An Integrated System for Perception-Driven Autonomy with Modular Robots

Jonathan Daudelin*, *Member, IEEE,* Gangyuan Jing*, *Member, IEEE,* Tarik Tosun*, *Member, IEEE,* Mark Yim, *Member, IEEE,* Hadas Kress-Gazit, *Member, IEEE,* and Mark Campbell, *Member, IEEE*

***Abstract*—The theoretical ability of modular robots to reconfigure in response to *a priori* unknown environments has frequently been cited as an advantage, but has never been experimentally demonstrated. For the first time, we present a system that integrates perception, high-level mission planning, and modular robot hardware, ~~providing~~ allowing a modular robot to autonomously reconfigure in response to its perceived environment in order to complete a task. We validate our system in three hardware experiments, where we demonstrate a modular robot autonomously exploring, reconfiguring, and manipulating objects to complete high-level tasks in unknown environments.**

We present the details of our system architecture and implementation, which provides a general framework allowing modular robots to solve tasks in unknown environments using autonomous, reactive reconfiguration. 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.

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 [1]. Existing literature provides ample evidence of MSRR systems reconfiguring and assuming interesting morphologies, as well as methods for programming, controlling, and simulating modular robots [2, 3, 4].

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 reconfigure in response to these known-task-unknown-environment problems has been frequently cited as an advantage [1], but has never been experimentally demonstrated. For the first time, we present a system that integrates perception, high-level mission planning, and modular robot hardware, allowing a modular robot to autonomously reconfigure in response to its perceived environment in order to complete a task. We validate our system in three hardware experiments. Based on a high-level task specification, we show how our system allows the robot to autonomously explore, decide when and how to reconfigure, and manipulate objects to complete its task. 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 tasks in unknown environments using autonomous, reactive reconfiguration. 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

The Millibot system demonstrated the ability to map a partially unknown environment when operating as a swarm [5]. The autonomy of the Millibot swarm is limited: a human operator makes all high-level decisions, and is responsible for navigation using a GUI. Certain members of the swarm are designated as "beacons", and have known locations, making the environment only partially unknown.

Swarm-Bots is a MSRR system that has been applied in exploration [6] and collective manipulation [7] 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.

J.Daudelin, G. Jing, M. Campbell, and H. Kress-Gazit are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, 14850.

T. Tosun and M. Yim are with the Mechanical Engineering and Applied Mechanics Department, University of Pennsylvania, Philadelphia, PA, 19104.

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

Swarm-Bots have demonstrated the capability to use self-assembly to solve low-level tasks, such as crossing a gap [8] or ascending a small hill [9]. In these scenarios, ground 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 [10]. 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 [2, 11, 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.

Other work has focused on reconfiguration planning, but not autonomous reconfiguration. Paulos et al. present a system in which self-reconfigurable modular boats self-assemble into prescribed floating structures, such as a bridge [13]. 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.

MSRR systems have demonstrated the ability to accomplish low-level tasks such as various modes of locomotion [4]. 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 [3]. This system demonstrates autonomy with respect to task-related decision making, but is designed to operate in a fully known environment with external sensing.

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 experiments, we demonstrate that autonomous self-reconfiguration allows our system to adapt to the environment to complete complex tasks.

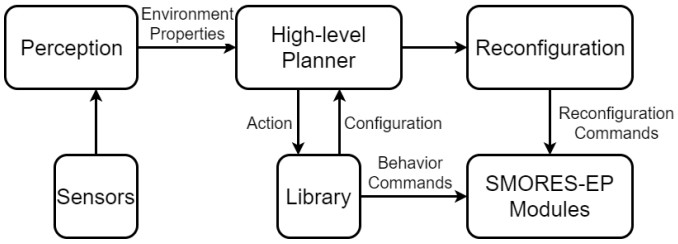


Fig. 1: System Overview Flowchart

III. System Architecture and Implementation

Given a high-level specification of the desired task, our 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. While other systems have demonstrated reconfiguration, ours is the first to do so in response to perceived environments in order to complete tasks.

For example, consider the following mail delivery task: Explore the environment, locate a mailbox, and drop an object in the mailbox. To complete this task, the robot explores and locates the mailbox. It then recognizes that in order to reach the mailbox, it needs to ascend a flight of stairs. In response, it chooses to reconfigure into a stair-climbing configuration, which it then uses to climb the stairs and deliver the object. Afterwards, it descends and reconfigures back into a driving configuration, to return to home base.

