Configuration Recognition with Distributed Information for Modular Robots

Chao Liu and Mark Yim

Abstract Modular robots are usually composed of multiple blocks with uniform docking interfaces that can be transformed into different configurations. It is a significant challenge to recognize modular robot configurations composed of hundreds of modules. Given a new configuration, it is important to match it to an existing configuration and, if true, map each module to the module in this matching configuration when applying many modular robot control schemes. An efficient algorithm is presented to address this matching and mapping problem by making use of distributed information from each module and new structure to design the configuration library. The cluster is discovered and the root module is determined first. Then the matching and mapping problem is solved simultaneously in polynomial time. The algorithm is demonstrated on real modular robots and shown to be efficient to solve this configuration recognition problem.

1 Introduction

Self-reconfigurable modular robots are usually composed of multiple building blocks of a relatively small repertoire, with uniform docking interfaces that allow transfer of mechanical forces and moments, electrical power, and communication throughout the robot [1]. Numerous modular robotic systems have been developed in the past few years. The design goal for modular robots is supposed to be versatile, robust and low-cost. Modular robots should be able to adapt or be adapted to many different functions or activities, handle hardware and software failures, and also be cost-effective to be used in more cost-sensitive tasks [2]. With these advantages,

Chao Liu
University of Pennsylvania, Philadelphia PA, e-mail: chaoliu@seas.upenn.edu

Mark Yim
University of Pennsylvania, Philadelphia PA, e-mail: yim@grasp.upenn.edu

modular robotic systems are promising to do a wide variety of tasks. However, it is also a challenge to come up with planning and control algorithms to handle numerous modules. One key reason is that, the number of all possible configurations of the system increases drastically as the number of modules increases. Plus, each module may have multiple degrees of freedom and the configurations in topology become even more complicated. Hence, configuration recognition, namely matching a new modular robotic configuration to an existing configuration in a library and mapping each module in these two configurations, is an essential problem we need to solve for planning and control of modular robots. Automatic configuration recognition is the process by which a modular system can determine its own configuration without having it explicitly programmed and a variety of its uses are described in [3]. In addition to those applications, automatic configuration recognition is also necessary for fully autonomous modular robots.

Graphs have been shown useful to represent modular robot configurations and there are a number of techniques from graph theory that we can use. However, it has been mentioned that the number of all possible configurations of modular robots grows drastically as the number of modules increases, so it is challenging to put forward an efficient algorithm for configuration recognition, which is even more difficult when taking the connection types between modules into consideration. This paper presents a new representation of modular robot configurations and a polynomial time algorithm for configuration recognition which can finish matching and mapping problem simultaneously.

The paper is organized as follows. Sec. 2 reviews relevant and previous works. Sec. 3 introduces the hardware platform which is used as an example in the paper and how the local information is gathered. Sec. 4–5 discuss the design of library and the algorithm with enough details. Some experiment results are shown in Sec. 6. Finally, Sec. 7 talks about the conclusions and future work.

2 Related Work

Configuration recognition is an important part for modular robotics research. There is a variety of work on this topic. Chen and Burdick [4] presented a method to enumerate the non-isomorphic assembly configurations of a set of modules. Assembly incidence matrix (AIM) is used to represent a modular robot configuration. Park et al. [3] compared three methods of matching and mapping, an automorphism grouping method (*nauty*), a spectral decomposition approach and a heuristic graph search (3DLL). In particular, the *nauty* method first finds all graph isomorphism configurations with the adjacency matrix of the given robot configuration and then finds the matching configuration by checking the port-adjacency matrix and considering the symmetry property of the modules. However, this method scales badly with increase in the number of modules [3]. The spectral decomposition method is trying to find the permutation matrix between isomorphic configurations which are represented by port-adjacency matrices. This method requires more computation for structures

with symmetry and numerical stability is a concern for large matrices, and the time to compute eigenvectors becomes extremely long at very large scale. For the 3DDL method, robots are represented by a three-dimensional linked list of module objects (3DDL) and some heuristics are evaluated in order to filter out most of the configurations in the library with respect to all possible choices of origin module. However, for different modular systems, different reasonable heuristics may need to be designed and some heuristics can be invariant under some similar configurations so that it cannot find the accurate matching. Also, the 3DLL method is specific to CKBot and cube-oriented modules [3]. An algorithm based on the methods of linear algebra to recognize isomorphism between two configuration graphs of modular robots is discussed in [5]. It is also focusing on the configuration matrix based on the adjacency matrix and the complexity is $\mathcal{O}(n^6 \log n)$ in terms of manipulations of elementary operations [5]. An improved 3DLL approach is presented in [6]. It first finds all graph isomorphism configurations with the adjacency matrix (similar to *nauty*) which may take a lot of time for configurations with large amount of modules [6] and then compare the configuration's space representation list with those in the library which requires the global position and orientation information with respect to a base module, so it only works for modular systems with fixed discrete states for each degree of freedom.

Butler et al. [7] presented distributed goal recognition algorithms but only to solve the matching problem. Castano and Will [8] used directed graphs to represent a modular robot CONRO and showed the corresponding configuration matrix can be found and used for configuration identification. Baca et al. [9] also presented a real-time distributed algorithm to discover the modules and the topology of the configuration. They both only focused on the configuration discovery and didn't address the configuration recognition problem.

In this paper, a local algorithm to do cluster discovery of modular robots in linear time is presented, and then a more efficient polynomial time algorithm for configuration recognition is presented which requires a new design of library to store modular robot configurations so that the configuration matching and mapping problems can be solved simultaneously. We also demonstrate this approach on a real modular robotic system and experiment results and necessary analysis are provided.

3 Hardware Platform

A modular robotic system named SMORES is used as an example for analysis and demonstration. SMORES is a modular robotic system published in 2012 and SMORES-EP is the current version of the system where EP refers to the Electro-Permanent magnets the module use for its connector [10]. Each module has four active rotational degrees of freedom (pan, tilt and left/right wheels) and four connectors which are equipped with an array of electro-permanent (EP) magnets as illustrated in Fig. 1.

