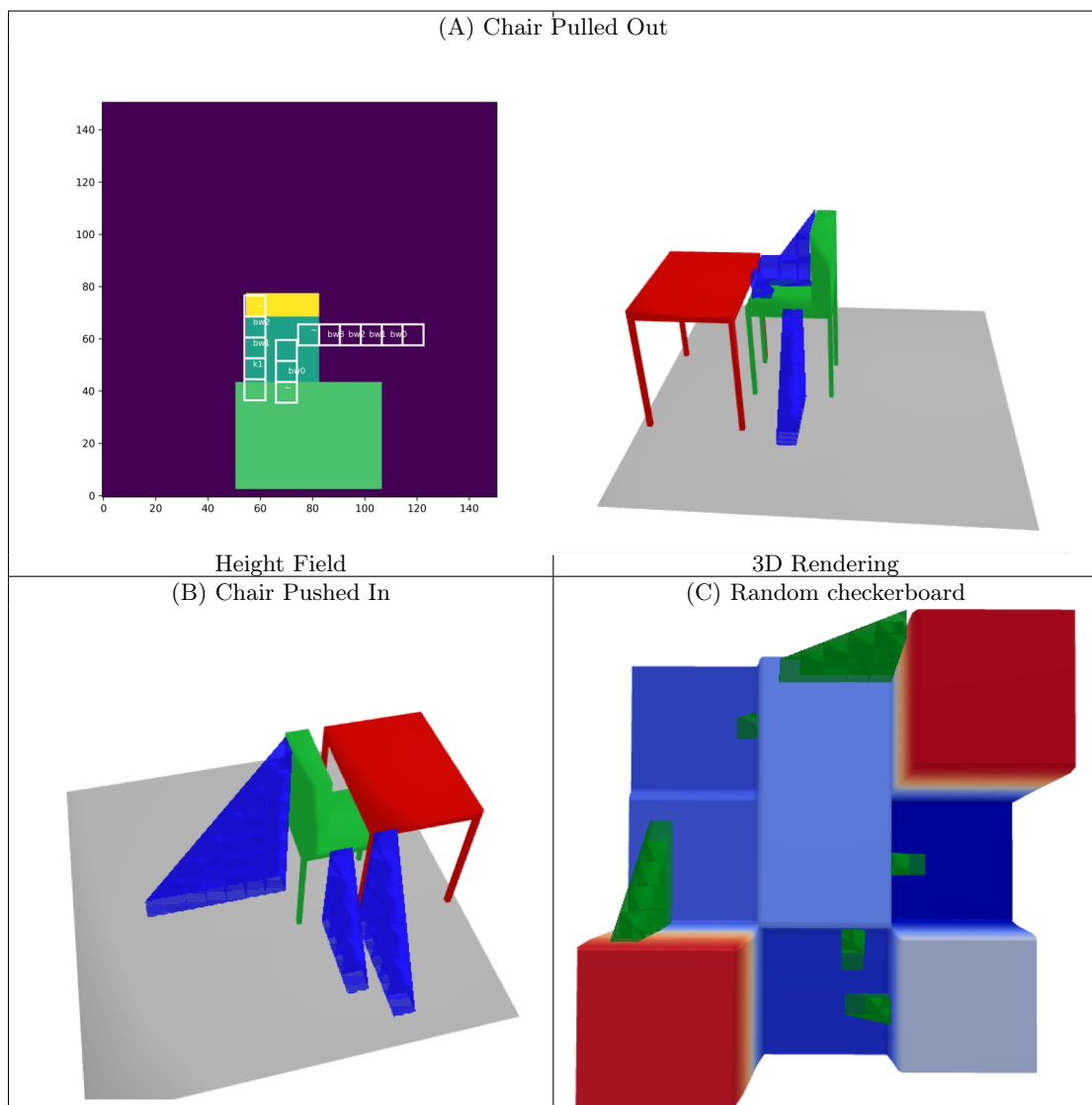


Figure 62: Optimal solutions in simulated environments.