Figure 1 illustrates the system architecture. 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 controller. **Active Perception** components perform Simultaneous Localization and Mapping (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 the Robot Operating System (ROS)[1].

A. Hardware

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

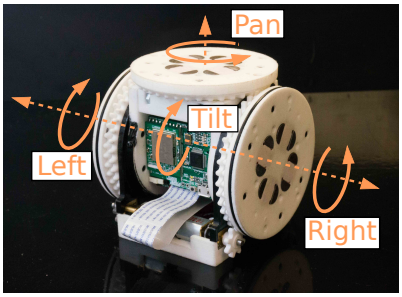


Fig. 2: SMORES-EP module

1) SMORES-EP Modular Robot: Our system implementation is built around the SMORES-EP robot [14], [15], 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 magnets 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 (e.g. steel). The EP magnets require very little energy to connect and disconnect, and no energy to maintain their attachment force of 90N [14].

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. Wireless networking was provided by a standard off-the-shelf router, with a range of about 100 feet, and commands to a single module can be received at a rate of about 20hz. Battery life is about one hour (depending on motor, magnet, and radio usage).

The hardware system also includes passive cubes that modules can connect to, providing lightweight passive structure in SMORES-EP configurations. Cubes have the same 80mm form factor as modules.

2) Sensor Module: 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 with SMORES-EP, and is shown in Figure 3. 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 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 sensor module, and is used to track AprilTag [16] fiducials for reconfiguration. A 7.4V, 2200mAh LiPo battery provides about one hour of running time.

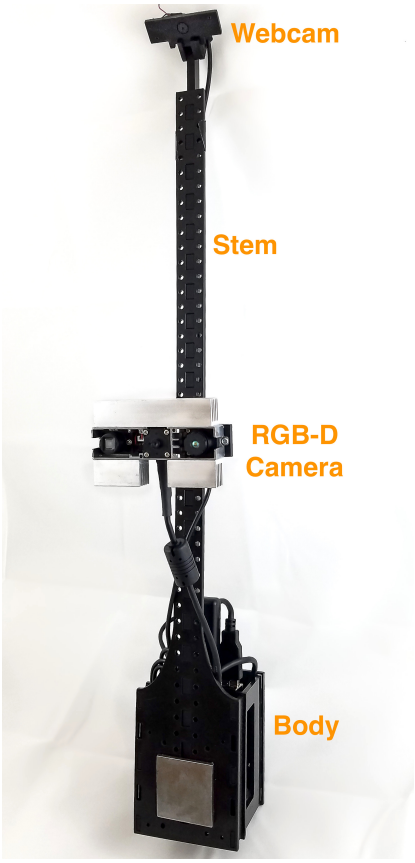


Fig. 3: Sensor Module with labelled components. UP board and battery are inside the body.

B. 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 reconfiguration. Recall in the example mail delivery task, the robot needed to explore and navigate its environment, visually locate the mailbox, and characterize the environment near the mailbox as "stairs." In our system architecture, the perception and exploration subsystem is responsible for performing SLAM, providing navigation goals for exploration, and recognizing objects and regions of interest. It is also responsible for characterizing the environment in terms of robot configurations abilities, allowing the high-level planner to make decisions regarding reconfiguration.

In our implementation, the RGB-D SLAM software package RTAB-MAP[17] provides mapping and robot pose. The system incrementally builds a 3D map of the environment and stores it in an efficient octree-based volumetric map using Octomap[18]. The Next Best View algorithm by Daudelin et. al.[19] enables the system to explore unknown environments. It uses 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). It also estimates the amount of information

(in an entropy-reduction sense) that would be gained from a sensor measurement taken at that viewpoint. The system provides the volumetric map of the environment and the reachable space of sensor viewpoints in that map to the algorithm. The algorithm then returns Next Best View waypoints to the high-level planner upon request. 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 it finds the dropoff zone.

To identify objects of interest in the task (such as the dropoff zone), our system uses color detection and tracking. It recognizes colored objects using CMVision[2], and tracks them in 3D[3] using depth information from the onboard RGB-D sensor. In our mail delivery example, the mailbox could be marked with a pink sign, allowing the robot to recognize it in the environment. Although we implement object recognition by color, more sophisticated methods could be included under 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 4.

When the system recognizes an object in the environment, it 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 5). It then selects the closest reachable point to the object within 20^o 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 system characterizes the environment as a "tunnel". If above the ground, it characterizes 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.

