An Integrated System for Perception-Driven Autonomy with Modular Robots

Jonathan Daudelin* Cornell University jd746@cornell.edu
Gangyuan Jing* Cornell University gj56@cornell.edu
Tarik Tosun* Univ. of Pennsylvania tarikt@grasp.upenn.edu
Mark Campbell Cornell University mc288@cornell.edu
Hadas Kress-Gazit Cornell University hadaskg@cornell.edu
Mark Yim Univ. of Pennsylvania yim@grasp.upenn.edu

***Abstract*—We present an integrated system architecture that allows self-reconfigurable modular robots to autonomously perform high-level tasks in unknown environments, using perception and reconfiguration to recognize and adapt to environment obstacles and constraints. By performing three hardware experiments, we demonstrate that our implementation of the system is capable of autonomously completing different object manipulation tasks in different environments without human changes to the system. The physical robot is composed of modules that support multiple robot configurations. An onboard 3D sensor provides information about the environment and informs exploration, reconfiguration decision making and feedback control. A centralized high-level mission planner uses information from the environment and the user-specified task description to autonomously compose low-level controllers to perform locomotion, reconfiguration, other behaviors. A novel, centralized, self-reconfiguration method is used to change robot configurations as needed. This is the first modular robot system that uses perception-driven reconfiguration to ~~intelligently~~ adapt to *a priori* unknown environments to perform complex tasks.**

I. Introduction

Modular self-reconfigurable robot (MSRR) systems are composed of a number of simple repeated robot elements (called *modules*) that connect together to form larger robotic structures. These robots can *self-reconfigure*, changing their shape (*i.e.* the connective arrangement of the modules) to meet the needs of the task at hand. The number of possible morphologies of these systems typically scales exponentially with the number of identical modules, making them highly adaptable for a wide variety of tasks.

Over the past three decades, dozens of MSRR systems have been built [21]. Existing literature provides ample evidence of MSRR systems reconfiguring and assuming interesting morphologies, as well as methods for programming, controlling, and simulating modular robots [23, 8, 20].

Some of the most challenging problem domains for robots are those in which they must autonomously complete a task in an unknown environment. Search-and-rescue problems fall into this category: we might want a robot to enter a collapsed building and locate survivors, without knowing what kind of terrain the robot needs to traverse before it actually enters the building.

The theoretical ability of modular robots to address these known-task-unknown-environment problems by reconfiguring has been frequently cited as an advantage [21], but has never been experimentally demonstrated. For the first time, we present a system architecture that integrates perception, high-level mission planning, and modular robot hardware, to solve these kinds of problems. In three hardware experiments, we show that our system allows a modular robot to autonomously explore, reconfigure and solve tasks in unknown environments. These demonstrations fill an experimental gap in the field, showing that reconfiguration can provide an advantage in practice, not just in theory.

However, our contribution is not just the experimental results, but also our system architecture, which provides a general framework to solve known-task-unknown-environment-problems with modular robots. We present the tools we used to achieve this capability (some novel, some existing), and discuss their roles in the system (Section III). Our hardware experiments with the SMORES-EP modular robot serve as a proof-of-concept, demonstrating how the system can address a variety of tasks in different unknown environments (Section IV). Finally, we discuss lessons learned during development, and comment on challenges and opportunities for others who might wish to adopt this architecture.

II. Related Work

Swarm-Bots is a MSRR system that has been applied in exploration [3] and collective manipulation [11] scenarios. Exploration is demonstrated in partially unknown environments, with some members of the swarm acting as "beacons" with known location. 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.

Swarm-Bots have demonstrated the capability to use self-assembly to solve low-level tasks, such as crossing a gap [6] or ascending a small hill [14]. In these scenarios, ground

*J. Daudelin, G. Jing, and T. Tosun contributed equally to this work.

proximity sensors and tilt sensors are used to trigger self-assembly. In our work, 3D map data is used to characterize the environment, and the system autonomously selects specific morphologies that are appropriate to the task and environment.

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 [4]. 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.

Autonomous 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 [23, 15, 12]. 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.

Recent work includes a system which integrates many low-level capabilities of a MSRR system in a design library, and accomplishes high-level user-specified tasks by synthesizing elements of the library into a reactive state-machine [8]. This system demonstrates autonomy with respect to task-related decision making, but is designed to operate in a fully known environment with external sensing.

III. System Architecture and Implementation

Our system enables modular robots to autonomously complete known tasks in unknown environments. Given a high-level specification of the desired task, the system enables the robot to explore the environment, gather information, and appropriately interact with the environment and objects in order to achieve the task objectives. The system is reactive: actions are selected based on the sensed environment, and different actions may be selected to complete the same task in different environments. Furthermore, it is the first system that will autonomously reconfigure the robot whenever it determines that a different configuration is needed to complete the task in the ~~current~~ environment.