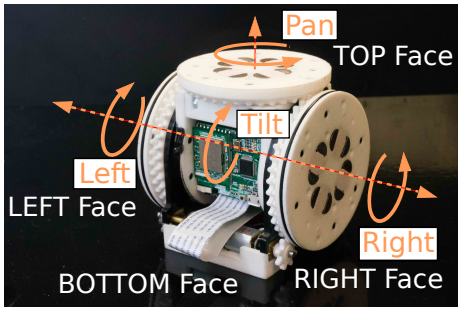


Fig. 1 A SMORES-EP module with four active rotational degrees of freedom and four connectors using an array of electro-permanent (EP) magnets.

These four degrees of freedom are named *LEFT DoF*, *RIGHT DoF*, *PAN DoF* and *TILT DoF* for convenience. In particular, LEFT DoF, RIGHT DoF and PAN DoF can continuously rotate (no angular limits on rotation) to produce a twist motion of docking ports relative to the rest of the module, and TILT DoF is limited to ± 90 degrees to produce a bending joint.

A SMORES module can be considered as a cube with four docking ports named *LEFT Face*, *RIGHT Face*, *TOP Face* and *BOTTOM Face* for convenience. Each face of the module can form a strong connection with other modules by the use of four EP magnets arranged in a ring, with south poles counterclockwise of north shown in Fig. 2. The ring arrangement of the magnets makes the connector able to connect in four possible configurations. Also connected EP-Faces are able to exchange data through the magnetic coupling of connected EP-magnets which are capable of UART serial communication [10].

Each module also has a 802.11b wireless module configured to send and receive UDP packets. A center computer is used to control the whole system and all modules exchange data with the center computer via wireless network provided by a router.

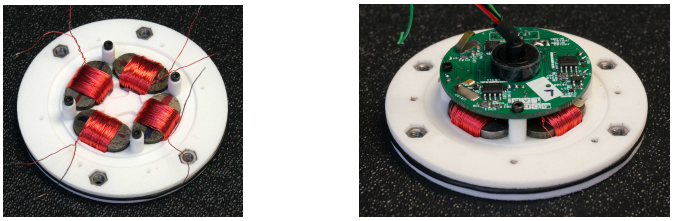


Fig. 2 (a) Internal view of magnets in EP-Face. (b) Internal view of EP-Face with circuit board.

4 Configuration Topology and Library Design

A good configuration representation is helpful for configuration recognition. In particular, a modular robot configuration can be represented as a graph $G = (V,E)$. Each vertex of the graph represents a module and each edge of the graph represents the connection between two modules. This graph can then be converted into an adjacency matrix and then a port adjacency matrix [3] or a configuration graph matrix [5] which has been discussed in Sec. 2. A new representation is presented in this section which can be used by our algorithm and SMORES modular robotic system is used as an example.

Let $G = (V,E)$ be an undirected graph, where V is the set of vertices of G and E is the set of edges of G. For connections, some attributes have to be added for modular robots which is discussed later. The maximum degree of a vertex is the number of connectors the corresponding module has.

Graphs with only one path between each pair of vertices are *trees*. Any acyclic graph is a tree. Some preliminaries on rooted trees are discussed here. Once a tree $G = (V,E)$ is rooted with respect a vertex $\tau \in V$, the parent of a vertex $v \in V$ is the vertex connected to it on the path to τ which is unique except for τ, and the child of a vertex $v \in V$ is a vertex of which v is the parent. The configuration of a modular robot cluster is represented as a rooted tree and the root has to be selected as the center of its graph [11] defined as the following.

Definition 1 (McColm, 2004). Let T be a tree. The **center** of T is the unique vertex (or unique pair of adjacent vertices) such that removing that vertex (or vertices) from T leaves a collection of components each having less half the vertices of T.

Given a tree T and a vertex v, $T - v$ is the result of removing v from T. If $T - v$ is connected, v is a leaf; otherwise, $T - v$ consists of several acyclic components, and if $T - v$ has no components of order greater than $n/2$, v is a central vertex. This is the rule to define the root for the graph of a given configuration.

A modular robot module usually has multiple connectors and there may also be multiple ways to connect them. For each connection between two modules, the involved faces and orientations are meant to be considered.

Definition 2. A **connection** between module u's connector U_Con and module v's connector V_Con with orientation Ori is defined as

$$\text{connect}(u,v) = \{\text{Face} : \text{U\_Con}, \text{Face2Con} : \text{V\_Con}, \text{Orientation} : \text{Ori}\} \quad (1)$$

from module u's point of view and

$$\text{connect}(v,u) = \{\text{Face} : \text{V\_Con}, \text{Face2Con} : \text{U\_Con}, \text{Orientation} : \text{Ori}\} \quad (2)$$

from module v's point of view.

Each connection has three attributes: *Face*, *Face2Con* and *Orientation*, so modular robots can be connected in multiple ways and the number of all possible configurations grows dramatically as the number of modules increases. However, some

(a) (b)

Fig. 3 Connection with two BOTTOM Faces. Orientation is 0 in (a) and Orientation is 1 in (b).

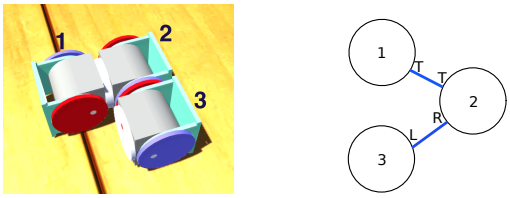
seemingly different connections can actually be considered equivalent. For example, each SMORES module has four faces (LEFT Face, RIGHT Face, TOP Face, BOTTOM Face) and each face can be configured to connect with any face of other modules. Since there are no angular limits on rotation for LEFT DoF, RIGHT DoF and PAN DoF, and EP-Faces are hermaphroditic, for connections among LEFT Face, RIGHT Face and TOP Face, the connections in which only the *Orientation* attribute is different are equivalent. In contrast, for connections between two BOTTOM Faces, the orientation is also needed to be considered and there are two different orientations (Orientation $= [0,1]$) in total which is shown in Fig. 3. In addition, the module is bilaterally symmetric, namely LEFT Face or LEFT DoF is a mirror image of the RIGHT Face or RIGHT DoF, so all possible connections between LEFT Face and RIGHT Face are also equivalent, namely all connections with LEFT Face or RIGHT Face as their *Face* or *Face2Con* attribute are equivalent. We use $\cong$ to denote the equivalent relation between two connections.