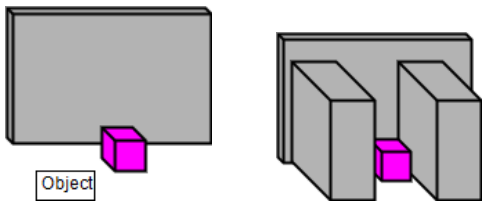
Based on the environment characterization and target location, the algorithm also returns a waypoint for the robot to position itself to perform its task (or to reconfigure, if necessary). In our mail delivery example, the environment characterization algorithm directs the robot to drive to a waypoint at the base of the stairs, which is the best place for it to reconfigure and begin climbing the stairs.

C. Library of Configurations and Behaviors

In our architecture, we encode the full set of capabilities of the modular robot, such as driving and picking up items, in a library of robot configurations and behaviors. Specifically, we label each entry 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 configurations and behaviors that can perform the task in the current environment. In the case of our mail delivery example, when the environment characterization algorithm reports that the mailbox is located in a "stairs"-type environment, the high-level planner queries the library for configurations that can climb stairs. Since the library indicates that current configuration is only capable of driving on flat ground, the high-level planner opts to reconfigure to the stair-climber configuration, and executes its **climbUp** behavior.

Our implementation relies on a framework first presented in [3], 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 can perform. A behavior is a controller that commands the robot to perform a specific



(a) **"free"** environment

(b) **"tunnel"** environment

(c) **"high"** environment

(d) **"stairs"** environment

Fig. 4: Environment characterization types.

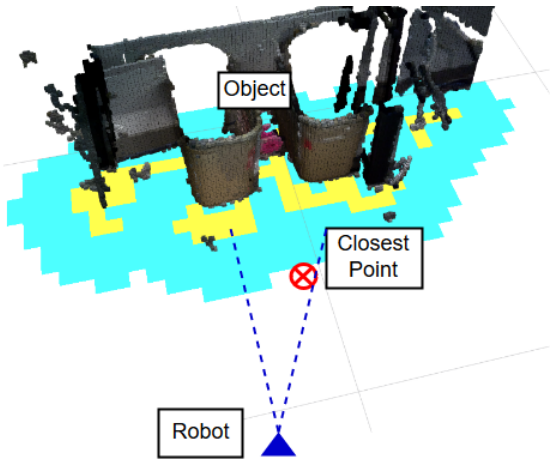


Fig. 5: An example of a **tunnel** environment characterization. Yellow grid cells are occupied, light blue cells are unreachable resulting from bloating obstacles.

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

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

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.

To create robot configurations and behaviors, we utilize ~~a computer simulation program called~~ VSPARC (Verification, Simulation, Programming And Robot Construction [4]) presented in [3]. VSPARC allows users to design, simulate and test configurations and behaviors for the SMORES-EP robot system. Our system can then use the behavior designs to control the physical SMORES-EP robot to perform various actions.

In [3], all robot behaviors are *static* behaviors. That is, once users create a behavior in VSPARC, joint values for each module are fixed and cannot be modified during behavior execution. Static behaviors, such as a car with a fixed turning radius, do not provide enough maneuverability for the robot to navigate around unknown environment. In this work, we expand the type of behaviors in the library by using ~~the~~ *parametric* behavior, which ~~is~~ first introduced in [20]. Parametric behaviors have joint values that can be altered during run-time, and therefore allow a wider range of motions. Recall the mail delivery task, based on the sensed environment, the perception and exploration subsystem (Section III-B) can ~~a~~ generate collision-free path, which is used to calculate real-time velocity for the robot to follow the path. Our system then converts the robot velocity to joint values in parametric behaviors during run-time.

The library introduced in [3] has 57 robot configurations and 97 behaviors. Here we list 10 entries for four different configurations in Table I that are used in this work. 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-B.

To provide an illustrative example, we discuss two configurations and their capabilities in detail. The "Car" configuration shown in Figure 8e is capable of picking up and dropping objects in a "free" environment. In addition, the "Car" 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 8d 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 "high" environment to drop items. However, the locomotion ability of this configuration is limited to forward/backward motion, making it unsuitable for general navigation.

[4]www.vsparc.org

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

TABLE I: A library of robot behaviors

This library-based framework allows users to express desired robot actions in an abstract way by specifying behavior properties. For example, if a task specifies that the robot should execute a behavior with the **drop** property, our system could choose to use either the Car or Proboscis configurations to perform the action, since both have behaviors with the **drop** property. The decision of which configuration to use is made during task execution, based on the sensed environment. For example, if the perception system reports that the environment is of type "tunnel", the Proboscis configuration will be used, because the library indicates that it can be used in "tunnel"-type environments while the Car cannot.