The system architecture is shown in Figure 1. The physical robot consists of a set of **Robot Modules** (which can move, and connect to one another) and a single **Sensor Module** (which includes environment sensors and a computer). The **High-Level Planner** allows tasks to be specified at a high level using a formal language, which is compiled into a provably-correct ~~state machine~~ controller. **Active Perception**

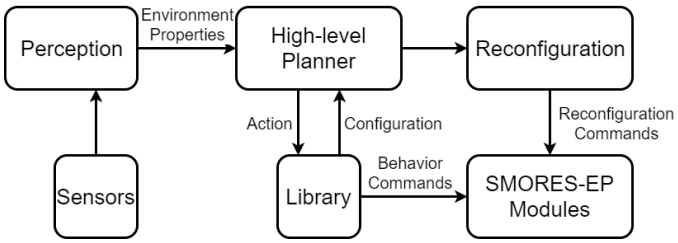


Fig. 1: System Overview Flowchart

components perform SLAM, plan waypoints for the robot to explore its environment, and identify features of the environment relevant to the high-level task. They also characterize the environment in terms of robot capabilities. Based on the task requirements and observed environment, the high-level planner selects an appropriate behavior for the robot from the **Library**. If the current configuration of the robot cannot execute the desired behavior, the high-level planner will command the **Reconfiguration subsystem** to transform the robot into a configuration that can execute the behavior.

The following sections discuss the role of each component within the general system architecture, and provide details of the implementation used in our experiments. Inter-process communication between the many software components in our implementation is provided by ROS[1].

A. Hardware

Our system implementation is built around the SMORES-EP robot [17], [18], but the system architecture could be used with any modular robot capable of self-reconfiguration. Each SMORES-EP module is the size of an *80mm cube* and has four actuated joints, including two wheels that can be used for differential drive on flat ground. The modules are equipped with electro-permanent magents that allow any face of one module to connect to any face of another, allowing the robot to self-reconfigure. The magnetic faces can also be used to attach to objects made of ferromagnetic materials (~~like~~ steel). EP magnets require very little energy to connect and disconnect, and no energy to maintain their holding force [17].

Each module has an onboard battery, microcontroller, and WiFi module to send and receive messages. In this work, clusters of SMORES modules were controlled by a central computer running a Python program that sends WiFi commands to control the four DoF and magnets of each module. Battery life is about one hour (depending on motor, magnet, and radio usage).

Our system architecture assumes there is a single *sensor module*, containing sensors and a computer, that is carried by a cluster of modules. The sensor module used in our experiments was designed to work ~~well~~ with SMORES-EP, and is shown in Figure 2. The body of the sensor module is a $90mm \times 70mm \times 70mm$ box with thin steel plates on its front and back that allow SMORES-EP modules to connect to it. Computation is

[1] http://www.ros.org

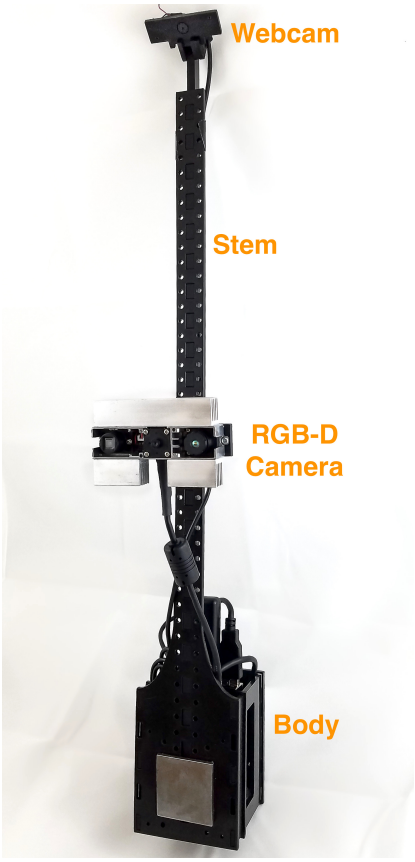


Fig. 2: Sensor Module with labelled components. UP board and battery are inside the body.

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 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 $0.75m \times 0.5m$ area in front of the brain module, and is used to track AprilTag [13] fiducials for reconfiguration. A $7.4v$, $2200mAh$ LiPo battery provides about one hour of running time.

B. High-Level Planner

In our architecture, the high-level planner subsystem provides a framework for users to specify tasks at a high level using a formal language, and acts as the main, centralized controller that directs robot motion and actions based on the given tasks. In order to do so, the high-level planner needs to treat each component of our system as a low-level atomic controller. At system level, the sensing components gather and process environment information for the high-level planner, which then take actions based on the given robot tasks by invoking appropriate low-level controllers.

The robot and environment status are abstracted as a set of Boolean propositions. For example, the robot action **pickUp** is `True` if the robot is currently picking up an object (and `False` otherwise) and the environment proposition **pinkItem** is `True` if the robot is currently sensing a pink item (and `False` otherwise). A wide range of reactive robotic tasks can be specified in terms of these propositions. By using a library of robot configurations and behaviors as well as environment characterization tools, these high-level statements are mapped to low-level sensing programs and robot controllers. The user specifies high-level robot actions in terms of behavior properties from the library. For example, our system can choose to do a pick up action by executing any behavior from the library which has the behavior property **pickUp**, and which also satisfies the current environment characteristics. If the current robot configuration cannot execute an appropriate behavior, the robot will reconfigure to a different configuration that can. In this way, the system autonomously chooses to implement **pickUp** appropriately in response to the sensed environment. Propositions related to the state of the environment are evaluated using our perception and environment characterization tools. For example **pinkItem** is `True` if the robot is currently sensing a pink item, which is determined by consulting the color tracking functions.