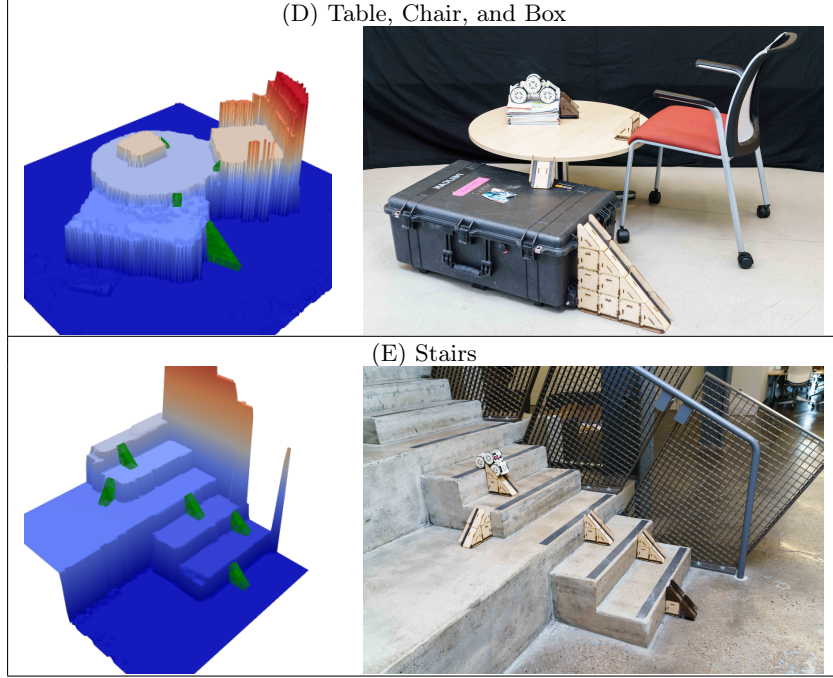


Figure 63: Real-world environments, and algorithm solutions generated from 3D map data taken with a Kinect sensor.

In Figure 63, Environment (D) consists of a round table, a stack of magazines, an office chair, and a storage container. The algorithm determines that adding one 4-block high ramp structure and three single wedges to this environment will allow the robot to reach every surface large enough to support it. Environment (E) is a staircase. The algorithm determines that six 2-block high ramps can be introduced to make it globally traversable by the robot. The location of each ramp on a given step is effectively random, because solutions that use the same set of structures in different locations have equal cost.

10.5.2 Runtime Performance

Table 11 shows metrics for the example environments from Figs 62 and 63. The algorithm generates thousands of structures and solves problems with 4-9 regions in minutes. In many cases, the number of potential conflicts (which determines the worst-case runtime) is in the tens of thousands, but the algorithm explores a tiny fraction of them (less than thirty). The Conflict Pair Fraction (CPF) for each problem is the percentage of pairs of structures which conflict, and provides a measure of problem difficulty. In all examples, a significant fraction of the total time required to reach a solution is spent preprocessing the world (e.g. generating the initial grid graph and identifying cliffs edges and steep areas).

Runtime performance was profiled by generating and solving random environments similar to environment (C) (Fig 62). Each environment is a 3×3 checkerboard with $6L_B$ wide squares with randomly selected heights between zero and $3L_B$ (in one-pixel increments). Trials were terminated after a timeout of 10 minutes. Of 1118 total trials, 981 found a solution (conflict-free MST), and 137 found that no solution exists. 834 solutions were found in one try, and 16 no-solutions were found in one try. 17 trials timed out, of which 11 found

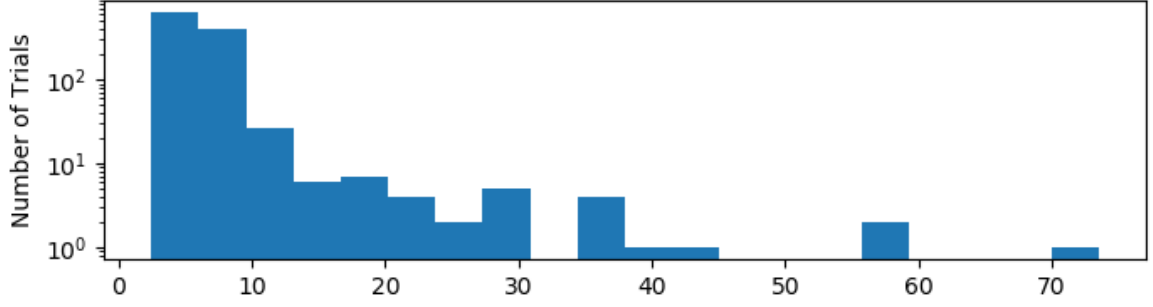


Figure 64: Log-histogram of solution times for 1118 random environments. In addition to the data shown, 17 environments timed out after 10 minutes.

Environment	(A)	(B)	(C)	(D)	(E)
Size (square, px)	151	151	72	240	115
Structures	634	148	474	3302	1275
Regions	4	4	9	5	6
Branches Explored	33	1	13	1	1
Potential Conflicts	16196	476	12974	33908	18206
CPF	14.0%	16.3%	7.6%	3.3%	4.7%
Structure Time (s)	9.834	2.25	3.188	61.824	9.507
BB-MST Time (s)	6.642	0.024	9.588	0.552	0.14
Total Time (s)	20.604	6.329	13.828	86.848	15.708
Blocks used	17	46	40	13	15

Table 11: Runtime Performance

no solution and 6 found a feasible solution. On average, each problem generated 363.2 structures with 10184.6 conflicts, and CPF of 7.83%.

10.6 Discussion and Future Work

This chapter presents an algorithm to find min-cost spanning sets of structures allowing a robot to reach every surface of an environment. Finding optimal (as opposed to feasible) solutions to these problems is important - building larger structures takes more time, and in real scenarios the number of available building blocks is always limited. For example, the solution in environment (A) requires 17 blocks, whereas solution (B) (which is also a feasible solution for (A)) requires 46.

