An End-to-End System for Accomplishing Tasks with Modular Robots: Perspectives for the AI Community

Gangyuan Jing
Cornell Univ.

Tarik Tosun
Univ. of Pennsylvania

Mark Yim
Univ. of Pennsylvania

Hadas Kress-Gazit
Cornell Univ.

Abstract

The advantage of modular robot systems lies in their flexibility, but this advantage can only be realized if there exists some reliable, effective way of generating configurations (shapes) and behaviors (controlling programs) appropriate for a given task. In this paper, we present an end-to-end system for addressing 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 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 paper condenses the material originally presented in [Jing *et al.*, 2016] into a shorter format suitable for a broad audience.

1 Introduction

Modular self-reconfigurable robots (MSRR) are systems composed of repeated robot elements (called *modules*) that connect together to form larger robotic structures. They distinguish themselves from traditional robots through their ability to *self-reconfigure*: changing the connective structure of the modules to assume different shapes with different capabilities. Over the last three decades, many kinds of MSRR have been built [Østergaard *et al.*, 2006; Kurokawa *et al.*, 2008; Lipson and Pollack, 2000], and many different approaches have been introduced for controlling and programming them [Salemi *et al.*, 2001; Stoy *et al.*, 2002]. In this work, we use the SMORES-EP robot, pictured in Figures 1 and 2.

In principal, MSRR systems can transform to meet the needs of each new task and environment they encounter. For example, such a system operating in a home setting might configure itself into a car to cross the kitchen floor, a snake to crawl into a cabinet and retrieve cooking ingredients, and an arm to help stir cake batter. The ability to reconfigure poses an obvious challenge: given a task, is it possible to automatically select an appropriate configuration (robot shape) and behavior (controlling program) to address it? This is an important unsolved problem in the field, and remains a significant barrier to using MSRR to solve real-world problems [Yim *et al.*, 2007].

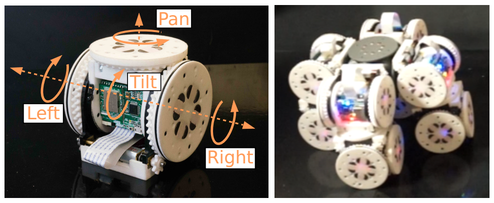


Figure 1: Left: A single SMORES-EP module. Right: Modules connected in the SwerveLifter configuration.

In this paper, we present a system that addresses this problem. Our system takes into account the capabilities of different robot configuration and behaviors, properties of the environment, and high-level task objectives specified by a user in order to select appropriate configurations and behaviors to complete the task. Through hardware experiments with the SMORES-EP robot, we demonstrate that it is capable of addressing a multi-part room-cleaning task.

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 self-reconfigurable robot hardware. While our work uses SMORES-EP, the system could be adapted to other modular robot platforms as well.

We leverage ideas from recent work on automatic controller synthesis with correctness guarantees from high-level task specification [Belta *et al.*, 2007; Bhatia *et al.*, 2010; Kloetzer and Belta, 2008; Kress-Gazit *et al.*, 2009; Raman *et al.*, 2015; Wongpiromsarn *et al.*, 2010]. These methods have proven effective for addressing high-level tasks with traditional robots; applying them 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. [Castro *et al.*, 2011] introduce a high-level control framework for the CKBot modular robot, which lays the theoretical foundations for our high-level mission planner, one of the four major components of our system. We also build upon [Tosun *et al.*, 2015], which introduces a physics-based simulator, design creation tool, and a small hierarchically organized library for the SMORES-EP robot.

This paper presents the details of the system, discusses its strengths and weaknesses, and provides a roadmap forward to

This work was funded by NSF grant numbers CNS-1329620 and CNS-1329692.

apply a similar system in a real-world setting.

2 System Structure

Here we provide an overview of each system component.

2.1 SMORES-EP Modular Robot

Our system is validated using the SMORES-EP modular self-reconfigurable robot. Four faces of the module are equipped with *electro-permanent magnets* (*EP magnets*) that allow modules to connect. 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 any energy. Some motions that the modules can perform are limited by the strength of the magnetic connectors, which can support three modules held out horizontally against gravity [Tosun *et al.*, 2016].