Our implementation employs an existing framework called LTLMoP to automatically generate robot controllers from user-specified high-level instructions using formal methods [5, 9]. Given a specification, LTLMoP will synthesize a robot controller that satisfies the given task (if one exists). The controller is synthesized in the form of a finite state automaton. Each state specifies a set of high-level robot actions that need to be performed, and transitions between states are specified with a set of environment propositions. Execution of the high-level controller begins at the predefined initial state in the finite state automaton. In each iteration, the values of all environment propositions are determined by calling the corresponding sensing program. Then, we determine the next state in the finite state machine by taking the transition that matches the current value of all environment propositions. In the next state, the specified high level behavior properties are mapped to a behavior from the design library which satisfies both the behavior properties and current environment type.

C. Perception and Planning for Information

The perception and exploration subsystem interprets sensor data to explore and understand the environment and inform robot behavior and adaptation. The architecture of this subsystem requires algorithms to perform SLAM, provide navigation goals for exploration, recognize objects of interest, and characterize the environment in terms of robot configuration abilities.

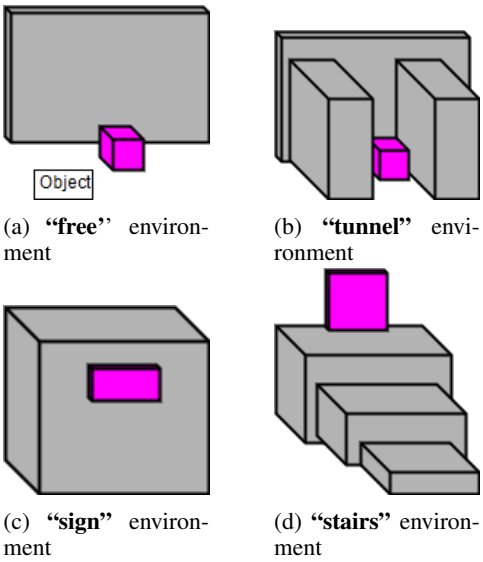
In our implementation, mapping and robot pose are provided by ~~an~~ RGB-D SLAM software package ~~called~~ RTAB-MAP[10]. A 3D map of the environment is incrementally built and stored in an efficient octree-based volumetric map using Octomap[7]. To drive exploration of the unknown environment, a Next Best View planner [1] is used. Upon request, it provides waypoints to the high-level planner that maximize

expected information collection of unknown regions of the environment. To identify objects of interest in the task, color detection and tracking is used. Colored objects are recognized using CMVision[2], and tracked in 3D using depth information from the onboard 3D sensor[3]. Although we implement object recognition by color, more sophisticated methods could be included in the same system architecture.

To enable the high-level planner to choose appropriate configurations for a task, our framework requires an environment characterization algorithm. This algorithm reasons about regions of interest in the environment to classify them in terms of the abilities of robot configurations. For our proof-of-concept, we implemented an algorithm that can differentiate between four environment types relevant to object manipulation, shown in Figure 3.

When an object is recognized in the environment, the 3D information about its surroundings is evaluated. An occupancy grid is created around the object location, and all grid cells within a robot-radius of obstacles are denoted unreachable (as shown in Figure 4). The closest reachable point to the object within 20^o of the robot's line of sight to the object is selected. If the distance from this point to the object is greater than a threshold value and the object is on the ground, the environment is characterized as a "tunnel", likely requiring a configuration that is narrow enough to move between the surrounding obstacles to retrieve the object. If above the ground, the environment is characterized as a "stairs" environment, requiring a robot that can climb to reach the object. If the closest reachable point is under the threshold value, the environment is classified as either a "free" or "sign" environment, depending on the height of the colored object.

In addition to characterizing the environment, the algorithm also selects a waypoint based on the characterization for the robot to position itself for reconfiguration and performing the object manipulation task.



(a) **"free"** environment (b) **"tunnel"** environment (c) **"sign"** environment (d) **"stairs"** environment

Fig. 3: Environment characterization types.

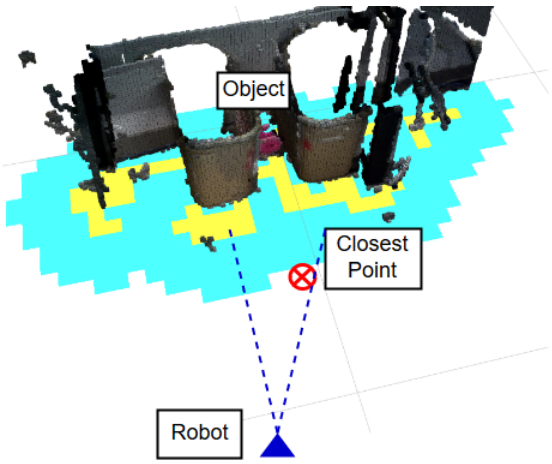


Fig. 4: An example of a **tunnel** environment characterization. Yellow grid cells are occupied, light blue cells are unreachable resulting from bloating obstacles.