Given a build point and direction vector, WATERFALL generates an optimal structure in linear time, allowing thousands of candidate structures to be generated in under a minute. Given a set of candidate structures, BB-MST will always eventually find a conflict-free solution to a struct-MST problem, if one exists. Because struct-MST is NP-Hard, any algorithm that solves it will have exponential worst-case complexity, and for some problems (especially when no solution exists) BB-MST will run for an impractically long time before returning. However, in typical problems, BB-MST explores the solution space efficiently and returns optimal solutions in a few seconds. In many realistic problems, the number of potential

conflicts is relatively small compared to the number of potentially useful combinations of structures. For example, the (D) and (E) have low CPF values, and in both cases the first MST generated had no conflicts.

Some tasks might require a robot to access only a subset of the regions in an environment (as opposed to every region). The framework introduced in this chapter could be extended in a straightforward way to solve for min-cost *paths* of structures (connecting a pair of regions) by calling Dijkstra’s algorithm in place of Kruskal’s algorithm in BB-MST. With slight modification, the framework could also solve for approximately-optimal *Steiner trees* of structures, to make a selected subset of regions accessible. Solving for min-cost Steiner trees in graphs is \mathcal{NP} -hard, but poly-time algorithms can solve the problem approximately [9]. Calling such an algorithm in place of Kruskal’s algorithm would allow BB-MST to compute Steiner trees, as long as the approximation factor is taken into account when comparing solution costs in the shortcut-return case.

Future work includes taking robot path planning into account when selecting structure locations. For example, in the stairs environment in Figure 63, placing the ramps in a line would allow the robot to move more efficiently through the environment. Optimization of structure positions could be performed as a separate post-processing step after the algorithm selects structures, or information about structure position could be directly incorporated into the cost function for evaluating solutions.

The presented method assumes that structures may only be placed to sharp cliff features in the environment. This assumption is fairly realistic for the building blocks we consider, which work best when placed against nearly-vertical surfaces in indoor environments. However, this assumption also serves to simplify the geometric search problem by limiting the search for candidate structures to points on the region perimeters (a one-dimensional space). Future work that aims to extend our method to other building blocks may need to develop more general ways to delimit the space of possible solutions.

10.7 Conclusion

This chapter presents a complete, optimal algorithm to generate sets of structures that could be added to an environment to make it globally accessible to a robot. In experiments using real and simulated environments, we demonstrated that the algorithm can synthesize optimal sets of structures with practical speed in realistic settings. This opens up the possibility for a structure-building robot to enter a new environment and quickly determine what structures should be built to enable free movement, enabling tasks that would otherwise be very difficult or impossible for the robot.

Chapter 11

Conclusion and Future Work

This dissertation presented hardware, algorithms, and integrated systems that allow robots to employ reconfiguration and environment augmentation as viable strategies to address tasks. In the realm of self-reconfiguration, we demonstrated for the first time that reconfiguration can be successfully and autonomously deployed to complete tasks in unknown environments. In the realm of environment augmentation, we demonstrated that robots enhance their capabilities with respect to a challenging environment and task by building structures. The problems addressed in this thesis are far from solved. To conclude, this chapter highlights some of the limitations of the work, and presents a vision for future research in reconfiguration and environment augmentation.

11.1 Limitations

11.1.1 Reconfiguration

Our library-based framework is a pragmatic way of representing the capabilities of a modular robot: rather than attempting to map out the full space of configurations and behaviors for a set of modules, our library contains only configurations and behaviors that we know to be useful, because they were hand-made by a human designer for a specific purpose. That purpose is encoded with property labels, also provided by a human. This framework has the capacity to capture a huge (arguably unlimited) range of functionality, insofar as it can contain just about anything a person is willing and able to design. The strength of the library is that it *can* contain just about anything; but the downside is that we have sidestepped the question of what it *should* contain. Long-term deployment in a realistic setting (e.g. a person’s home) clearly requires a lot of different capabilities, and our ad-hoc hand-crafted design process is unlikely to scale up. One school of thought would be to take a “big data” approach, employing machine learning to generate a huge number of library entries, eventually generating enough to cover the full space of desired capabilities. Another approach would proceed from first principles, attempting to generate a minimal set (or “basis set”) of designs that spans the space of required capabilities with as little redundancy as possible.

Describing robot capabilities in a principled way is very hard to do, even with respect to a specific task or environment. Our framework encodes capabilities with labels that fundamentally serve to communicate intent between the library designer and end user. When the designer labels a behavior with the attribute “`climb`”, they are counting on the end user

to understand what “climb” means. A very interesting avenue for future work could lie in establishing unifying rules or laws about how properties are assigned, so that every property label has a well-defined physical meaning. It would also be useful to expand environment and behavior property labels to encode information about probability of success - a stair-climbing behavior might succeed with high probability on stairs with short steps, but fail often on stairs with tall steps.

11.1.2 Environment Augmentation

The work in this thesis serves as an initial demonstration that structure building can provide benefit to a robot in completing tasks. Viewed through the lens of real-world utility, the system we presented has some clear practical limitations. Our building blocks well work on flat, smooth surfaces, more or less limiting their use to indoor environments. Blocks can only be connected in straight lines, so structures need to be built serially - multiple robots can’t add blocks to a structure at the same time. Most SMORES-EP configurations can only carry one or two building blocks at a time, so transporting a large quantity of blocks into a new environment is time consuming.