A three-module SMORES configuration example is shown in Fig. 4. For this configuration, the connections can be expressed as following

$$\text{connect}(1,2) = \{\text{Face} : \text{TOP Face}, \text{Face2Con} : \text{TOP Face}, \text{Orientation} : \mathit{Null}\}$$
$$\text{connect}(2,3) = \{\text{Face} : \text{RIGHT Face}, \text{Face2Con} : \text{LEFT Face}, \text{Orientation} : \mathit{Null}\}$$

In addition, for this simple configuration, the root module can be defined as Module 2 according to Definition 1.

The library is a collection of modular robot configurations and each configuration has its unique representation. The representation of each configuration contains the corresponding rooted graph G, the information of all connections defined in Defi-



(a) (b)

Fig. 4 (a) Three-module configuration simulation. (b) Three-module configuration graph.

nition 2 and all CN values described in Sec. 5.2. Some more properties can also be added to configurations in the library, like robot behavior property, for other design purposes. While, for configuration recognition, they are not necessary.

5 Algorithm for Configuration Recognition

We now present the modular robot configuration recognition algorithm that can solve the matching and mapping problem simultaneously in polynomial time. The algorithm contains three parts:

1. Discover the configuration of a cluster of modular robots;
2. Given a new modular robot configuration, decide the root module;
3. Verify if the configuration can be matched onto an existing configuration in the library and, if true, map each module to the module in this known isomorphic configuration.

5.1 Configuration Discovery

When a modular robot configuration is constructed, a fully autonomous robotic system has to be able to figure out the configuration topology by itself, then the graph representation can be discovered to represent its current configuration. This discovery process is affected by the hardware design, especially how a module communicates with its neighbors and the communication protocol. For the SMORES modular robotic system, every module can talk to its neighbors via EP-Faces and sending and receiving messages share the same route so that each EP-Face is either in sending state or receiving state. In addition, the system requires a center controller to control the whole cluster of modules. Each module can only exchange data with this center controller via Wi-Fi and exchange data with other modules via EP-Face.

We present a centralized cluster discovery algorithm based on distributed information among modules. The pseudocode is presented in Algorithm 1. The input to this algorithm is a starting module. This module can be selected randomly. The output is all the connections in this configuration. The idea is to traverse every module in the breadth-first search order and record every connection one by one.

Line 9: The algorithm only checks faces that are not verified. Initially every face of every module is not verified and each face can be verified via two conditions: a. when a face is not verified in *Line 9*, mark it to be verified; b. when a face receives messages in *Line 13*, mark it to be verified.

Line 10 — 12: Switch all not verified faces in the configuration to Receiving Mode except F to wait for the message sent from F.

Line 13 — 16: If there is a connection that this face F is involved in, the connected face F' should be able to receive the message sent from F. Then after some certain time during which F is already switched to Receiving Mode in *Line 14*,

Algorithm 1: Configuration Discovery

```
   Input: Start module S
   Output: Configuration Connections
1  Create empty set V to store visited modules;
2  Create empty queue Q;
3  Q.enqueue(S);
4  while Q is not empty do
5      Current module C = Q.dequeue();
6      if C is not in V then
7          Add C to V;
8          Create empty set NM to store newly recognized modules;
9          for Each Face F of C that is not verified do
10             for Each module M ∉ V do
11                 for Each Face F' of M that is not verified do
12                     Switch F' to Receiving Mode;
13             Switch F to Sending Mode and send message;
14             Switch F to Receiving Mode;
15             Update connections;
16             Update NM;
17         Q.enqueue(NM − V);
```

the connected module will control F' to send out feedback message and turn off the communication function for F' so F should receive this feedback message then module C will also turn off the communication function for F. Only MAGNET 1 is active when sending and all magnets are active when in Receiving Mode so that the orientation can be determined by checking which magnet can receive message. Both messages contain module ID and sending face information. Then both modules will store the connection information. This connected module will then be added to set NM in *Line 16*. If there is no connection, namely after some certain time F doesn't receive any feedback message, module C will also record that there is no connection on F and turn off the communication function, and set NM will remain the same.

Once this process is done, every module stores its connection information locally and the center computer will request this information from modules one by one. This will then be used to build the graph representation of the configuration.

5.2 Root Module

The root module is defined to be the center vertex of the graph of a given modular robot configuration. As discussed in Sec. 4, intuitively this can be done by iteratively removing each vertex in the graph and counting the number of vertices in these generated acyclic components that requires the traversal of these subgraphs. An efficient algorithm using dynamic programming to figure out the root module

is presented here. The idea is to compute the order of all acyclic components after removing every vertex in the graph, and then find the one or a pair satisfying the requirement. This problem has been solved in [12] in a distributed way in that every module runs the same code and exchanges connection information with its neighbors. This requires the robots to exchange more data and it is time-consuming when coordinating all the modules and ensuring that they can receive data properly. This is especially difficult for a system like SMORES where the sending mode and receiving mode between connectors share the same route. Hence, we present this centralized algorithm which uses local information from previous step.

Based on the gathered local information, the graph of the configuration $G = (V,E)$ can be built and rooted with respect to a random module v_0. The parent and children of any vertex $v \in V$ are then determined. We denote the set of connectors as $\mathscr{C}$ and define an array $\mathrm{CN}^v(c)$ denoting the total number of modules connected to v via its connector $c \in \mathscr{C}$ (as in [12]) which is the number of vertices of the component of $T - v$ corresponding to connector c (for SMORES modules, $\mathscr{C} = \{\text{LEFT Face}, \text{RIGHT Face}, \text{TOP Face}, \text{BOTTOM Face}\}$). The number of modules n satisfies $n = \sum_{c\in\mathscr{C}} \mathrm{CN}^v(c) + 1,\ \forall v \in V$ and a configuration graph G can be rooted with respect to all n vertices but $\mathrm{CN}^v(c)$ is invariant under the root.