D. Library of Configurations and Behaviors

In our architecture, the full set of capabilities of the modular robot is encoded in a library of robot configurations and behaviors. Each entry is labeled with attributes that describe what it does, as well as the environment conditions under which it may be used. This allows the high-level planner to match environment characterizations from the perception subsystem with the configuration and behavior that can perform the task in the current environment.

Our implementation relies on a framework first presented in [8], which we summarize here. In this framework, a library entry is defined as $l = (C, B_C, P_e, P_b)$ where:

- C is the robot *configuration*, specified by the connected structure of the modules.
- B_C is a *behavior* that C can perform. A behavior is a controller that commands the robot to perform a specific ~~useful~~ action.
- P_b is a set of *behavior properties* that describes what B_C does.
- T_e is a set of *environment types* that describe the environments in which this library entry is suitable.

When we specify tasks at the high level, we use behavior properties P_b to describe a desired robot action without explicitly specifying a configuration or behavior. Environment types T_e specify the conditions under which a behavior can be used.

We implemented a library containing 10 entries for four different configurations, listed in Table I. This library was created to enables the robot to explore and interact with the environment and objects, but the library can be expanded to include more entries for other environments and tasks.

[2]CMVision: http://www.cs.cmu.edu/~jbruce/cmvision/

[3]Lucas Coelho Figueiredo: https://github.com/lucascoelho91/ballFollower

Configuration	Behavior properties	Environment Types
Tank	**pickUp**	"free"
Tank	**drop**	"free"
Tank	**drive**	"free"
Proboscis	**pickUp**	"tunnel" or "free"
Proboscis	**drop**	"tunnel" or "free"
Proboscis	**placeStamp**	"sign"
Scorpion	**drive**	"free"
Snake	**climbUp**	"stairs"
Snake	**climbDown**	"stairs"
Snake	**drop**	"stairs" or "free"

TABLE I: A library of robot behaviors

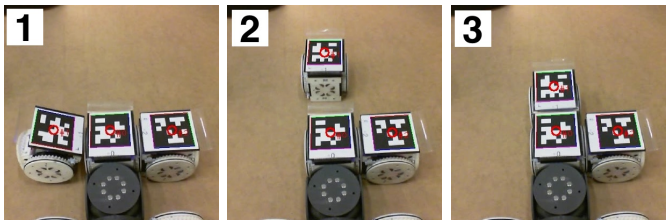


Fig. 5: Module movement during reconfiguration. (1) Left module detaches from cluster. (2) Module drives to waypoint, and spins to establish ~~perfect~~ heading. (2) Module drives straight in to dock point, and reattaches.

Note that each library entry consists of a configuration with its available behaviors subject to restrictions in environment conditions. To characterize the environment in terms of these restrictions, we use the perception algorithm described in Section III-C.

To provide an illustrative example, we discuss two configurations and their capabilities in detail. The "Tank" configuration shown in Figure 7e is capable of picking up and dropping objects in a "free" environment. In addition, the "Tank" configuration can locomote on flat terrain. It uses a parametric differential drive behavior to convert a desired velocity vector into motor commands (**drive** in Table I).

The "Proboscis" configuration shown in Figure 7d has a long arm in front, and is suitable for reaching between obstacles in a narrow "tunnel" environment to grasp objects or reaching up in a "sign" environment to drop items. However, the locomotion ability of this configuration is limited to forward/backward motion, making it unsuitable for general navigation.

This library-based framework allows us to specify tasks for modular robots at a ~~high~~ level, by associating abstract robot actions with robust low-level controllers that execute appropriate robot behaviors based on the sensed environment. For example, if a task specification requires the robot to execute a **pickUp** behavior, our system could select either the Tank or Proboscis configurations to perform the action, since both have behaviors property **pickUp**. To choose the appropriate configuration to use, the robot uses perception to characterize its environment in terms of the environment types T_e associated with each configuration. If the environment is of type "tunnel", only the Proboscis configuration can be used, and the robot will take reconfigure to form the Proboscis if it is not already in that configuration. Since self-reconfiguration is time-consuming, decisions are biased toward behaviors associated with the current configuration whenever possible, so that the robot only reconfigures when it is necessary to complete a task.

E. Reconfiguration

When the high-level planner decides to use a new configuration during a task, the robot must reconfigure. Our system architecture allows any method for reconfiguration, provided that it requires no external sensing.

SMORES-EP is capable of all three classes of modular self-reconfiguration (chain, lattice, and mobile reconfiguration) [2, 22]. We have implemented tools for mobile reconfiguration with SMORES-EP, taking advantage of the fact that individual modules can drive on flat surfaces as described in Section III-A.