In spite of these limitations, the success of our demonstrations with SMORES-EP serves as evidence that environment augmentation can provide real benefit to robots, and provides ample motivation to design future systems to take the concept further. The SMORES-EP building blocks were designed long after the SMORES-EP modules - they were a “retrofit” that brought a new feature to a robot that was never designed to build structures. In the future, environment-augmenting robots could be co-designed with the structures they build - imagine, for example, a mobile manipulator with an integrated silo of building materials, and a nail gun integrated into its arm for assembly.

11.2 Future Work

Reconfigurable Robots Many modular robot systems are designed with configurability as their primary goal: the systems are homogeneous (all modules are identical), and designed to connect in as many geometric configurations as possible [107]. This design philosophy implicitly assumes that all configurations are equally important. My experience deploying SMORES-EP in large-scale experiments involving complex tasks runs counter to this assumption: some capabilities of a reconfigurable system are decidedly more important than others. For example, the ability to drive precisely and predictably proved essential for nearly every task we attempted with SMORES-EP, whereas the ability to climb stairs was needed for only a few tasks.

Future work in reconfigurable systems could close the loop from these task-oriented experiments down to hardware design. Rather than emphasizing geometric and topological generality, we should strive to maximizing task coverage. Practically speaking, I believe this will mean embracing heterogeneity, building systems that include a small number of different module designs, each optimized for different purposes. In particular, to address the scaling issues discussed in the previous section, we should consider building systems with modules with different physical sizes: large modules for heavy lifting, and small modules for fine-grained motions.

This idea can be taken further. “Hemi-modular” robots could combine a set of reconfigurable modules with a traditional, non-reconfigurable robot. For example, imagine a warehouse picking robot with a traditional, large manipulator arm, but with a wrist and

gripper made of modular robots that can disconnect and operate independently as mobile robots. This team of modules could fetch a box on a warehouse shelf and drive it back to the arm, and then reconfigure back into a wrist, allowing the arm to lift the box.

Environment Augmentation In my work with SMORES-EP, I have explored only one kind of environment augmentation - building structures that enhance the robot's ability to move. I believe future work in environment augmentation could go beyond locomotion, enhancing a robot's manipulation and perception capabilities as well. For example, to *manipulate* a challenging object, a robot might permanently attach a handle to it. The approach is particularly interesting for multi-robot teams, where the modified environment can be leveraged repeatedly. A team of robots might create a ladder by repeatedly attaching handles to a wall, making it possible to *traverse* this obstacle and reach a previously inaccessible region. In a heterogeneous team, one robot might use sophisticated sensing to identify objects and regions in its environment, and then place visible labels on them, allowing a team of other robots with basic cameras to easily *perceive* and navigate the area. Some modifications blur the line between robot and environment: four mobile robots equipped with strong adhesive connectors might attach themselves to four legs of a table, transforming a piece of the environment into a mobile base.

To employ environment augmentation as a strategy to accomplish tasks, we need to develop theory about when and how robots should alter their environments. My work in Chapters 9 and 10 begin to tackle these questions, but there are ample opportunities to go further. Future research could focus on developing metrics and theory that evaluate the relative costs and benefits of environment augmentation in the context of a task and environment. This is a challenging interdisciplinary problem. Evaluating the *benefits* of altering the environment requires an understanding of the relationship between the robot, its environment, and its task. Will attaching a handle to an object allow the robot to carry it? Do the benefits of building a structure justify the time, energy, and materials required? These questions open up interesting research problems in mechanics, computational design, and perception. If the robot is sharing its environment with humans, evaluating the *costs* of augmentation requires knowledge in human factors and human-robot interaction: if the robot is attaching things to walls and furniture, how will the human inhabitants of the space respond to these alterations?

In the long term, this research has the potential influence the ways in which robots enter our daily lives. Much as the rise of automobiles heralded the construction of highways across the world, the rise of ubiquitous robots in our society will require widespread support infrastructure that enables robots to behave safely and effectively. Environment augmentation can be viewed as robots installing their own infrastructure in the environment. Fundamentally, this research explores the relationship between robot, infrastructure, and environment, and seeks to determine what kinds of infrastructure have the smallest cost and the largest benefit to robots. I believe the things we can learn from research in environment augmentation will enable us to most effectively design infrastructure for a world filled with robots.