D. 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) [21, 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 a 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 sensor 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). 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 6. Currently, reconfiguration plans from one configuration to another are created manually and stored in the

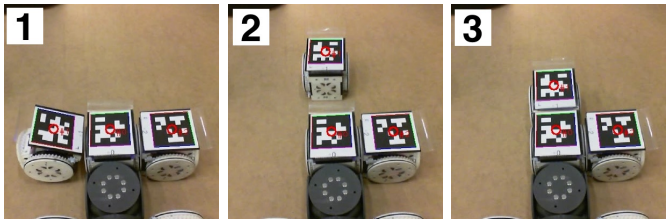


Fig. 6: Module movement during reconfiguration. (1) Left module detaches from cluster. (2) Module drives to waypoint, and spins to establish its heading. (~~2~~) Module drives straight in to dock point, and reattaches.

library. In the future, existing assembly planning algorithms ([23, 24]) could be adapted to generate reconfiguration plans automatically. Because the reconfiguration zone is free of obstacles, we 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 begin 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 [14]. 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.

E. 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. Users do not specify the configurations and behaviors used to complete tasks, but rather describe desired actions in terms of goals and outcomes. Returning to our mail delivery example, the user indicates that the robot should **explore** until it locates the **mailBox**, then **drop** the object off. The sequence of configurations and behaviors used to complete this task is not specified by the user, but instead determined by the high-level planner as it continually reacts to the sensed environment.

The high-level planner treats 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.~~

We abstract the robot and environment status as a set of Boolean propositions. In the mailbox example, the robot action **drop** is `True` if the robot is currently dropping an object to the mailbox (and `False` otherwise) and the environment proposition **mailBox** is `True` if the robot is currently sensing a mailbox (and `False` otherwise). Moreover the proposition **explore** encodes whether or not the robot is currently searching for the target, the mailbox in this case. By using a library of robot configurations and behaviors as well as environment characterization tools, we can map these high-level abstraction to low-level sensing programs and robot controllers. As discussed in Section III-C, the user specifies high-level robot actions in terms of behavior properties from the library. In the mail delivery example, our system can choose to do a drop action by executing any behavior from the library which has the behavior property **drop**, and which also satisfies the current "stairs"-type environment. 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 **drop** appropriately in response to the sensed environment. Our system evaluate propositions related to the state of the environment using perception and environment characterization tools in Section III-B. For example, we can map proposition **mailBox** to the color tracking function in our perception subsystem, which assign the value `True` to **mailBox** if and only if the robot is currently seeing a mailbox with the onboard camera. The system treats propositions, such as **explore**, that require the robot to navigate in the workspace differently from the other simple robot actions, such as **drop**. In this example, we can map **explore** to behavior property **drive**, which represents a set of parametric behaviors as discussed in Section III-C. In order to obtain joint values for behaviors at run-time, a path planner takes into account the robot goal as well as the current environment information from our perception subsystem, and generates a collision-free path for the robot to follow. Our system then converts this path to joint values, which are used to execute the **drive** behaviors.

Our implementation employs an existing framework called LTLMoP to automatically generate robot controllers from user-specified high-level instructions using ~~formal methods~~ [25, 26]. The user describes the desired robot tasks with high-level specification over the set of abstracted robot and environment propositions that are mapped to behavior properties from the library. LTLMoP automatically converts the specification to logic formulas, which are then used to synthesize a robot controller that satisfies the given tasks (if one exists). The controller is 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 include 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, LTLMoP determines the values of all environment propositions by calling the corresponding sensing program. Then, LTLMoP chooses 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, for each robot proposition LTLMoP chooses a behavior from

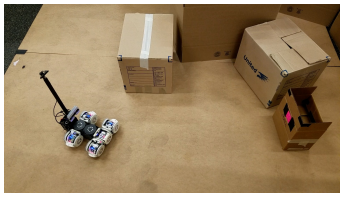
Environment Setup	Task Description
	Explore environment to find all pink or green objects and blue dropoff zone. Deliver all objects to dropoff zone.
	Explore environment to find mailbox, then deliver a circuit to the bo
	Explore environment to find package, then place a stamp on the package.

TABLE II: Environments and tasks for hardware experiments

the design library which satisfies both the behavior properties and current environment type. Since self-reconfiguration is time-consuming, if there are multiple configurations capable of executing the selected behavior, the controller ~~minimizes times for reconfiguration by biasing towards~~ the current robot configuration. If the robot is not currently in ~~a capable~~ configuration, the controller ~~commands~~ the robot to reconfigure to a randomly selected ~~capable~~ configuration.