For any vertex $v \in V$ that is not a leaf with respect to root τ, we denote its child connected via its connector c as $\hat{v}^c$, the mating connector of $\hat{v}^c$ as $\hat{c}'$, the set of its children as $\mathscr{N}(v,\tau)$ and the set of c connected with its children as $\mathscr{C}_d(v) \subseteq \mathscr{C}$. If $v \in V$ has both a parent and some children, and $\mathscr{N}(v,\tau)$ and $\mathscr{C}_d(v)$ are known, then

$$\mathrm{CN}^v(c) = \sum_{\hat{c}\in\mathscr{C}-\hat{c}'} \mathrm{CN}^{\hat{v}^c}(\hat{c}),\ c \in \mathscr{C}_d(v) \text{ and } \hat{v}^c \in \mathscr{N}(v,\tau) \tag{3a}$$

$$\mathrm{CN}^v(c) = n - 1 - \sum_{c'\in\mathscr{C}_d(v)} \mathrm{CN}^v(c'),\ c \notin \mathscr{C}_d(v) \tag{3b}$$

According to Definition 1, root module τ has to satisfy the following condition

$$\mathrm{CN}^\tau(c) \leq \frac{1}{2}n,\ \forall c \in \mathscr{C} \tag{4}$$

We denote the descendants of $v \in V$ with respect to root τ as $\mathrm{desc}(v,\tau)$. With the above recursive solution, solving $\mathrm{CN}^{v_1}(c_1)$ and $\mathrm{CN}^{v_2}(c_2)$ have overlapping subproblems if $v_2 \in \mathrm{desc}(v_1,\tau)$. They both need to solve $\mathrm{CN}^u(c)$, where $u \in \mathrm{desc}(v_2,\tau)$ and $c \in \mathscr{C}_d(u)$. A bottom-up algorithm is constructed. The pseudocode is presented in Algorithm 2. The algorithm starts from vertices whose height are one. If $v \in V$ is a leaf, then $\mathscr{C}_d(v) = \emptyset$ and $\mathrm{CN}^v(c) = 0$ except when c is the connector connected with its parent. For $v \in V$ with $h(v) > 0$, $\mathscr{N}(v,v_0) \neq \emptyset$ and $\mathscr{C}_d(v) \neq \emptyset$. We can compute $\mathrm{CN}^v(c)$ when $c \in \mathscr{C}_d(v)$ using Eq. (3a) and also compute $\mathrm{CN}^{\hat{v}^c}(\hat{c}')$ where $h(\hat{v}^c) = h(v) - 1$ using Eq. (3b). In this iteration, $\mathrm{CN}^v(c)$ where $c \notin \mathscr{C}_d(v)$ is not computed which will be computed when visiting its parent vertex. After this iteration, move to the modules with higher height and the algorithm ends until the predefined root v_0 is visited. It is clear that the root module can be found in time $\mathscr{O}(|V|)$.

Algorithm 2: Root Module Search

Input: Graph representation $G = (V, E)$
Output: Root module τ

Root $G = (V, E)$ with respect to a module v_0 with height of H and the height of $v \in V$ is $h(v)$;
Initialize $\mathrm{CN}^v(c)$ for $v \in V$ and $c \in \mathscr{C}$ to be zero;
Initialize $\mathbf{h} \leftarrow 1$;
while $\mathbf{h} \leq H$ **do**
  for *Each module v with* $h(v) = \mathbf{h}$ **do**
    for *Each module* $\hat{v}^c \in \mathscr{N}(v, \hat{\tau})$ **do**
      $\mathrm{CN}^v(c) = \sum_{\hat{c} \in \mathscr{C} - \hat{c}'} \mathrm{CN}^{\hat{v}^c}(\hat{c}),\ c \in \mathscr{C}_d(v)$;
      $\mathrm{CN}^{\hat{v}^c}(\hat{c}') = n - 1 - \sum_{\hat{c} \in \mathscr{C}_d(\hat{v})} \mathrm{CN}^{\hat{v}^c}(\hat{c})$;
  $\mathbf{h} \leftarrow \mathbf{h} + 1$;
return τ such that $\mathrm{CN}^{\tau}(c) \leq \frac{1}{2} n \ \forall\ c \in \mathscr{C}$

5.3 Matching and Mapping

The configuration graph $G = (V, E)$ can be rooted with respected to the root module defined from Algorithm 2. Then given two configurations, we want to verify if they are isomorphic in terms of modular robotic system topology and, if true, map each module from one configuration to that in another configuration. Intuitively, this can be solved by traverse the trees from the root to the leaves and check all the edges and the corresponding $\mathrm{CN}^v(c)$ values and, if the connections are equivalent and $\mathrm{CN}^v(c)$ are equal, then these two connections share the same topology. However, since modular robots are usually symmetric in their geometry, for example, for SMORES system, the left side is a mirror image of the right side for each module, there will be multiple candidates for some connection when trying to find the equivalent one in the other configuration. It is a heavy computation burden to track all the options and the number of cases to track can quickly grow.

If two modular robot configurations $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, then each module $v_1 \in V_1$ must be able to be mapped to a unique module $v_2 \in V_2$ who shares the same topology and the number of mapped pairs of modules are the number of modules in G_1 or G_2. Hence, if there exists this bijective mapping $f : V_1 \to V_2$, then G_1 and G_2 are isomorphic and f is the mapping we are looking for. For each module $v_1 \in V_1$, there may be multiple modules in G_2 that share the same topology with v_1 in G_1. The idea of our algorithm is to try to find some possible mappings which result in multiple common subgraphs of both configurations, and, if these two configurations are isomorphic, then there must be one common subgraph containing all the modules that is also the isomorphism mapping.

Given two modular robot configurations $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, there may be some common parts between them. Two definitions are introduced:

Definition 3. Given two modular robot configurations $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a **common subconfiguration** is a set of connected graphs $\{G_1', G_2'\}$ where

$G'_1 = (V'_1, E'_1) \subseteq G_1$, $G'_2 = (V'_2, E'_2) \subseteq G_2$ such that G'_1 and G'_2 are isomorphic. The corresponding bijective **common subconfiguration mapping** is defined as $f' : V'_1 \to V'_2$.