Bibliography

- [1] Unity3d. <http://unity3d.com/>. Accessed: 2015-04-35.
- [2] Potentiometers. www.novotechnik.de/en/products/sensor-technologies/potentiometers. Accessed: 8/1/16.
- [3] *Measurement Systems Analysis*. Chrysler Corp., 2010.
- [4] *Sensoink Technical Specifications*, 2016. www.sensoink.com.
- [5] *Membrane Potentiometer*, 2016. www.spectrasymbol.com.
- [6] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G.J. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *Robotics Automation Magazine, IEEE*, 14(1):61–70, March 2007.
- [7] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. Sampling-based motion planning with temporal goals. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2689–2696. IEEE, 2010.
- [8] Stéphane Bonardi, Massimo Vespignani, Rico Moeckel, Jesse Van den Kieboom, Soha Pouya, Alexander Sproewitz, and Auke Ijspeert. Automatic generation of reduced cpg control networks for locomotion of arbitrary modular robot structures. In *Proceedings of Robotics: Science and Systems*, 2014.
- [9] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved lp-based approximation for steiner tree. In *ACM Symposium on Theory of Computing*, pages 583–592. ACM, 2010.
- [10] Sebastian Castro, Sarah Koehler, and Hadas Kress-Gazit. High-level control of modular robots. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3120–3125. IEEE, 2011.
- [11] I-Ming Chen and Joel W Burdick. Determining task optimal modular robot assembly configurations. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 1, pages 132–137. IEEE, 1995.
- [12] Gregory S Chirikjian. Kinematics of a metamorphic robotic system. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 449–455. IEEE, 1994.

- [13] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Intro. to Algorithms*, chapter 15, pages 394–395. MIT Press, 3rd edition, 2009.
- [14] Andreas Darmann, Ulrich Pferschy, Joachim Schauer, and Gerhard J Woeginger. Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16):1726–1735, 2011.
- [15] J. Daudelin and M. Campbell. An adaptable, probabilistic, next-best view algorithm for reconstruction of unknown 3-d objects. *IEEE Robotics and Automation Letters*, 2(3):1540–1547, July 2017. ISSN 2377-3766. doi: 10.1109/LRA.2017.2660769.
- [16] Jonathan Daudelin*, Gangyuan Jing*, Tarik Tosun*, Mark Yim, Hadas Kress-Gazit, and Mark Campbell. An integrated system for perception-driven autonomy with modular robots. In *Preparation*. URL <https://arxiv.org/abs/1709.05435>.
- [17] Jay Davey, Ngai Kwok, and Mark Yim. Emulating self-reconfigurable robots-design of the smores system. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4464–4469. IEEE, 2012.
- [18] R. Diestel. *Graph Theory*, chapter 1. Springer, 4th edition, 2010.
- [19] Mehmet R Dogar and Siddhartha S Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, 33(3):217–236, 2012.
- [20] Marco Dorigo, Elio Tuci, Roderich Groß, Vito Trianni, Thomas Halva Labella, Shervin Nouyan, Christos Ampatzis, Jean-Louis Deneubourg, Gianluca Baldassarre, Stefano Nolfi, Francesco Mondada, Dario Floreano, and Luca Maria Gambardella. The SWARM-BOTS Project. *LNCS*, 3342:31–44, 2005.
- [21] Marco Dorigo, Dario Floreano, Luca Maria Gambardella, Francesco Mondada, Stefano Nolfi, Tarek Baaboura, Mauro Birattari, Michael Bonani, Manuele Brambilla, Arne Brutschy, Daniel Burnier, Alexandre Campo, Anders Lyhne Christensen, Antal Decugniere, Gianni Di Caro, Frederick Ducatelle, Eliseo Ferrante, Alexander Förster, Javier Martinez Gonzales, Jerome Guzzi, Valentin Longchamp, Stephane Magnenat, Nithin Mathews, Marco Montes De Oca, Rehan O’Grady, Carlo Pinciroli, Giovanni Pini, Philippe Rétornaz, James Roberts, Valerio Sperati, Timothy Stirling, Alessandro Stranieri, Thomas Stützle, Vito Trianni, Elio Tuci, Ali Emre Turgut, and Florian Vausard. Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine*, 20(4):60–71, 2013.
- [22] Ayan Dutta, Prithviraj Dasgupta, and Carl Nelson. Distributed configuration formation with modular robots using (sub) graph isomorphism-based approach. *Autonomous Robots*, pages 1–21, 2018.
- [23] Nick Eckenstein and Mark Yim. Modular advantage and kinematic decoupling in gravity compensated robotic systems. *Journal of Mechanisms and Robotics*, 5(4):041013, 2013.