SMORES-EP has four actuated joints (Figure 1). The *left* and *right* faces of the module can be used as wheels to drive on smooth surfaces. The circular top face (called the *pan* joint) is also able to rotate continuously. A central bending joint (the *tilt* joint) has a 180° range of motion, allowing the top face to bend forward or backward until it is perpendicular to the bottom face. Each of the joints is equipped with a custom potentiometer for feedback control [Tosun *et al.*, 2017]. Each module has its own battery, microcontroller, and WiFi radio, allowing it to operate independently or as part of a cluster. A central computer running a Python program sends wireless messages to all modules.

2.2 Design and Simulation Tool - VSPARC

We developed an interactive software tool called VSPARC, which stands for **V**erification, **S**imulation, **P**rogramming And **R**obot **C**onstruction. Using graphical user interfaces and a physics engine, VSPARC allows users to design and test configurations and behaviors with unlimited number of modules in a simulated 3D environment. Moreover, users can command positions or velocities for each joint of all modules. VSPARC is available for free online at `www.vsparc.org`. Users can save and share their designs on a web server, allowing VSPARC to be used as a crowdsourcing tool to generate a large robot design library. Behaviors created in VSPARC can be run without modification on either the simulator or the physical SMORES-EP robot.

2.3 Design Library

Here we present the formalisms used to define our design library.

Modular Robot Systems A modular robot system consists of a set of basic units called *modules*, that can move and connect to other modules. Modules can be controlled by *joint commands*. We assume modules have feedback controllers (*e.g.* PID controllers) that can drive the corresponding joint to satisfy each command. A set of connected modules forms a *configuration*, which we treat as a single robot. A *behavior* for a configuration is a sequence of joint commands for all modules in the configuration, commanding it to perform an action.

Property A property is a descriptive label of the robot behavior or the environment. A property is defined as $p=(p_n,\Omega)$, where p_n is in an English description as the name of the property and Ω is the set of values of the property. For example, we use a robot property $p=(\text{Action},\{\text{Climb},\text{Push}\})$ to describe the robot's ability to *Climb* and *Push*. Properties are also used to describe the environment state in which the robot performs the behavior. For example, the property $p=(\text{BoxMass},[2,5])$ specifies that the mass of a box in the environment could be between 2 and 5 module-weights. 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 $\Omega_1\subseteq\Omega_2$.

Robot Design Library The robot design library is a collection of configurations and behaviors labeled with environment and behavior properties. The design library is defined as $\mathcal{L}=\{l_1,l_2,...\}$ consisting of a set of library entries. A library entry $l=(C,B,P_e,P_r)$ consists of a configuration C, a behavior B, and sets of environment and behavior properties P_e and P_r respectively. As an example, the library entry:

$$\begin{aligned} & l=(C=\texttt{snake},B=\texttt{climb},P_e,P_r)\\ \text{where:}\quad & P_e=\{(\text{Ledge\_Height},[3])\}\\ \text{and:}\quad & P_r=\{(\text{Action},[\text{Climb}]),(\text{Speed},[1])\}\end{aligned}$$

represents a `snake` shape configuration performing a `climb` behavior with the speed of one module-length per second in the environment with a three module-length-high ledge. 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 our design library we distributed VSPARC to student volunteers, and hosted three hackathons to designing configurations and behaviors for various robot tasks. The library currently includes 52 designs and 97 behaviors contributed by 20 volunteers. We provide a representative sampling of the design library in Fig. 2.

2.4 Reactive Controller Synthesis and Execution

Controller synthesis refers to the process of generating a controller to satisfy a given declarative specification, or proving such controller does not exist. Existing work in [Finucane *et al.*, 2010; Kress-Gazit *et al.*, 2009] provides a framework to automatically generate robot controllers from high-level user specifications. We extend the framework to allow controller synthesis from the design library for modular robot systems.