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

Given an initial configuration and a goal configuration, our reconfiguration controller commands a set of modules to disconnect, move and reconnect in order to form the new topology of the goal configuration. The robot first takes actions to establish the conditions needed for reconfiguration. It begins by confirming that the reconfiguration zone is a flat surface free of obstacles (other than the modules themselves). If the robot is carrying an object, it drops the object outside of the reconfiguration zone. It 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 5. Currently, reconfiguration plans are created ~~by hand~~. In the future, existing assembly planning algorithms ([19, 16]) could be adapted to generate reconfiguration plans automatically. Because the reconfiguration zone is free of obstacles, collision-free paths can be pre-computed and stored 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 begin performing desired behaviors, and if necessary, the robot picks up any objects it dropped. ~~This completes the reconfiguration process.~~

IV. Experiment Results

As mentioned earlier, our work presents the first MSRR system that uses perception and reconfiguration to autonomously perform high-level tasks in unknown environments. We tested our system implementation on multiple, real-world tasks. These experiments accomplish two goals: First, they demonstrate our system's ability to autonomously adapt to multiple unknown environments and to complete different tasks. Second, they provide new observations of challenges with fully autonomous MSRR systems. These observations provide insight on what can be improved in autonomous MSRR systems to make them more robust and versatile.

Three experiments were run, each time varying the environment and the type of task to be performed. Each experiment required the robot to explore an unknown environment to search for and identify one or more objects of interest, and then complete some manipulation task with those objects.

A. Experiment I

The first experiment was designed to involve a long and complex task in order to fully demonstrate the robot's exploration, reconfiguration, and manipulation abilities. For this experiment, the robot is tasked with cleaning a messy graduate student office. It is directed to explore the unknown office environment and search for any brightly-colored metal garbage that can be recycled. Whenever it finds brightly colored metal objects, it should pick them up and bring them to a designated drop-off zone for recycling, which is marked with a blue square on the wall.

The high-level task consisted of three different types of sub-tasks: explore the environment, find and retrieve all metal objects, and deliver each to the drop-off zone. Figure 6 shows the environment layout used in the experiment, designed to represent a messy office. Two colored objects were placed in the environment: a green soda can was placed in the open (a "free" environment type), and a pink spool of wire was positioned in a narrow gap between two trash cans (a "tunnel" environment). Obstacles, environment types, and locations of the objects and drop-off zone were unknown to the robot *a priori*. Note that the object colors (pink and green) have no influence on environment characterization: in other words, the robot does not know that the pink object is in a "tunnel" before the experiment starts.

Five SMORES-EP modules, one Sensor Module, and three passive cubes composed the robot for performing the task. The robot grasped objects using modules' electro-permanent magnets to attach to steel washers on the objects. During the experiment, the high-level planner used the library of two configurations and five behaviors described in Section III-D.

Figure 7 shows snapshots from the experiment run. A video of the entire experiment is available as an attachment to this paper. The robot starts in the "Tank" configuration. The starting location prevented the robot from seeing the objects initially, forcing it to explore the environment to search for them. After a period of exploration, the robot locates the pink object. The characterization algorithm correctly classified the surrounding environment as a "tunnel" type, and accordingly the robot navigated in front of the object and reconfigured to the "Proboscis" configuration. The "Proboscis" then used its long arm to reach into the tunnel and pull the object out into the open.

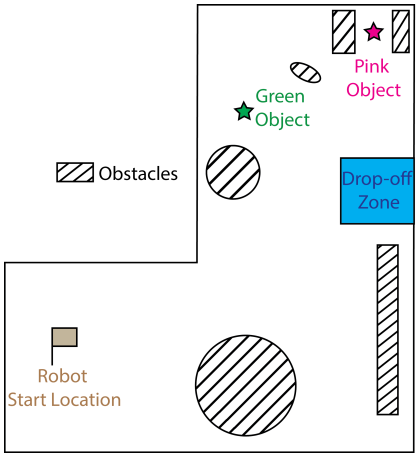


Fig. 6: Diagram of Experiment I environment

The "Proboscis" can only drive forwards and backwards (it cannot steer), so the robot concluded it needed to reconfigure back to the "Tank" to drive the object to the drop-off zone. To do so, the robot dropped the object, reconfigured, and picked the object back up. The "Tank" then navigated to and dropped off the object at the drop-off zone, which was previously located during exploration and recorded in the global map built by the SLAM algorithm.

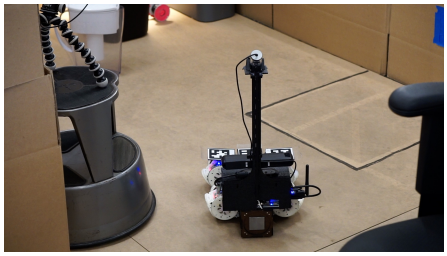
Once done, the robot navigated directly to the green object, which it discovered while dropping off the pink object. The robot correctly determined that the green object was in a "free" environment, and picked it up without reconfiguring. The robot finished the experiment by also delivering the green object to the drop-off zone. Figure 8 shows a volumetric map of the environment generated as the robot explored and delivered objects. The robot successfully completed all tasks in the experiment in about 26 minutes.

Note that at one point in the video a ~~human~~ reaches onto the field to dislodge the green object from a crack in the floor. This was due to a defect in the experimental setup (which is not supposed to have a crack in the floor) and not an error in the robot's performance.