- [24] Susan Finger, James Rinderle, et al. *A transformational approach to mechanical design using a bond graph grammar*. [Carnegie Mellon University], Engineering Design Research Center, 1990.
- [25] C. Finucane, Gangyuan Jing, and H. Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1988–1993, Oct 2010.
- [26] Toshio Fukuda and Yoshio Kawauchi. Cellular robotic system (cebot) as one of the realization of self-organizing intelligent universal manipulator. In *ICRA*, pages 662–667. IEEE, 1990.
- [27] Kyle Gilpin, Keith Kotay, Daniela Rus, and Iuliu Vasilescu. Miche: Modular shape formation by self-disassembly. *The International Journal of Robotics Research*, 27(3-4):345–372, 2008.
- [28] Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *ICRA*, pages 2485–2492. IEEE, 2010.
- [29] Robert Grabowski, Luis E. Navarro-Serment, Christiaan J J Paredis, and Pradeep K. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, 8(3):293–308, 2000.
- [30] Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 479–488. ACM, 2011.
- [31] Roderich Groß, Michael Bonani, Francesco Mondada, and Marco Dorigo. Autonomous self-assembly in swarm-bots. *IEEE transactions on robotics*, 22(6):1115–1130, 2006.
- [32] Bahar Haghighat, Emmanuel Droz, and Alcherio Martinoli. Lily: A miniature floating robotic platform for programmable stochastic self-assembly. In *ICRA*, pages 1941–1948. IEEE, 2015.
- [33] Harwin Inc. *S1791-42R Customer Information Sheet*, 2016.
- [34] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.
- [35] Gregory S Hornby, Hod Lipson, and Jordan B Pollack. Evolution of generative design systems for modular physical robots. In *ICRA*, volume 4, pages 4146–4151. IEEE, 2001.
- [36] Gregory S Hornby, Hod Lipson, and Jordan B Pollack. Generative representations for the automated design of modular physical robots. *IEEE transactions on Robotics and Automation*, 19(4):703–719, 2003.
- [37] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

- [38] Feili Hou and Wei-Min Shen. Graph-based optimal reconfiguration planning for self-reconfigurable robots. *Robotics and Autonomous Systems*, 62(7):1047–1059, 2014.
- [39] Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. An end-to-end system for accomplishing tasks with modular robots. In *Robotics: Science and Systems*, 2016.
- [40] Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. Accomplishing high-level tasks with modular robots. *Autonomous Robots*, pages 1–18, 2017.
- [41] Yoshihiro Kawahara et al. Instant inkjet circuits: lab-based inkjet printing to support rapid prototyping of ubicomp devices. In *ACM conference on Pervasive and Ubiquitous computing*, pages 363–372. ACM, 2013.
- [42] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 53(1): 287–297, Feb 2008.
- [43] Ara N Knaian et al. The milli-motein: A self-folding chain of programmable matter with a one centimeter module pitch. In *IROS*, pages 1447–1453. IEEE, 2012.
- [44] Ara Nerses Knaian. *Electropermanent magnetic connectors and actuators: devices and their application in programmable matter*. PhD thesis, Massachusetts Institute of Technology, 2010.
- [45] Ross A Knepper, Todd Layton, John Romanishin, and Daniela Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 855–862. IEEE, 2013.
- [46] Hadas Kress-Gazit, Gerogios E. Fainekos, and George J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6): 1370–1381, 2009.
- [47] Haruhisa Kurokawa, Kohji Tomita, Akiya Kamimura, Shigeru Kokaji, Takashi Hasuo, and Satoshi Murata. Distributed self-reconfiguration of m-tran iii modular robotic system. *The International Journal of Robotics Research*, 27(3-4):373–386, 2008.
- [48] M. Labbe and F. Michaud. Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. In *IROS*, pages 2661–2666, Sept 2014.
- [49] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.
- [50] Stéphane Magnenat, Roland Philippsen, and Francesco Mondada. Autonomous construction using scarce resources in unknown environments. *Autonomous Robots*, 33(4):467–485, 2012.
- [51] Spyros Maniatopoulos, Philipp Schillinger, Vitchyr Pong, David C Conner, and Hadas Kress-Gazit. Reactive high-level behavior synthesis for an atlas humanoid robot.

- In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4192–4199. IEEE, May 2016. doi: 10.1109/ICRA.2016.7487613.
- [52] Yannis Mantzouratos, Tarik Tosun, Sanjeev Khanna, and Mark Yim. On embeddability of modular robot designs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1911–1918. IEEE, 2015.
 - [53] B. McKay. Nauty user’s guide (v2.4). *Computer Science Dept., Australian Nat. Univ.*, 2007.
 - [54] AM Mehta, Nicola Bezzo, B An, P Gebhard, Vijay Kumar, Insup Lee, and Daniela Rus. A design environment for the rapid specification and fabrication of printable robots. In *14th International Symposium on Experimental Robotics (ISER’14), Marrakech/Essaouira, Morocco, June*, pages 15–18, 2014.
 - [55] Ankur M Mehta, Joseph DelPreto, Benjamin Shaya, and Daniela Rus. Cogeneration of mechanical, electrical, and software designs for printable robots from structural specifications. In *Intelligent Robots and Systems (IROS 2014)*, pages 2892–2897. IEEE, 2014.
 - [56] *Total Ground Carbon Conductive Coating 838 Technical Data Sheet*. MG Chemicals, January 2013. Ver. 1.04.
 - [57] Shuhei Miyashita, Laura Meeker, Maurice Go, Yoshihiro Kawahara, Daniela Rus, et al. Self-folding printable elastic electric devices: Resistor, capacitor, and inductor. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1446–1453. IEEE, 2014.
 - [58] Shuhei Miyashita et al. An untethered miniature origami robot that self-folds, walks, swims, and degrades. In *ICRA*. IEEE, 2015.
 - [59] Francesco Mondada, Luca Maria Gambardella, Dario Floreano, Stefano Nolfi, Jean Louis Deneubourg, and Marco Dorigo. The cooperation of swarm-bots: Physical interactions in collective robotics. *IEEE Robotics and Automation Magazine*, 12(2): 21–28, 2005.
 - [60] Satoshi Murata, Haruhisa Kurokawa, and Shigeru Kokaji. Self-assembling machine. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 441–448. IEEE, 1994.
 - [61] Satoshi Murata, Eiichi Yoshida, Akiya Kamimura, Haruhisa Kurokawa, Kohji Tomita, and Shigeru Kokaji. M-tran: Self-reconfigurable modular robotic system. *IEEE/ASME transactions on mechatronics*, 7(4):431–441, 2002.
 - [62] Satoshi Murata, Kiyoharu Kakomura, and Haruhisa Kurokawa. Docking experiments of a modular robot by visual feedback. *IEEE International Conference on Intelligent Robots and Systems*, pages 625–630, 2006.
 - [63] Nils Napp and Radhika Nagpal. Distributed amorphous ramp construction in unstructured environments. *Robotica*, 32(2):279–290, 2014.

