Proceedings of the 2001 IEEE
International Conference on Robotics & Automation
Seoul, Korea · May 21-26, 2001

Representing and Discovering the Configuration of Conro Robots

Andres Castano * Peter Will
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292

Abstract

A Conro reconfigurable robot is formed by joining a set of self-contained modules in a particular configuration; the actions of the robot are the result of the coordinated actions of its modules. These actions can be controlled using a master-slave approach only if the master can map the particular configuration of the robot to one that it already knows how to control. In this paper we discuss how to describe this configuration using graphs, how to discover a robot configuration and how to identify it as a particular known configuration. The methodology used is very general and can be applied easily to other modular robots. Experimental results for Conro quadrupeds and snakes are presented.

1 Introduction

Reconfigurable robots are made of modules that can be assembled into different shapes and thus, require a flexible control to handle a variety of configurations. These robots are classified as homogeneous or heterogeneous depending on whether the robot uses a single type of module or many. They can also be classified as lattice-based depending on whether their modules are organized in a grid (in either 2-D or 3-D) or not.

Reconfigurable robots have been the subject of active research. Examples of reconfigurable robots are those of Unsal et. al [1], Murata et. al [2], Tomita et. al [3], Kotay et. al [4], and Yim et al. [5], among others, who have worked on lattice-based robots in both 2-D and 3-D. These robots are homogeneous with highly symmetrical modules, e.g., triangular and hexagonal modules for planar robots and cubic and dodecahedral modules for robots that move in 3-D. Lattice-based robots require their modules to rotate or slide along their adjacent modules to change their shape. Among the non-lattice-based robots we find the Polypod and Polybot robots of Yim [6] and the Conro robots [7]. These robots change their shape by extending limbs that can connect to any part of their bodies thus changing their topology. In contrast with the lattice-based types, these robots do not require to reconfigure in order to move; they can adopt a shape and then use a configuration-dependent gait.

The modules of the Conro robot are self-contained robots themselves; they have their own CPU, power supply, sensors, actuators and communication systems. The control of a Conro robot can be seen as the control of a distributed system where each module needs to be controlled correctly to achieve an overall robot motion. To control the robot using a master-slave approach, the master must identify the robot configuration as one of the preprogrammed configurations that it already knows how to control. This paper discusses our representation and identification of such configurations which can be adapted to other modular and reconfigurable robots.

This paper is divided as follows. In Sec. 2 we describe the Conro module and discuss the need to discover the configuration of a Conro robot and identify it as a known configuration. In Sec. 3 we present a general representation of a module and a modular robot. The case of Conro is treated as an instance of the representation. In Secs. 4 and 5 we describe the discovery and identification of the robot configuration, respectively. The experimental results, carried out on Conro quadrupeds and snakes, are discussed in Sec. 6. Finally, we present our conclusions in Sec. 7.

2 The Conro module

The Conro module, shown in Fig. 1, is a self-sufficient robot composed of a passive and an active connectors, to allow it to attach to other modules, and a body, that serves as a frame for the motors, power supply, and electronics [7], [8]. Three lateral faces of the passive connector have pins that can be inserted into holes on the face of the active connectors of other modules, thus creating structures. Hence, a module

*A. Castano was with the Information Sciences Institute at the University of Southern California during the performance of this work. Presently he is with the Jet Propulsion Laboratory at the California Institute of Technology, Pasadena, CA 91109 USA (e-mail: Andres.Castano@jpl.nasa.gov).