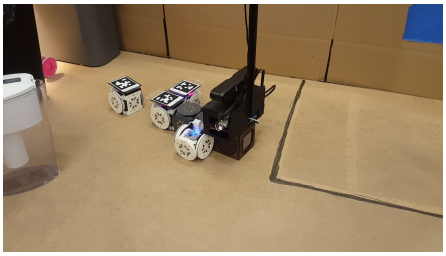
B. Experiment II

The second and third experiments were designed to demonstrate the breadth of our system's adaptive capabilities. As such, these two experiments included the same high-level task specification that is different from the task in Experiment I. However, differing environments between Experiment II and Experiment III force the robot to adapt differently while performing the task. For Experiment II, the robot is given a circuit that needs to be shipped, and is tasked with finding the

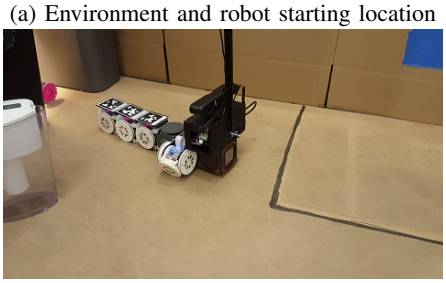
(a) Environment and robot starting location



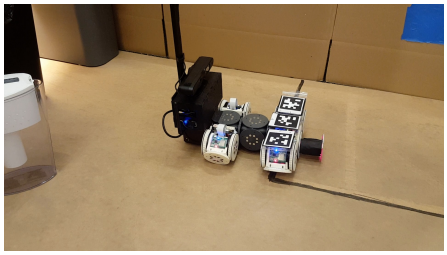
(b) Exploring while searching for objects



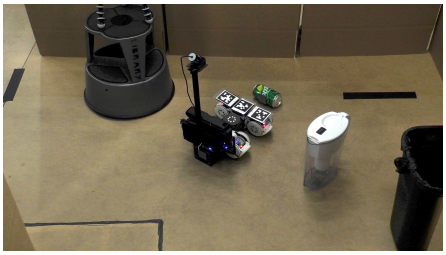
(c) Reconfiguring to retrieve pink object



(d) Retrieving pink object



(e) Depositing an object in the drop-off zone



(f) Retrieving green object

Fig. 7: Phases of Experiment I.

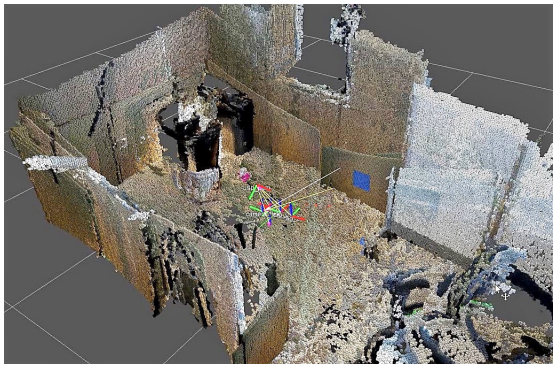
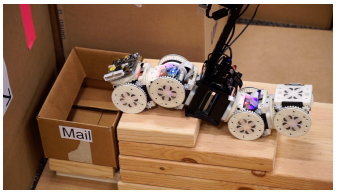
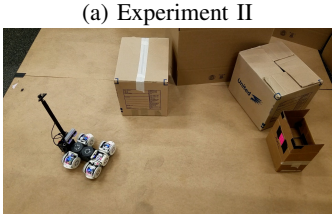


Fig. 8: Volumetric map of environment built by visual SLAM

(a) Experiment II



(b) Successful circuit delivery



(c) Experiment III

(d) Successful stamp placement

Fig. 9: Experiments II and III.

shipping bin in which to place the circuit. A pink sign above the bin denotes the bin's location. The high-level task includes sub-tasks of exploring the unknown environment, locating and characterizing the environment surrounding the mail bin, and delivering the package to the bin.

Figure 9a shows the environment setup for Experiment II. ~~The full performance of the experiment is included in the video attachment to this paper~~. Note that the mail bin has stairs in front of it, which the robot must climb in order to drop the circuit into the bin successfully. The robot begins the experiment in the "Scorpion" configuration. Shortly after beginning exploration of the environment, the robot observes and recognizes the mailbox, and characterizes the environment surrounding it as "stairs". Based on this characterization, it determines that the "Snake" configuration is needed 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. It 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.

C. Experiment III

For the final experiment, the robot is given a stamp and is tasked with placing it on a package that needs to be shipped. The high-level task has the same specification as in Experiment II, which for its environment includes the following sub-tasks: explore the unknown environment, find the package with stamp location denoted by a pink area, and place the stamp on the package. The robot was started in the "Tank" configuration used in the first experiment.

Figure 9c shows the environment setup for Experiment III. ~~Please see the video attachment for the full performance of this experiment~~. The robot begins exploring the unknown environment, and quickly observes the package to be stamped. It characterizes the environment as the "sign" environment using the 3D map created of the environment, and determines that the "Proboscis" configuration is needed with the "placeStamp"

behavior to place the stamp on the pink goal area. The robot then navigates to a position directly in front of the package, reconfigures appropriately, and executes the gait to place the stamp correctly on the package to complete the task.