In controller synthesis, environment events and robot capabilities are abstracted into sets of boolean variables, which represent sensed environment information or current robot actions. For example, the environment variable **Person** is `True` if and only if the robot is currently sensing a person with its camera. Similarly, the robot variable **Pickup** is `True` if and only if the robot is currently performing a pick up action. A wide range of robot tasks can be specified using Linear Temporal Logic (LTL). A tool called LTLMoP introduced in [Finucane *et al.*, 2010] allows inexperienced users to use a fixed English grammar instead of LTL to specify robot tasks. The framework introduced in [Kress-Gazit *et al.*, 2009] can automatically generate a high-level robot controller from a task specification, or decide such controller does not exist. The synthesized controller is a finite-state automaton, which is implemented continuously to satisfy the task specification.

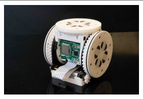
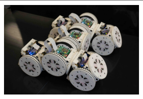
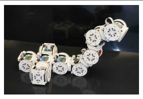
Configuration Name	Single module	Rolling Loop	DoubleDriver	Stair Climber	Swerve Lifter	backhoe
Number of modules	1	8	7	4	9	9
						
Locomotion						
Max robot height	1	2.5	1.5	2	2	4
Max robot width	1	1	3	1	4	4
Max robot length	1	5	3	3.5	3	7
Terrain - Rough		✓		✓		
Terrain - Sloped		✓	✓	✓		
Driving - Straight	✓	✓	✓	✓	✓	
Driving - Differential drive	✓		✓	✓	✓	
Driving - Holonomic					✓	
Manipulation						
Attachment - Push	✓	✓	✓	✓	✓	✓
Attachment - Magnetic	✓		✓	✓		✓
Attachment - Carry					✓	
Workspace size	$X: [-\inf, \inf]$ $Y: [-\inf, \inf]$ $Z: [0, 1]$	$X: [-\inf, \inf]$ $Y: [0, 1]$ $Z: [0, 2.5]$	$X: [-\inf, \inf]$ $Y: [-\inf, \inf]$ $Z: [0, 1.5]$	$X: [-\inf, \inf]$ $Y: [-\inf, \inf]$ $Z: [0, 2]$	$X: [-\inf, \inf]$ $Y: [-\inf, \inf]$ $Z: [0, 2]$	$X: [-3, 3]$ $Y: [-3, 3]$ $Z: [0, 4]$
Payload mass	1	2	4	2	3	1

Figure 2: Matrix of designs and properties. (Unit Length = the side length of a module. Unit Mass = the mass of a module)

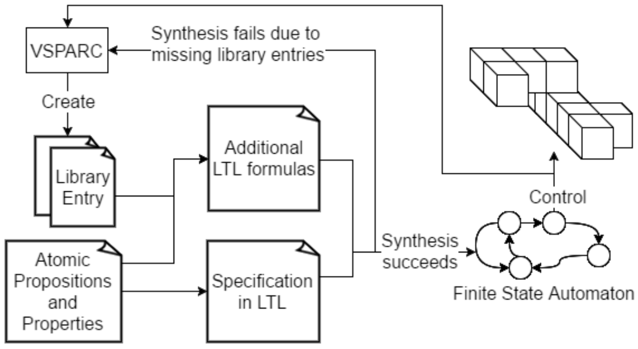


Figure 3: Controller synthesis and execution.

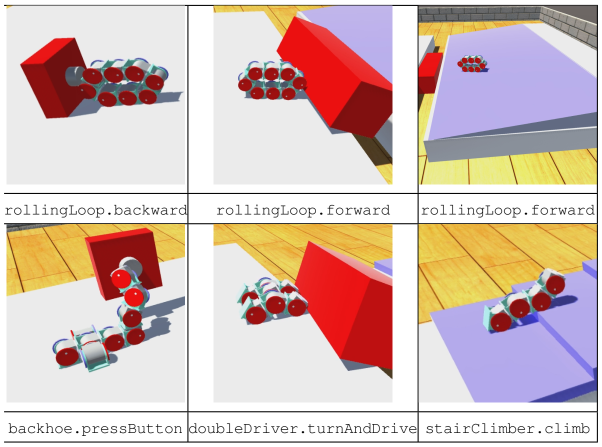


Figure 4: Simulated Demo

To use the robot design library with the controller synthesis framework, we allow users to assign a set of environment and behavior properties to each robot variable. For example, an action variable “Pickup” can be mapped to {(Box_Mass, [3]), (Action, [Pickup])}. Then we can search through the design library to find a set of library entries that satisfies all given properties. Once each robot action variable is mapped to a set of library entries, we need to ensure multiple robot actions can be executed simultaneously if needed. For example, consider two robot action variables “Pickup” and “Push”. If there does not exist a library entry mapped to both variables, we must guarantee that “Pickup” and “Push” actions cannot be executed simultaneously. This constraint is automatically encoded in an LTL formula and appended to the original task specification for synthesis.