IV. Experiment Results

In three hardware experiments, we demonstrate how our system allows the robot to autonomously explore, decide how and when to reconfigure, and manipulate objects to complete tasks in unknown environments. Table II presents the environments and tasks in each experiment. In Experiment I, the robot completes a complex, multi-part object retrieval task that requires significant exploration, reactive decision-making, and reconfiguration in response to its environment. In Experiments II and III, the robot is given the same high-level task (delivering an object to an *a priori* unknown goal) in two different environments. We show how the system successfully reacts to the two scenarios by taking very different actions, both requiring reconfiguration.

These experiments accomplish two goals. First, they fill an experimental gap in the field, representing the first evidence of a modular robot autonomously reconfiguring in response to its perceived (*a priori* unknown) environment in order to complete tasks. Second, they provide observations and insights into the challenges presented by autonomous modular robot systems. By discussing lessons learned, we hope to allow others to build on our work and create even more capable systems.

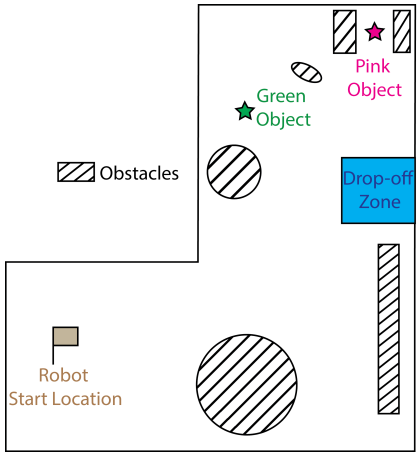


Fig. 7: Diagram of Experiment I environment

A. Experiment I

Experiment I tasks the robot with cleaning a messy graduate student office. The robot must 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 must pick them up and deliver them to a designated drop-off zone for recycling, which is marked with a blue square on the wall.

The high-level task is compact, consisting of three sub-tasks: explore the environment, find and retrieve all metal objects, and deliver each to the drop-off zone. However, it results in a long, complex execution that fully utilizes the robots exploration, reconfiguration, and manipulation abilities.

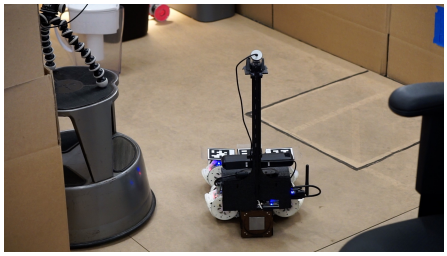
Figure 7 shows the environment layout used in the experiment, designed to represent a messy office. The environment contains two colored objects: a green soda can in the open (a "free" environment type), and a pink spool of wire in a narrow gap between two trash cans (a "tunnel" environment). Obstacles, environment types, and locations of the objects and drop-off zone are unknown *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. Both objects have small pieces of steel attached to them, making it possible for the modules to grasp them using their magnetic connectors.

Five SMORES-EP modules, one Sensor Module[5], and three passive cubes composed the robot for performing the task. During the experiment, the high-level planner used the library of configurations and behaviors described in Section III-C.

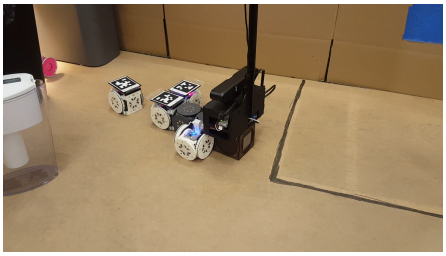
Figure 8 shows snapshots from the experiment run. A video of the entire experiment is available as an attachment to this paper. The robot starts in the "Car" configuration. The starting location prevents the robot from seeing the objects initially, forcing it to explore the environment to search for

[5]This experiment uses an older version of the sensor module. The body is larger (160 × 80 × 80mm box), and it uses an older Asus Xtion Pro Live RGB-D camera, which is larger than the Orbecc Astra but provides similar data. The only functional difference between this older sensor module and the version presented in Section III-A2 is the larger size.