V. Discussion

We evaluate our system with respect to four criteria: *autonomy*, *capability*, *reconfigurability*, and *robustness*. In all four areas, we evaluate both the system *architecture*, which is more general, and the system *implementation*, which was used in our experiments.

A. Autonomy

How independent is the robot? To what degree can it operate without external ~~external~~ computation, sensing, power, or other help?

Autonomy was a major goal of this project, and an area in which the system architecture and implementation are both successful. Like most modular robots, the limited size and strength of SMORES-EP presented challenges for carrying sensors and computing equipment. Feasibility of implementation influenced our choice of system architecture: in contrast to the distributed sensing approaches used in prior modular work, our decision to use a centralized sensing architecture allowed the robot to carry a single, powerful sensor (the RGB-D camera) enabling sophisticated capabilities like Visual SLAM and environment characterization. As a result, our implementation has more sensor-based autonomy than any previous modular robot.

Our implementation relies in part on external computation and wireless network access, but this is not a fundamental system requirement. In fact, in our implementation the majority of heavy computation (mapping, navigation, video processing) was done onboard on the sensor module, with a few processes (high-level planner, reconfiguration planner, and next-best-view planner) run offboard, primarily for development convenience. In the future we believe everything could be run onboard with only minor adjustments.

B. Capability

How capable is the implementation, in terms of solving real tasks? How general is the system architecture (does it place any limitations on the kinds of tasks this system can address)?

Broadly speaking, our system architecture provides the capability to *specify high-level tasks*, *gather information* about an unknown environment, *explore the environment*, and *interact with objects in the environment*. In theory (given appropriate implementation), the architecture is capable of addressing any problem consisting of these tasks.

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, Experiments II and III are the same *task:* deliver an object at a point of interest. However, after characterizing the environment, the system determines that vastly different behaviors are required to complete the task, and in both cases, that reconfiguration is needed before the appropriate behavior can be performed. Similarly, in Experiment 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 is solved in a simple way (green object), and the more difficult problem is solved in a more sophisticated way (pink object).

The system's *information-gathering* capability is largely dependent on the available sensing and computing hardware, and the algorithms used to process sensor data. We opted to use simple techniques for navigation, object recognition, and differentiating between environments, which were suitable for our proof-of-concept experiments.

To deploy a similar system in real life, more sophisticated techniques would need to be used. Considering the ongoing trend towards high-performance sensors and computers at small sizes, we would expect that implementing these sophisticated techniques onboard the sensor module will become easier over time.

The ability of the robot to *move* in the environment is also largely determined by the hardware. The SMORES-EP designs we used for exploration move slowly, and can only drive over smooth, flat ground. In a demanding real-world application like search-and-rescue, the hardware system would likely need to be able to move more quickly.

In theory, the system architecture places no fundamental limitations on the kinds of gaits or behaviors used to locomote in the environment. However, our architecture does require a suitable *motion model* and *path planner* for any behavior used for locomotion. In practice, this means that while a modular robot may be capable of a large number of creative or unusual gaits for locomotion, those that conform to standard, well-understood motion models (like differential or holonomic drive) require much less effort to implement, and often end up being the most useful for actually accomplishing tasks.

The three scenarios in our experiments were selected to 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, and climbing over rough terrain, and manipulating objects. This broad range of functionality is impressive for such a small robot, and is only accessible to SMORES-EP by reconfiguring between different morphologies.

In general, the range of ways in which the robot may interact with its environment is determined by the contents of the design library, which would ideally contain a large number of configurations and behaviors suitable for a diverse set of tasks. That fact that our architecture relies upon a discrete representation of behaviors could mean that a real-world implementation would require a very large library, which presents some challenges. To precisely describe capabilities and limits of each behavior, a large set of attributes would be needed when labeling library entries. As a result, users will need to understand all these attributes and use accurate ones when specifying the robot tasks. When searching the library

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 II: Reasons for experiment failure.

for behaviors with user specified constraints, our method scales linearly with respect to the size of the library.

C. Reconfigurability

How good is the system at reconfiguring? How useful is reconfiguration?

In general, our system architecture is not tied to a specific method of reconfiguration. However, the centralized sensing architecture drove our decision to implement reconfiguration in a centralized way as well. High-fidelity centralized sensing during reconfiguration, provided by AprilTags, allowed our implementation to reconfigure quickly and reliably enough to undergo multiple transformations in the process of completing a task. In contrast, for previous modular systems, the high cost of reconfiguration (in terms of time and risk of failure) would make them difficult to deploy as a practical tool for accomplishing tasks. This is significant: for the first time, we have demonstrated that the benefits of reconfiguring to complete a task can outweigh the costs in a realistic setting.

Our centralized implementation of reconfiguration has limitations. Since modules can only move in 2D within the camera view, some reconfigurations are not possible. For example, the "Tank" cannot reconfigure into the "Snake" using our implementation, because the camera cannot see the back wheels of the "Tank." Consequently, if the robot had started out as the "Tank" at the beginning of Task 3, it would not have been able to complete its task.