3 Experimental Results

We demonstrate the capabilities of our system through experiments. An accompanying video can be found at `http://www.modlabupenn.org/2016/09/15/end-to-end/`.

3.1 Simulated Demonstrations

Scenario 1 The environment for Scenario 1 consists of a button, a lightweight block, a gap in the ground, and a ramp, all in a straight line. The robot’s objective is to move from its starting point to the goal area at the top of the ramp. When the button is pushed, it causes the block (which begins floating in the air) to fall to the ground, where it can be pushed into the gap, forming a bridge between the flat region and ramp. The action definitions for this task are: `pushButton` (height = 1.5); `pushBox` (payload = 2, distance_x = 3); `climb` (drive = Straight, terrain = Sloped). The high-level mission planner determines that all the properties are fulfilled by the `rollingLoop` configuration. In the top row of Figure 4 (and in the accompanying video), we see the rolling loop complete the task.

Scenario 2 The environment for Scenario 2 is similar to the environment from Scenario 1, but with several small changes that make the task more difficult. The button is now on the left side of the environment, and floats at a height of 4 module-lengths above the ground. The box is twice as heavy, weighing 4

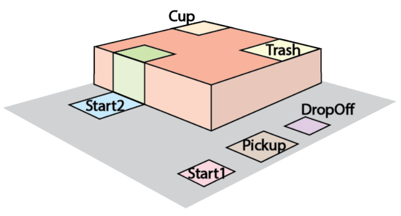


Figure 5: Map of hardware demo

module-weights rather than 2. The ramp has been replaced with stairs with the step height of 0.75.

The `rollingLoop` can no longer complete this task - it can't reach the button, it's not strong enough to push the box, and it can't ascend steps higher than 0.25. When the specification is compiled, our system selects three different configurations from the library to complete the task. The `backhoe` is used to push the button, because it is the only configuration with a large enough vertical workspace. It then reconfigures into the `doubleDriver` to drive over and push the box into the hole, because it is able to drive, turn, and push objects as heavy as 5 module-weights. Then, it reconfigures into the `stairClimber` to climb the stairs. In the bottom row of Figure 4 and the accompanying video, we see how the task is completed. For the purposes of this paper, we assume that reconfiguration between any two configurations is possible as long as the final configuration does not have more modules than the initial configuration.

3.2 Hardware Demonstrations

In these scenarios, a cluster of SMORES-EP modules is directed to clean the top of a table. The map is shown in Fig. 5. There are two components to this task: first, the robot must move a waste bin near the table, and then, the robot must climb up to the top of the table, explore the surface, and react appropriately to the objects it encounters. These two scenarios showcase the translation of behaviors from the simulator to hardware, and the ability to use LTLMoP to create and execute mission plans with the hardware. Since the modules have no sensors, localization is provided by AprilTags [Olson, 2011] mounted to the modules and objects of interest, tracked by an overhead camera.

Moving the Waste Bin In the first scenario, the cluster starts in region `Start1` and needs to move a waste bin from region `Pickup` to region `DropOff` (a distance of 10 module lengths) to be near the table, and then travel to the table edge (region `Start2`). The task is made more difficult by the fact that the waste bin is supported by four legs, so it cannot be pushed by designs with a height of two modules or less. The workspace requirement (10 modules lengths) rules out all stationary manipulators in the library, and height requirement rules out most car-like designs. Fortunately, the `swerveLifter` design, which can carry objects by driving under them and lifting up, is perfect for the job (Fig. 1). As seen in the accompanying video, the high-level controller directs the robot to complete the task. The robot waits until it senses the waste bin (marked with an AprilTag). Then it lowers itself, drives under the waste bin, carries it next to the table, and performs omnidirectional drive to travel to the table.