Definition 4. Given the condition that module $v_1 \in V_1$ of $G_1 = (V_1, E_1)$ must be mapped to module $v_2 \in V_2$ of $G_2 = (V_2, E_2)$, the common subconfiguration with maximum common connections is called **maximum common subconfiguration with respect to** v_1 **and** v_2 denoted as $\mathrm{MCS}(v_1, v_2)$ with mapping $f : \hat{V}_1 \to \hat{V}_2$ where $\hat{V}_1 \subseteq V_1$ and $\hat{V}_2 \subseteq V_2$. If a module pair (v'_1, v'_2) satisfies $v'_1 \in \hat{V}_1$, $v'_2 \in \hat{V}_2$ and $f(v'_1) = v'_2$, then $(v'_1, v'_2) \in \mathrm{MCS}(v_1, v_2)$ under $f : \hat{V}_1 \to \hat{V}_2$.

Given two graphs G_1 and G_2 rooted with respect to $\tau_1 \in V_1$ and $\tau_2 \in V_2$ respectively, for module $v_1 \in V_1$ and $v_2 \in V_2$, we can construct a common subconfiguration $\{G'_1, G'_2\}$ where $V'_1 = \{v_1, \hat{v}_1^{c_1}\}$, $V'_2 = \{v_2, \hat{v}_2^{c_2}\}$ under a subconfiguration mapping $f' : V'_1 \to V'_2$ such that $f'(v_1) = v_2$ and $f'(\hat{v}_1^{c_1}) = \hat{v}_2^{c_2}$ if and only if $\mathrm{connect}(v_1, \hat{v}_1^{c_1}) \cong \mathrm{connect}(v_2, \hat{v}_2^{c_2})$ which is called **feasibility rule**. We can make a stronger feasibility rule by adding $\mathrm{CN}^{v_1}(c_1) = \mathrm{CN}^{v_2}(c_2)$ then the rule becomes a sufficient condition which is expressed as a form of a function $F((v_1, c_1), (v_2, c_2))$. c_1 is not necessarily to be equal to c_2 because of the symmetry property of modules.

Theorem 1. *Given* $\mathrm{MCS}(v_1, v_2)$ *under mapping* $f : V_1 \to V_2$*, for any module pair* $(v'_1, v'_2) \in \mathrm{MCS}(v_1, v_2)$ *under* $f : V_1 \to V_2$*,* $\mathrm{MCS}(v'_1, v'_2)$ *is equal to* $\mathrm{MCS}(v_1, v_2)$.

According to Theorem 1 and previous analysis, if $F((v_1, c_1), (v_2, c_2))$ is true, then $(\hat{v}_1^{c_1}, \hat{v}_2^{c_2}) \in \mathrm{MCS}(v_1, v_2)$ under $f : V_1 \to V_2$ and $\mathrm{MCS}(\hat{v}_1^{c_1}, \hat{v}_2^{c_2})$ is equal to $\mathrm{MCS}(v_1, v_2)$. Hence, whether computing $\mathrm{MCS}(v_1, v_2)$ is equivalent to computing $\mathrm{MCS}(\hat{v}_1^{c_1}, \hat{v}_2^{c_2})$ depends only on $F(\hat{v}_1^{c_1}, \hat{v}_2^{c_2})$. If two modular robot configurations $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, τ_1 and τ_2 are the only root module of G_1 and G_2 respectively, then a common subconfiguration $\{G_1, G_2\}$ under mapping $f : V_1 \to V_2$ must exist and this common subconfiguration is actually $\mathrm{MCS}(\tau_1, \tau_2)$. There may be multiple options of $f : V_1 \to V_2$ because of the symmetry property of the modules. The problem to compute $\mathrm{MCS}(\tau_1, \tau_2)$ can be converted

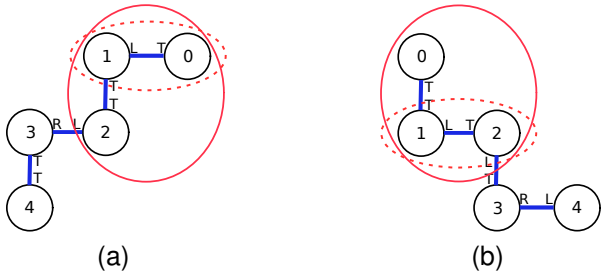


Fig. 5 SMORES Configurations. The subgraphs of (a) and (b) circled by "- -" is an example of common subconfiguration with mapping $1 \to 1$ and $0 \to 2$. The subgraphs of (a) and (b) circled by "—" is $\mathrm{MCS}(1,1)$ with mapping $1 \to 1$, $2 \to 0$ and $0 \to 2$.

into the problem to compute $\mathrm{MCS}(\hat{\tau}_1^{c_1}, \hat{\tau}_2^{c_2})$ just by checking $F((\tau_1, c_1), (\tau_2, c_2))$. This gives us the recursive solution to compute $\mathrm{MCS}(\tau_1, \tau_2)$ and the subproblem is to solve $\mathrm{MCS}(\hat{\tau}_1^{c_1}, \hat{\tau}_2^{c_2})$ where $c_1 \in \mathscr{C}_d(\tau_1)$ and $c_2 \in \mathscr{C}_d(\tau_2)$. Whichever of these MCSs is equivalent to $\mathrm{MCS}(\tau_1, \tau_2)$ is the solution and the corresponding mapping $f : V_1 \to V_2$ is generated accordingly. For some configurations that may have a pair of root modules while each configuration is only rooted with respect to one root in the library, we just need to root this configuration with respect to both of the root modules and compare both of them with that in the library. With this, we bring up a bottom-up algorithm to check the isomorphism and map modules simultaneously.

The algorithm takes two rooted configuration graphs as input. Some preliminary checks can be added here, like the height of trees, the number of modules in each level and so on. If they fail these tests, they cannot be isomorphic because there can be two root modules at most and, if two graphs are isomorphic, then they should have the same root modules. We start this bottom-up algorithm from the modules that are leaves of two given configurations and move generally to their roots. During the process, all possible MCS which may be equivalent to $\mathrm{MCS}(\tau_1, \tau_2)$ are computed and the corresponding mapping is generated. In the end, the one that contains all the modules is the solution and the common subconfiguration mapping tells us how to map each module. The pseudocode is presented in Algorithm 3. For any vertex $v \in V$ with its depth $d(v) > 0$, we denote its parent connected via its connector c as $\tilde{v}^c$ and the mating connector of $\tilde{v}^c$ as $\tilde{c}'$.