(a) Environment and robot starting location

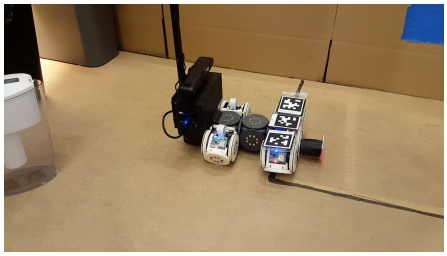


(b) Exploring while searching for objects

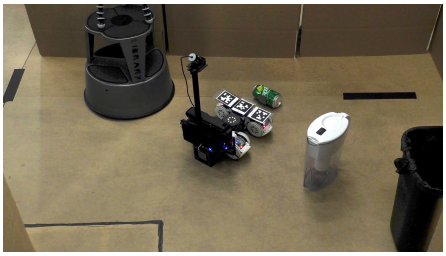


(c) Reconfiguring to retrieve pink object

(d) Retrieving pink object



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



(f) Retrieving green object

Fig. 8: Phases of Experiment I.

them. After a period of exploration, the robot locates the pink object. The characterization algorithm correctly classifies the surrounding environment as a "tunnel" type, and accordingly the robot navigates in front of the object and reconfigures to the "Proboscis" configuration. The "Proboscis" then uses its long arm to reach into the tunnel and pull the object out into the open.

The "Proboscis" can only drive forwards and backwards (it cannot turn), so the robot concludes that it must reconfigure back to the "Car" to drive the object to the drop-off zone. To do so, the robot drops the object, reconfigures, and picks the object back up. The "Car" then navigates to and drops off the object at the drop-off zone, which the robot previously located during exploration and recorded in the global map built by the SLAM algorithm.

Once done, the robot navigates directly to the green object, which it discovered while dropping off the pink object. The robot correctly determines that the green object is in a "free" environment, and picks it up without reconfiguring. The robot finishes the experiment by also delivering the green object to the drop-off zone. Figure 9 shows a volumetric map of the environment generated as the robot explores and delivers objects. The robot successfully completed all tasks in the experiment in about 26 minutes.

Note that at one point in the video a person 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

In Experiments II and III, the robot helps researchers at the University of Pennsylvania mail a circuit board to collaborators at Cornell. First, in Experiment II, the robot must place the circuit board in a mailbox, and then in Experiment III, the robot must place a postage stamp on the box so that it can be shipped. Both experiments have the same high-level task specification: starting with an object in its possession, the robot must explore until finding a drop-off point, then deposit the object at the drop-off point. As we will see, the different environments in which these tasks are executed result in very different actions.

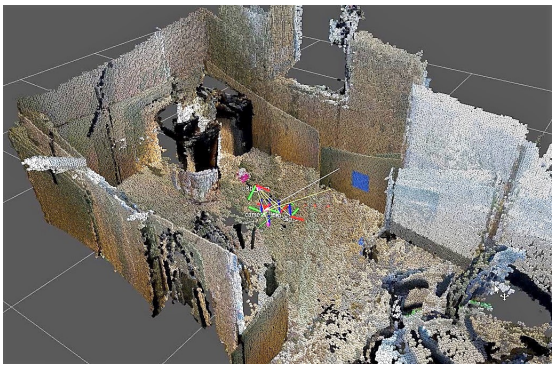
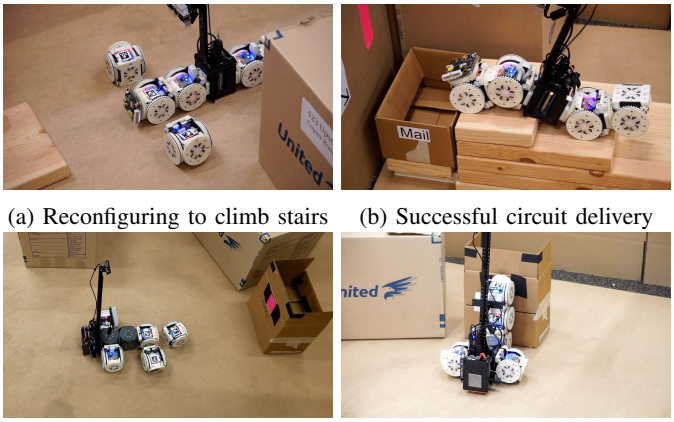


Fig. 9: Volumetric map of environment built by visual SLAM

The environment setup for Experiment II (see Table II) includes an obstacle (cardboard box), a staircase, and the mailbox at the top of the staircase. A pink rectangle indicates the mailbox. Note that this scenario is identical to the mail delivery example mentioned earlier in the paper.

The robot begins the experiment in the "Scorpion" configuration with its view of the staircase and mailbox blocked by the cardboard box. After a short period of exploration, it observes and recognizes the mailbox, and characterizes the environment surrounding it as "stairs". Based on this characterization, it determines that it must 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. 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



(a) Reconfiguring to climb stairs (b) Successful circuit delivery

(c) Reconfiguring to place stamp (d) Successful stamp placement

Fig. 10: Experiments II and III.

to the "Scorpion" configuration to end the mission.