- [64] Nils Napp and Radhika Nagpal. Robotic construction of arbitrary shapes with amorphous materials. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 438–444. IEEE, 2014.
- [65] *Series PRS Resistance Elements*. Novotechnik Siedle Group, 2016. http://www.novotechnik.com/pdfs/PRS_e.pdf.
- [66] Rehan O’Grady, Roderich Groß, Francesco Mondada, Michael Bonani, and Marco Dorigo. Self-assembly on demand in a group of physical autonomous mobile robots navigating rough terrain. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3630 LNAI(September):272–281, 2005.
- [67] Simon Olberding, Nan-Wei Gong, John Tiab, Joseph A Paradiso, and Jürgen Steimle. A cuttable multi-touch sensor. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 245–254. ACM, 2013.
- [68] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011.
- [69] Esben Hallundbæk Østergaard, Kristian Kassow, Richard Beck, and Henrik Hautop Lund. Design of the atron lattice-based self-reconfigurable robot. *Autonomous Robots*, 21(2):165–183, 2006.
- [70] Rehan O’Grady, Roderich Groß, Anders Lyhne Christensen, and Marco Dorigo. Self-assembly strategies in a group of autonomous mobile robots. *Autonomous Robots*, 28(4):439–455, 2010.
- [71] Michael Park, Sachin Chitta, Alex Teichman, and Mark Yim. Automatic configuration recognition methods in modular robots. *The International Journal of Robotics Research*, 27(3-4):403–421, 2008.
- [72] James Paulos, Nick Eckenstein, Tarik Tosun, Jungwon Seo, Jay Davey, Jonathan Greco, Vijay Kumar, and Mark Yim. Automated Self-Assembly of Large Maritime Structures by a Team of Robotic Boats. *IEEE Transactions on Automation Science and Engineering*, pages 1–11, 2015.
- [73] T. Peixoto. The graph-tool python library. <http://graph-tool.skewed.de/>, 2014.
- [74] Kirstin Petersen, Radhika Nagpal, and Justin Werfel. Termes: An autonomous robotic system for three-dimensional collective construction. *Proc. Robotics: Science & Systems VII*, 2011.
- [75] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC ’15*, pages 239–248, New York, NY, USA, 2015.

- [76] John W Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus. 3d m-blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1925–1932. IEEE, 2015.
- [77] M. Rubenstein, K. Payne, P. Will, and Wei-Min Shen Wei-Min Shen. Docking among independent and autonomous CONRO self-reconfigurable robots. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, 3:2877–2882, 2004.
- [78] Graham G Ryland and Harry H Cheng. Design of imobot, an intelligent reconfigurable mobile robot with novel locomotion. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 60–65. IEEE, 2010.
- [79] Maira Saboia, Vivek Thangavelu, Walker Gosrich, and Nils Napp. Autonomous adaptive modification of unstructured environments. In *Robotics: Science and Systems*, 2018.
- [80] Behnam Salemi, Wei-Min Shen, and Peter Will. Hormone-controlled metamorphic robots. In *ICRA*, volume 4, pages 4194–4199. IEEE, 2001.
- [81] Behnam Salemi, Mark Moll, and WM Shen. Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *IROS*, pages 3636–3641. IEEE, 2006.
- [82] Adriana Schulz, Cynthia Sung, Andrew Spielberg, Wei Zhao, Yu Cheng, Ankur Mehta, Eitan Grinspun, Daniela Rus, and Wojciech Matusik. Interactive robogami: data-driven design for 3d print and fold robots with ground locomotion. In *SIGGRAPH 2015: Studio*, page 1. ACM, 2015.
- [83] Jungwon Seo, Mark Yim, and Vijay Kumar. Assembly planning for planar structures of a brick wall pattern with rectangular modular robots. *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1016–1021, aug 2013.
- [84] Wei-Min Shen, Robert Kovac, and Michael Rubenstein. Singo: a single-end-operative and genderless connector for self-reconfiguration, self-assembly and self-healing. In *ICRA*, pages 4253–4258. IEEE, 2009.
- [85] Alexander Sproewitz, Rico Moeckel, Jérôme Maye, and Auke Jan Ijspeert. Learning to move in modular robots using central pattern generators and online optimization. *The International Journal of Robotics Research*, 27(3-4):423–443, 2008.
- [86] Kasper Stoy, Wei-Min Shen, and Peter M Will. Using role-based control to produce locomotion in chain-type self-reconfigurable robots. *IEEE/ASME transactions on mechatronics*, 7(4):410–417, 2002.
- [87] John W Suh, Samuel B Homans, and Mark Yim. Telecubes: Mechanical design of a module for self-reconfigurable robotics. In *ICRA*, volume 4, pages 4095–4101. IEEE, 2002.