In the context of the entire system, we believe the benefits of our centralized reconfiguration implementation outweigh its limitations. In comparison with previous modular systems, we have sacrificed some reconfiguration *flexibility* to achieve more *capability* and *autonomy* in the total system.

D. Robustness

How well does the system respond to disturbances or violated assumptions?

Our implementation is vulnerable to small errors - if things go wrong in the course of completing a task, the entire task will likely fail. Table II shows the causes of failure for 24 attempts of Experiment 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 on. 42% of failures were due to hardware errors, and 38% were due to failures in low-level software (object recognition, navigation, environment characterization). These components were designed for research, not real-world deployment, and could be engineered to perform much more reliably in a production setting.

The simulator served as a valuable prototyping tool, but all behaviors required significant fine-tuning in hardware before they would run reliably. Open-loop behaviors like stair-climbing and reaching up to place the stamp are fairly inflexible with respect to environment conditions, and also vulnerable to hardware errors like poorly calibrated encoders. Behaviors like exploration and reconfiguration use the sensor modules to close the loop, and are significantly less vulnerable to these kinds of errors. However, these behaviors required much more up-front development effort.

Lack of robustness is a weakness of the system architecture. The high-level planner assumes all underlying components are reliable and robust, so if any low-level component fails, the high-level planner might behave unexpected and the entire task may fail. In the future, we will explore high-level strategies for error recovery: for example, recognizing that a module has failed and adapting the approach so that task will not fail entirely.

E. Future Work and Conclusion

The abilities of the system and current MSRR technology could be extended in future work. Our experiment utilizes a library of four configurations, and our perception system recognizes four environment types (stairs, tunnel, sign, and free). To further develop the ability of MSRR systems to adapt to their environment, a larger library of configurations and gait controllers could be designed, along with a more powerful environment characterization algorithm.

The design of a more diverse library of configurations can be greatly aided by improving strength and versatility in module hardware. Real-world applications, such as search-and-rescue, require robots to climb over obstacles and navigate rough terrain. This requires modules with strong locomotion power and the ability to hold large payloads without breaking inter-module connections.

It is important to consider how the presented system could be adapted to work with other modular robot hardware. Our system relies on a 3D sensor for performing RGB-D SLAM, sufficient onboard processing power for perception and control algorithms, and a robust modular hardware system with a versatile self-reconfiguration ability that doesn't rely on external sensors. As long as another MSRR system provides these features and its own hardware-specific and reconfiguration controllers, our system can be integrated into it for performing high-level tasks.

To conclude, this paper presents the first MSRR system that uses perception of an unknown environment to reactively perform complex high-level tasks using intelligent reconfiguration. Components of this system include novel controller synthesis, environment characterization, and self-reconfiguration methods. The demonstration of this novel capability is crucial to the success of modular robots as a technology, and takes a step toward the application of modular robots to tasks in the real world.

References

[1] 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.

[2] Jay Davey, Ngai Kwok, and Mark Yim. Emulating self-reconfigurable robots-design of the SMORES system. In *IROS*, pages 4464–4469, 2012.

[3] Marco Dorigo, Elio Tuci, et al. The SWARM-BOTS Project. *LNCS*, 3342:31–44, 2005.

[4] Marco Dorigo, Dario Floreano, et al. Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine*, 20 (4):60–71, 2013.

[5] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*, pages 1988–1993, 2010.

[6] Roderich Groß, Michael Bonani, Francesco Mondada, and Marco Dorigo. Autonomous self-assembly in swarm-bots. *IEEE transactions on robotics*, 22(6):1115–1130, 2006.

[7] Armin Hornung, Kai M. Wurm, et al. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[8] Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. An End-To-End System for Accomplishing Tasks with Modular Robots. *Robotics: Science and Systems XII*, 2016.

[9] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Robotics*, 25(6):1370–1381, 2009.

[10] M. Labbe and F. Michaud. Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, Sept 2014.

[11] Francesco Mondada, Luca Maria Gambardella, et al. The cooperation of swarm-bots: Physical interactions in collective robotics. *IEEE Robotics and Automation Magazine*, 12(2):21–28, 2005.

[12] 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.

[13] 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.

[14] Rehan OGrady, 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.

[15] 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.

[16] 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.

[17] Tarik Tosun, Jay Davey, Chao Liu, and Mark Yim. Design and characterization of the ep-face connector. In *IROS*. IEEE, 2016.

[18] Tarik Tosun, Daniel Edgar, et al. Paintpots: Low cost, accurate, highly customizable potentiometers for position sensing. In *ICRA*. IEEE, 2017.

[19] 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.

[20] M Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Stanford University, 1994.

[21] M Yim, WM Shen, and Behnam Salemi. Modular self-reconfigurable robot systems: Challenges and Opportunities for the Future. *IEEE Robotics & Automation Magazine*, (March), 2007.

[22] Mark Yim, Kimon Roufas, et al. Modular reconfigurable robots in space applications. *Autonomous Robots*, 14(2): 225–237, 2003.

[23] Mark Yim, Babak Shirmohammadi, et al. Towards Robotic Self-reassembly After Explosion. In *IROS*, 2007.