C. Experiment III

For the final experiment, a student gives the robot a postage stamp[6] to place on the package so that it can be shipped to Cornell. The high-level task has the same specification as in Experiment II: explore the unknown environment, find the delivery goal indicated by a pink area, and place the object at the target. The environment is different, most significantly in that the delivery goal is in a "high" environment type instead of the "stairs" environment used in Experiment II (see Table II).

The robot begins in the "Car" configuration, and cannot see the package from its starting location. After a short exploration, the robot identifies the pink square marking the package. The pink square is unobstructed, but is about 25cm above the ground, so 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.

V. Discussion

The real-world experiments demonstrate that our system has the ability to autonomously reconfigure in response to observations in the environment in order to perform high-level tasks. The system performs complex tasks, determining robot actions and reconfiguration online in response to observations of the environment. The same system successfully performs two different high-level tasks. Experiments II and III demonstrate that the same high-level task can be performed in different environments, because the robot autonomously adapts to the challenges presented by each environment. Our system is the first to successfully demonstrate this level of perception-informed reconfiguration and autonomy.

In addition to demonstrating our system's abilities, the experiments revealed several insights related to perception-informed autonomy on MSRR systems. As this is the first successful system of its kind, we encountered many challenges with achieving robust performance, and discovered many limitations of the system implementation and with the modular robot hardware. We hope to share these observations to aid those who may wish to implement our system or improve its robustness and capability.

A. Single-Sensor Architecture Enables Autonomy

Autonomy in unknown environments requires powerful sensing capabilities. Our centralized sensing architecture allowed the robot to carry a single, powerful RGB-D camera, enabling sophisticated capabilities like Visual SLAM and environment characterization. In contrast, previous systems that used distributed sensing have been forced to rely on less sophisticated sensors that can be carried by each module in the cluster, limiting their ability to operate autonomously in unknown environments. Our system has demonstrated more sensor-based autonomy than any previous modular robot, and we believe this is due in large part to our decision to use a single, powerful sensor module.

Likewise, high-fidelity centralized sensing during reconfiguration, provided by AprilTags, allowed our implementation to transform between configurations quickly and reliably. Each reconfiguration action (a module disconnecting, moving, and reattaching) takes about one minute, and succeeds about 85% of the time. In contrast, past systems required 5-15 minutes for single reconfiguration actions [2, 11, 12], which would make it difficult to use them for the kinds of tasks we have demonstrated.

Centralized sensing has some drawbacks. The sensor module is larger than a single SMORES-EP module, and is only compatible with configurations that can hold it in a good view of the robot's surroundings. This also limits the kinds of behaviors that can be performed, because care must be taken to hold the sensor level and steady. The configurations and behaviors used in our experiments move more slowly than in past experiments with SMORES-EP, in part for this reason. Supporting the sensor module while traversing obstacles and rough terrain is particularly challenging. In Experiment II, we used rigid connectors to attach the sensor module to the surrounding SMORES-EP modules, in order to avoid connection breakage when climbing the stairs.

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

We believe the benefits provided to our system by centralized sensing outweigh the costs. The decision represents

[6]The postage stamp has a thin steel backing, so that the robot can pick it up with its magnets, and so that it adheres to magnets on the package.

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 III: Reasons for experiment failure.

a tradeoff, sacrificing some of the flexibility of the modular system to achieve more capability and autonomy. As future modular systems move out of the lab and into real-world deployment, it will become increasingly important for designers to consider similar tradeoffs.

It's worth noting that while our implementation relies in part on external computation and wireless network access, 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. Strengths - Reactive Capability

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).

We selected the three scenarios in our experiments 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.

C. Major Challenge - Robustness

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 III 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 module 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.

D. Challenges for Future Systems

Based on our small-scale, proof of concept experiments, we believe this system has the potential to be scaled and deployed in a real-world setting. Our experiences have provided us with some insight into the challenges future systems might face.

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 be needed. 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 and maneuver in the environment relies heavily on 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 general, real-world application would require modules with more powerful actuators, and the ability to hold large payloads without breaking inter-module connections.

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 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, the system would need a large set of attributes 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 for behaviors with user specified constraints, our method scales linearly with respect to the size of the library.

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.

E. Conclusion

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] M. Yim, W. Shen, and B. Salemi, "Modular self-reconfigurable robot systems: Challenges and Opportunities for the Future," *IEEE Robotics & Automation Magazine*, no. March, 2007.