Line 8 — 9: It is possible that (v_1, v_2) or $(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ has been added to some MCSs with respect to some pair of modules. $\mathscr{MCS}_P$ is the set of all MCSs containing $(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ with corresponding mapping f_p . Similarly $\mathscr{MCS}_C$ is the set of all MCSs containing (v_1, v_2) with corresponding mapping f_c.

Line 10 — 24: If $F((\tilde{v}_1^{c_1}, \tilde{c}_1'), (\tilde{v}_2^{c_2}, \tilde{c}_2'))$ is true, then $\mathrm{MCS}(v_1, v_2)$ is equal to $\mathrm{MCS}(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$. When $\mathscr{MCS}_P$ is empty but $\mathscr{MCS}_C$ is not empty, then any $\mathrm{MCS}(u_1, u_2) \in \mathscr{MCS}_C$ is equal to $\mathrm{MCS}(v_1, v_2)$ from Theorem 1. Hence, as long as $\tilde{v}_1^{c_1} \notin U_1 \wedge \tilde{v}_2^{c_2} \notin U_2$, each $\mathrm{MCS}(u_1, u_2)$ is also equal to $\mathrm{MCS}(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ and u_1 and u_2 are updated accordingly. Then $\mathscr{MCS}$ is also updated which is to remove the old $\mathrm{MCS}(u_1, u_2)$ if existing and add the new $\mathrm{MCS}(u_1, u_2)$. Similar to the previous case, when $\mathscr{MCS}_P$ is not empty and $\mathscr{MCS}_C$ is empty, then, as long as $v_1 \notin P_1 \wedge v_2 \notin P_2$, $\mathrm{MCS}(p_1, p_2)$ is equal to $\mathrm{MCS}(v_1, v_2)$ and p_1 and p_2 are also updated. In addition, it is possible that $v_1 \in P_1 \wedge v_2 \notin P_2$ or $v_1 \notin P_1 \wedge v_2 \in P_2$, then there is no $\mathrm{MCS} \in \mathscr{MCS}$ equal to $\mathrm{MCS}(v_1, v_2)$ or $\mathrm{MCS}(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$. So $\mathrm{MCS}(v_1, v_2)$ or $\mathrm{MCS}(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ is a new possible MCS which may be equal to $\mathrm{MCS}(\tau_1, \tau_2)$. Thus, we add $\mathrm{MCS}(v_1, v_2)$ with mapping $f : \{ v_1, \tilde{v}_1^{c_1} \} \to \{ v_2, \tilde{v}_2^{c_2} \}$ to $\mathscr{MCS}$.

Line 25 — 27: If both $\mathscr{MCS}_P$ and $\mathscr{MCS}_C$ are empty, then $\mathrm{MCS}(v_1, v_2)$ or $\mathrm{MCS}(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ is a new possible MCS which may be equal to $\mathrm{MCS}(\tau_1, \tau_2)$. Thus, we add $\mathrm{MCS}(v_1, v_2)$ with mapping $f : \{ v_1, \tilde{v}_1^{c_1} \} \to \{ v_2, \tilde{v}_2^{c_2} \}$ to $\mathscr{MCS}$.

Line 28 — 43: This is the last case when $F((\tilde{v}_1^{c_1}, \tilde{c}_1'), (\tilde{v}_2^{c_2}, \tilde{c}_2'))$ is true that both $\mathscr{MCS}_P$ and $\mathscr{MCS}_C$ are not empty. $\mathrm{MCS}(u_1, u_2) \in \mathscr{MCS}_C$ cannot be equal to any $\mathrm{MCS}(p_1, p_2) \in \mathscr{MCS}_P$ for the reason that the algorithm starts from leaves of two given trees and multiple modules may share a parent but multiple modules cannot have the same children. In *Line 29* and *Line 30* the algorithm checks if (v_1, v_2)

Algorithm 3: Matching and Mapping

Input: Graph representations $G_1 = (V_1, E_1)$ with root τ_1 and $G_2 = (V_2, E_2)$ with root τ_2 where $h(G_1) = h(G_2) = H$
Output: $\text{MCS}(\tau_1, \tau_2)$ with $f : V_1' \to V_2'$