- [88] Cynthia Sung, James Bern, John Romanishin, and Daniela Rus. Reconfiguration planning for pivoting cube modular robots. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1933–1940. IEEE, 2015.
- [89] Jaeyong Sung, Seok Hyun Jin, Ian Lenz, and Ashutosh Saxena. Robobarista: Learning to manipulate novel objects via deep multimodal embedding. *arXiv preprint arXiv:1601.02705*, 2016.
- [90] Yuzuru Terada and Satoshi Murata. Automatic assembly system for a large-scale modular structure-hardware design of module and assembler robot. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2349–2355. IEEE, 2004.
- [91] Kohji Tomita, Satoshi Murata, Haruhisa Kurokawa, Eiichi Yoshida, and Shigeru Kokaji. Self-assembly and self-repair method for a distributed mechanical system. *IEEE Transactions on Robotics and Automation*, 15(6):1035–1045, 1999.
- [92] Tarik Tosun, Jay Davey, Chao Liu, and Mark Yim. Design and characterization of the ep-face connector. pages 45–51, 2016. ISSN 21530866. doi: 10.1109/IROS.2016.7759033.
- [93] Tarik Tosun, Daniel Edgar, Chao Liu, Thulani Tsabedze, and Mark Yim. Paintpots: Low cost, accurate, highly customizable potentiometers for position sensing. pages 1212–1218, 2017.
- [94] Tarik Tosun*, Jonathan Daudelin*, Gangyuan* Jing, Hadas Kress-Gazit, Mark Campbell, and Mark Yim. Perception-informed autonomous environment augmentation with modular robots. In *ICRA*, 2018. URL <https://arxiv.org/abs/1710.01840>.
- [95] Tarik Tosun, Gangyuan Jing, Hadas Kress-Gazit, and Mark Yim. Computer-aided compositional design and verification for modular robots. In *International Symposium on Robotics Research*, pages 237–252. Springer, 2018.
- [96] Tarik Tosun, Cynthia Sung, Colin McCloskey, and Mark Yim. Optimal structure synthesis for environment augmenting robots. *In Preparation*, 2019.
- [97] K. Ulrich and W. Seering. Synthesis of schematic descriptions in mechanical design. *Res. in Eng. Design*, pages 3–18, 1989.
- [98] *Industry Standard Wirewound and Nonwirewound Precision Potentiometers*. Variable Electronic Components Institute, 1988. Rev. A.
- [99] Jennifer E Walter, Elizabeth M Tsai, and Nancy M Amato. Choosing good paths for fast distributed reconfiguration of hexagonal metamorphic robots. In *ICRA*, 2002.
- [100] Justin Werfel, Donald Ingber, and Radhika Nagpal. Collective construction of environmentally-adaptive structures. *IEEE International Conference on Intelligent Robots and Systems*, pages 2345–2352, 2007.
- [101] Paul Joseph White. *Miniaturization methods for modular robotics: External actuation and dielectric elastomer actuation*. PhD thesis, University of Pennsylvania, 2011.

- [102] Kevin C Wolfe, Matthew S Moses, Michael DM Kutzer, and Gregory S Chirikjian. M 3 express: a low-cost independently-mobile reconfigurable modular robot. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2704–2710. IEEE, 2012.
- [103] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC '10*, pages 101–110, New York, NY, USA, 2010.
- [104] Guilin Yang and I-Ming Chen. Task-based optimization of modular robot configurations: minimized DoF approach. *Mechanism and machine theory*, 35(4):517–540, 2000.
- [105] M Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Stanford University, 1994.
- [106] Mark Yim, David G Duff, and Kimon D Roufas. Polybot: a modular reconfigurable robot. In *International Conference on Robotics and Automation*, volume 1, pages 514–520. IEEE, 2000.
- [107] Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.
- [108] Mark Yim, Babak Shirmohammadi, Jimmy Sastra, Michael Park, Michael Dugan, and Camillo J Taylor. Towards robotic self-reassembly after explosion. In *International Conference on Intelligent Robots and Systems, 2007*, pages 2767–2772. IEEE, 2007.
- [109] Ying Zhang, Mark Yim, Craig Eldershaw, Dave Duff, and Kimon Roufas. Phase automata: a programming model of locomotion gaits for scalable chain-type modular robots. In *IROS*, volume 3, pages 2442–2447. IEEE, 2003.