[2] M. Yim, B. Shirmohammadi, J. Sastra, M. Park, M. Dugan, and C. J. Taylor, "Towards Robotic Self-reassembly After Explosion," in *IROS*, 2007.

[3] G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit, "An End-To-End System for Accomplishing Tasks with Modular Robots," *Robotics: Science and Systems XII*, 2016.

[4] M. Yim, "Locomotion with a unit-modular reconfigurable robot," Ph.D. dissertation, Stanford University, 1994.

[5] R. Grabowski, L. E. Navarro-Serment, C. J. J. Paredis, and P. K. Khosla, "Heterogeneous teams of modular robots for mapping and exploration," *Autonomous Robots*, vol. 8, no. 3, pp. 293–308, 2000.

[6] M. Dorigo, E. Tuci, R. Groß, V. Trianni, T. H. Labella, S. Nouyan, C. Ampatzis, J.-L. Deneubourg, G. Baldassarre, S. Nolfi, F. Mondada, D. Floreano, and L. M. Gambardella, "The SWARM-BOTS Project," *LNCS*, vol. 3342, pp. 31–44, 2005.

[7] F. Mondada, L. M. Gambardella, D. Floreano, S. Nolfi, J. L. Deneubourg, and M. Dorigo, "The cooperation of swarm-bots: Physical interactions in collective robotics," *IEEE Robotics and Automation Magazine*, vol. 12, no. 2, pp. 21–28, 2005.

[8] R. Groß, M. Bonani, F. Mondada, and M. Dorigo, "Autonomous self-assembly in swarm-bots," *IEEE transactions on robotics*, vol. 22, no. 6, pp. 1115–1130, 2006.

[9] R. OGrady, R. Groß, A. L. Christensen, and M. Dorigo, "Self-assembly strategies in a group of autonomous mobile robots," *Autonomous Robots*, vol. 28, no. 4, pp. 439–455, 2010.

[10] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugniere, G. Di Caro, F. Ducatelle, E. Ferrante, A. Förster, J. M. Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. Montes De Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard, "Swarmanoid: A novel concept for the study of heterogeneous robotic swarms," *IEEE Robotics and Automation Magazine*, vol. 20, no. 4, pp. 60–71, 2013.

[11] M. Rubenstein, K. Payne, P. Will, and W.-M. S. W.-M. Shen, "Docking among independent and autonomous CONRO self-reconfigurable robots," *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, pp. 2877–2882, 2004.

[12] S. Murata, K. Kakomura, and H. Kurokawa, "Docking experiments of a modular robot by visual feedback," *IEEE International Conference on Intelligent Robots and Systems*, pp. 625–630, 2006.

[13] J. Paulos, N. Eckenstein, T. Tosun, J. Seo, J. Davey, J. Greco, V. Kumar, and M. Yim, "Automated Self-Assembly of Large Maritime Structures by a Team of Robotic Boats," *IEEE Transactions on Automation Science and Engineering*, pp. 1–11, 2015.

[14] T. Tosun, J. Davey, C. Liu, and M. Yim, "Design and characterization of the ep-face connector," in *IROS*. IEEE, 2016.

[15] T. Tosun, D. Edgar, C. Liu, T. Tsabedze, and M. Yim, "Paintpots: Low cost, accurate, highly customizable potentiometers for position sensing," in *ICRA*. IEEE, 2017.

[16] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3400–3407.

[17] 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*, Sept 2014, pp. 2661–2666.

[18] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013.

[19] J. Daudelin and M. Campbell, "An adaptable, probabilistic, next-best view algorithm for reconstruction of unknown 3-d objects," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1540–1547, July 2017.

[20] G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit, "Accomplishing High-Level Tasks with Modular Robots,"

Autonomous Robots, submitted.

[21] J. Davey, N. Kwok, and M. Yim, "Emulating self-reconfigurable robots-design of the SMORES system," in *IROS*, 2012, pp. 4464–4469.

[22] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans, "Modular reconfigurable robots in space applications," *Autonomous Robots*, vol. 14, no. 2, pp. 225–237, 2003.

[23] J. Werfel, D. Ingber, and R. Nagpal, "Collective construction of environmentally-adaptive structures," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2345–2352, 2007.

[24] J. Seo, M. Yim, and V. 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)*, pp. 1016–1021, aug 2013.

[25] C. Finucane, G. Jing, and H. 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*, 2010, pp. 1988–1993.

[26] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.