Create a set $\mathscr{MCS} = \{\text{MCS}(\text{null}, \text{null})\}$ with a trivial mapping $f : \emptyset \to \emptyset$;
Initialize $\mathbf{h} \leftarrow H$;
while $\mathbf{h} > 0$ **do**
  for $v_1 \in V_1$ *with* $d(v_1) = \mathbf{h}$ **do**
    $\mathscr{MCS}' \leftarrow \mathscr{MCS}$;
    for $v_2 \in V_2$ *with* $d(v_2) = \mathbf{h}$ **do**
      if $F((\vec{v}_1^{c_1}, \vec{c}_1), (\vec{v}_2^{c_2}, \vec{c}_2))$ **then**
        $\mathscr{MCS}_P = \{\text{MCS}(p_1, p_2) \in \mathscr{MCS}' \; f_p : P_1 \to P_2 \mid (\vec{v}_1^{c_1}, \vec{v}_2^{c_2}) \in \text{MCS}(p_1, p_2)\}$;
        $\mathscr{MCS}_C = \{\text{MCS}(u_1, u_2) \in \mathscr{MCS}' \; f_c : U_1 \to U_2 \mid (v_1, v_2) \in \text{MCS}(u_1, u_2)\}$;
        if $\mathscr{MCS}_P = \emptyset \wedge \mathscr{MCS}_C \neq \emptyset$ **then**
          for $\text{MCS}(u_1, u_2) \in \mathscr{MCS}_C$ **do**
            if $\vec{v}_1^{c_1} \notin U_1 \wedge \vec{v}_2^{c_2} \notin U_2$ **then**
              Add $(\vec{v}_1^{c_1}, \vec{v}_2^{c_2})$ to $\text{MCS}(u_1, u_2)$ such that $f_c(\vec{v}_1^{c_1}) = \vec{v}_2^{c_2}$;
              $u_1 \leftarrow \vec{v}_1^{c_1}$ and $u_2 \leftarrow \vec{v}_2^{c_2}$;
              Update $\mathscr{MCS}$;
        else if $\mathscr{MCS}_P \neq \emptyset \wedge \mathscr{MCS}_C = \emptyset$ **then**
          for $\text{MCS}(p_1, p_2) \in \mathscr{MCS}_P$ **do**
            if $v_1 \notin P_1 \wedge v_2 \notin P_2$ **then**
              Add (v_1, v_2) to $\text{MCS}(p_1, p_2)$ such that $f_p(v_1) = v_2$;
              $p_1 \leftarrow v_1$ and $p_2 \leftarrow v_2$;
              Update $\mathscr{MCS}$;
            else if $(v_1 \in P_1 \wedge v_2 \notin P_2) \vee (v_1 \notin P_1 \wedge v_2 \in P_2)$ **then**
              Construct $\text{MCS}(v_1, v_2)$ with $f : \{v_1, \vec{v}_1^{c_1}\} \to \{v_2, \vec{v}_2^{c_2}\}$ such that $f(v_1) = v_2$ and $f(\vec{v}_1^{c_1}) = \vec{v}_2^{c_2}$;
              Add $\text{MCS}(v_1, v_2)$ to $\mathscr{MCS}$;
        else if $\mathscr{MCS}_P = \emptyset \wedge \mathscr{MCS}_C = \emptyset$ **then**
          Construct $\text{MCS}(v_1, v_2)$ with $f : \{v_1, \vec{v}_1^{c_1}\} \to \{v_2, \vec{v}_2^{c_2}\}$ such that $f(v_1) = v_2$ and $f(\vec{v}_1^{c_1}) = \vec{v}_2^{c_2}$;
          Add $\text{MCS}(v_1, v_2)$ to $\mathscr{MCS}$;
        else
          $\mathscr{MCS}'_P = \{\text{MCS}(p_1, p_2) \in \mathscr{MCS}_P \mid v_1 \notin P_1 \wedge v_2 \notin P_2\}$;
          $\mathscr{MCS}'_C = \{\text{MCS}(u_1, u_2) \in \mathscr{MCS}_C \mid \vec{v}_1^{c_1} \notin U_1 \wedge \vec{v}_2^{c_2} \notin U_2\}$;
          if $\mathscr{MCS}'_P = \emptyset \wedge \mathscr{MCS}'_C \neq \emptyset$ **then**
            for $\text{MCS}(u_1, u_2) \in \mathscr{MCS}'_C$ **do**
              Add $(\vec{v}_1^{c_1}, \vec{v}_2^{c_2})$ to $\text{MCS}(u_1, u_2)$ such that $f_c(\vec{v}_1^{c_1}) = \vec{v}_2^{c_2}$;
              $u_1 \leftarrow \vec{v}_1^{c_1}$ and $u_2 \leftarrow \vec{v}_2^{c_2}$;
              Update $\mathscr{MCS}$;
          else if $\mathscr{MCS}'_P \neq \emptyset \wedge \mathscr{MCS}'_C = \emptyset$ **then**
            for $\text{MCS}(p_1, p_2) \in \mathscr{MCS}'_P$ **do**
              Add (v_1, v_2) to $\text{MCS}(p_1, p_2)$ such that $f_p(v_1) = v_2$;
              $p_1 \leftarrow v_1$ and $p_2 \leftarrow v_2$;
              Update $\mathscr{MCS}$;
          else if $\mathscr{MCS}'_P \neq \emptyset \wedge \mathscr{MCS}'_C \neq \emptyset$ **then**
            $\text{MCS}(p_1, p_2) \leftarrow \text{MCS}(u_1, u_2) \cup \text{MCS}(p_1, p_2), \forall \text{MCS}(p_1, p_2) \in \mathscr{MCS}'_P$ and $\forall \text{MCS}(u_1, u_2) \in \mathscr{MCS}'_C$;
            Update $\mathscr{MCS}$;
  $\mathbf{h} \leftarrow \mathbf{h} - 1$;
return $\text{MCS}(\tau_1, \tau_2)$ with $f : V_1' \to V_2'$

can be added to any MCS(p_1,p_2) or $(\tilde{v}_1^{c_1},\tilde{v}_2^{c_2})$ can be added to any MCS(u_1,u_2). $\mathscr{MCS}$ is updated according to different conditions and the special one is that when both $\mathscr{MCS}'_P$ and $\mathscr{MCS}'_C$ are not empty, we have to merge MCS(u_1,u_2) and MCS(p_1,p_2) with all possible combinations.

Line 44: Update **h** and, in the next iteration, the one-level higher modules will be compared to update $\mathscr{MCS}$. The last group of modules the algorithm compares are those with depth equal to one.

Line 45: In the end, there will be multiple MCS$(\tau_1,\tau_2) \in \mathscr{MCS}$ and multiple MCS$(q_1,q_2) \in \mathscr{MCS}$ such that $(\tau_1,\tau_2) \in$ MCS(q_1,q_2). Among all these candidates, the one covering maximum number of modules is the solution of MCS(τ_1,τ_2) with corresponding mapping $f : V_1' \to V_2'$. If the roots of both two given configurations are unique, then they are isomorphic if $V_1' = V_1$ (obviously $V_2' = V_2$) and not isomorphic if $V_1' \neq V_1$ (namely $V_2' \neq V_2$).

This algorithm can be improved to be more efficient for matching and mapping task. When checking $v_1 \in V_1$ with $d(v_1) = h$, if $\forall v_2 \in V_2$ with $d(v_2) = h$, $F((\tilde{v}_1^{c_1},\tilde{c}_1'),(\tilde{v}_2^{c_2},\tilde{c}_2'))$ is not true, then these two configurations cannot be isomorphic and the algorithm can stop or continue with next candidate. This algorithm can solve the matching and mapping problem in time $\mathscr{O}(|E_1|^2)$ or $\mathscr{O}(|V_1|^2)$ for the worst case. In reality, the number of connectors a module has is usually small, so the time for a large number of modules should be much smaller than the worst case.