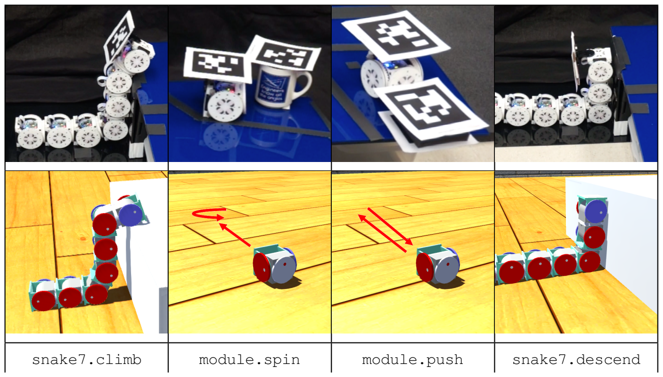


Figure 6: Cleaning the Table

Table Exploration With the waste bin in place, the cluster can clean the top of the table. It should explore the tabletop and react to what it finds: if it senses a piece of trash, it should push it off the table, and if it senses a coffee mug, it should back up and spin in place (alerting the mug's owner that it should be removed). After exploring both locations on the table, it should return to the ground.

The `snake7` configuration is selected to complete this task. it can use its `climbup` and `climbdown` 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 `differentialDrive` behavior to explore the tabletop, and its `spin`, and `push` behaviors to clean. The robot to successfully cleans the table, as shown in the accompanying video and Fig. 6. An overhead camera system tracks AprilTags attached to the first module of the snake, allowing LTLMoP to servo the left and right wheels of the module in differential drive and sense proximity to the coffee mug and trash (also marked with AprilTags).

4 Conclusion and Future Work

We presented an end-to-end system that is among the first to address complex, reactive, high-level tasks with modular self-reconfigurable robots. By providing this framework and demonstrating its success in the lab, we hope to lay the foundation for future systems to address tasks in the real world.

Environment and behavior properties provide an expressive way to specify task requirements, but the fact that a behavior is labeled with a specific property does not guarantee it will perform as intended if the environment is not similar enough to the one in which it was designed. Developing methods to automatically characterize new environments using sensor data is another avenue of future work.

References

[Belta *et al.*, 2007] Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *Robotics Automation Magazine, IEEE*, 14(1):61–70, March 2007.

[Bhatia *et al.*, 2010] 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.

[Castro *et al.*, 2011] 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.

[Finucane *et al.*, 2010] Cameron. Finucane, Gangyuan Jing, and Hadas. 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.

[Jing *et al.*, 2016] 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.

[Kloetzer and Belta, 2008] Marius Kloetzer and Calin 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.

[Kress-Gazit *et al.*, 2009] 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.

[Kurokawa *et al.*, 2008] Haruhisa Kurokawa, Kohji Tomita, Akiya Kamimura, Shigeru Kokaji, Takashi Hasuo, and Satoshi Murata. Distributed self-reconfiguration of m-tran iii modular robotic system. *IJRR*, 2008.

[Lipson and Pollack, 2000] Hod Lipson and Jordan B Pollack. Towards continuously reconfigurable self-designing robotics. In *ICRA*, 2000.

[Olson, 2011] 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.

[Østergaard *et al.*, 2006] 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.

[Raman *et al.*, 2015] 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.

[Salemi *et al.*, 2001] Behnam Salemi, Wei-Min Shen, and Peter Will. Hormone-controlled metamorphic robots. In *ICRA*, 2001.

[Stoy *et al.*, 2002] Kasper Stoy, Wei-Min Shen, and Peter M Will. Using role-based control to produce locomotion in chain-type self-reconfigurable robots. *Mechatronics, IEEE/ASME Trans. on*, 2002.

[Tosun *et al.*, 2015] 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*, 2015.

[Tosun *et al.*, 2016] Tarik Tosun, Jay Davey, Chao Liu, and Mark Yim. Design and characterization of the ep-face connector. 2016.

[Tosun *et al.*, 2017] Tarik Tosun, Daniel Edgar, Chao Liu, Thulani Tsabedze, and Mark Yim. Paintpots: Low cost, accurate, highly customizable potentiometers for position sensing. 2017.

[Wongpiromsarn *et al.*, 2010] 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.

[Yim *et al.*, 2007] 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.