6 Experiments

The integral algorithm is implemented in Python. The SMORES configuration used in this experiment is a walker with a manipulator shown in Fig. 6. The corresponding graph representations of them are shown in Fig. 7. This is a special configuration that has a lot of symmetric parts. For example, the left legs and right legs are symmetric and the front legs and the back legs are also symmetric.

Fig. 6 Walker configuration with different labels. (a) is the configuration in our library, and (b) is the new configuration to recognize.

We first run Algorithm 1 to discover all the connections in this configuration shown in Fig. 6b. The sequence to discover all the connections is connect$(0,1) \to$ connect$(0,4) \to$ connect$(0,12) \to$ connect$(1,11) \to$ connect$(1,3) \to$ connect$(4,8) \to$ connect$(12,5) \to$ connect$(11,6) \to$ connect$(3,7) \to$ connect$(3,2) \to$ connect$(6,13) \to$ connect$(2,9) \to$ connect$(7,10)$.

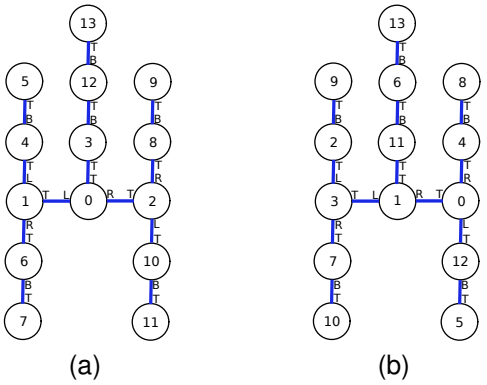


Fig. 7 Walker configuration graphs. (a) is configuration in our library, and (b) is the new design.

Then the graph $G = (V,E)$ can be built based on these connections. With $G = (V,E)$, the root and all $\mathrm{CN}^v(c)$ for $v \in V$ and $c \in \mathscr{C}$ can be computed by Algorithm 2 and the root module is Module 1. Some preliminary tests can be implemented to filter most of the configurations in the library and the one shown in Fig. 6a is the closest one. The graph representation $\hat{G} = (\hat{V},\hat{E})$ (Fig. 7a) rooted with respect to the corresponding root module (Module 0), all connection information and all CN values are stored in the library. Then run Algorithm 3 to match this new configuration $G = (V,E)$ to $\hat{G} = (\hat{V},\hat{E})$ and map each module $v \in V$ to $\hat{v} \in \hat{V}$. Since there are many symmetric parts, the mapping between G and $\hat{G}$ is not unique. By our algorithm, we can get all isomorphic mappings $f : V \to \hat{V}$. For this example, there are eight different mappings in total and one of them is $\{10 \to 5, 7 \to 4, 3 \to 1, 9 \to 7, 2 \to 6, 1 \to 0, 8 \to 9, 4 \to 8, 0 \to 2, 5 \to 11, 12 \to 10, 13 \to 13, 6 \to 12, 11 \to 3\}$.

7 Conclusion

We develop and implement an efficient algorithm for modular robot systems to do configuration recognition automatically. A new configuration can be discovered by use of local communication among modules and a graph representation can be generated. Then its root module(s) and all CN values are computed using dynamic programming. Each configuration in the library contains its rooted graph with respect to its root module, all connections and all CN values. Matching and mapping problem is solved by searching MCS and, if this new configuration is isomorphic to some configuration in the library, all the mapping results will be computed. The algorithm can be adapted to other modular systems easily as long as local communication among modules is supported and equivalent connections can be defined.

The future work will focus on the optimal mapping for modular robots since multiple mappings may exist. In addition, this algorithm should be helpful for reconfig-

uration planning to find the maximum overlapping between the initial configuration and the goal configuration. This will also be part of our future work.

Acknowledgements The authors would like to acknowledge the support of the National Science Foundation under Grants CNS-1329620 and CNS-1329692.

References

1. Yim, M., Shen, W., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., Chirikjian, G.: Modular self-reconfigurable robot systems: Grand challenges of robotics. IEEE Robotics & Automation Magazine 14(1), 43–52 (2007)
2. Stoy, K., Brandt, D., Christensen, D.: Self-Reconfigurable Robots. The MIT Press, Cambridge, MA (2010)
3. Park, M., Chitta, S., Teichman, A., Yim, M.: Automatic configuration recognition methods in modular robots. The International Journal of Robotics Research 27(3-4), 403–421 (2008)
4. Chen, I.M., Burdick, J.W.: Enumerating the nonisomorphic assembly configurations of modular robotic systems. In: Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93). vol. 3, pp. 1985–1992 (July 1993)
5. Shiu, M.C., Fu, L.C., Chia, Y.J.: Graph isomorphism testing method in a self-recognition velcro strap modular robot. In: 2010 5th IEEE Conference on Industrial Electronics and Applications. pp. 222–227 (June 2010)
6. Zhu, Y., Li, G., Wang, X., Cui, X.: Automatic function-isomorphic configuration recognition and control for ubot modular self-reconfigurable robot. In: 2012 IEEE International Conference on Mechatronics and Automation. pp. 451–456 (Aug 2012)
7. Butler, Z., Fitch, R., Rus, D., Yuhang Wang: Distributed goal recognition algorithms for modular robots. In: Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292). vol. 1, pp. 110–116 (May 2002)
8. Castano, A., Will, P.: Representing and discovering the configuration of conro robots. In: Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164). vol. 4, pp. 3503–3509 (May 2001)
9. Baca, J., Woosley, B., Dasgupta, P., Nelson, C.A.: Configuration discovery of modular self-reconfigurable robots: Real-time, distributed, ir+xbee communication method. Robotics and Autonomous Systems 91, 284 – 298 (2017)
10. Tosun, T., Davey, J., Liu, C., Yim, M.: Design and characterization of the ep-face connector. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 45–51 (Oct 2016)
11. McColm, G.L.: On the structure of random unlabelled acyclic graphs. Discrete Mathematics 277(1), 147 – 170 (2004)
12. Hou, F.: Self-Reconfiguration Planning for Modular Robots. Ph.D. thesis, University of Southern California, Los Angeles (2011)