0-7803-6475-9/01/$10.00© 2001 IEEE

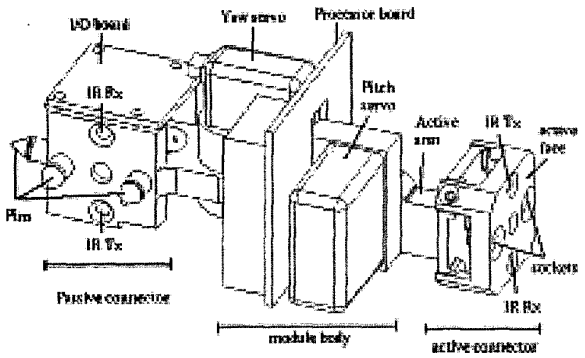

Figure 1: Anatomy of the Conro module

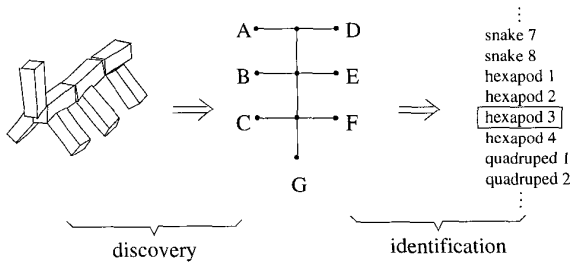

Figure 2: Configuration discovery and identification

has four docking ports: three on the passive connector and one on the active connector. Each port has an infrared pair that serves as a serial link, to establish communication with adjacent modules, and as a guidance system, to assist in the docking process of two modules. The axes of the motors lie at the joints of the connectors and body and give the module a pitch and a yaw degrees of freedom. The length of the module is 108 mm and its weight is 115 g.

A Conro robot (e.g., snake, hexapod) is formed by joining several modules together. This distributed system can be controlled using a distributed approach (e.g., [9]), a centralized master-slave approach, or a combination of both. In the master-slave approach a single module or an external host computer controls the robot. Consider the case where a Conro robot has just been assembled and powered up. Before the master can control the robot, it must identify its configuration because different configurations use different gaits and obey different kinematic and dynamic equations.

Two steps are needed to identify a robot configuration. First, as shown in Fig. 2, the master needs to discover the configuration of the robot, i.e., it needs to describe the robot using some suitable representation. Second, it must identify the configuration by comparing it against a catalog of configurations that the master already knows how to control. If the configuration is identified, the master can map the representation of the robot to that in the catalog and control the robot using preprogrammed routines that control the cataloged configuration. If the configuration is not identified, the master cannot control the robot using configuration-dependent routines.

3 The graph of a modular robot

The representation of a modular robot must take into account that a module might have multiple ports and that there might be multiple ways to connect two modules using the same two ports. A remarkable effort to formalize this representation was made by Chen and Burdick and is described in a series of papers that follow [10]. They represented a configuration using a matrix called the *assembly incidence matrix* (AIM). The AIM is a function of the way in which the modules of the robot are labeled, i.e., the labels are part of the matrix itself. Hence, two non-isomorphic AIMs might actually represent the same robot and thus it is not possible to directly compare two configurations; a step of cycling the labels of one of the AIMs to make it isomorphic with the other AIM is necessary.

In this section we represent a modular robot using directed graphs (digraphs). In contrast with the representation of Chen and Burdick and those of others (e.g., [2], [3]), a graph-only representation of the robot reduces the complexity of matching two robot configurations to exactly that of the isomorphism-complete class, i.e., there is no need to cycle permutations of labels. Furthermore, it can take into account multiport modules, multiconnector ports and loop structures. However, unlike the representation of Chen and Burdick, this representation is defined for static structures, i.e., extensions that address the kinematics or dynamics of the robot are not considered.

The basic Conro module, shown in Fig. 3.a, has four ports. Any of the ports of the passive head can be connected to the port of the active connector of another module in any of two orientations, "belly-up" or "belly-down". We denote the ports of the module as f_i and the orientations of the port as d_i.

We want to find a graph representation of the module that can be used to identify, without ambiguities, any legal robot configuration. In the connectivity-graph representation, a module is represented as a vertex (see Fig. 3.b) and a connection between modules, as an edge, as in the case shown in Fig. 2. This graph describes the topology of a robot but cannot be used to uniquely identify configurations with modules that can be connected to each other using different ports, i.e., if a module with n ports can be connected to any of

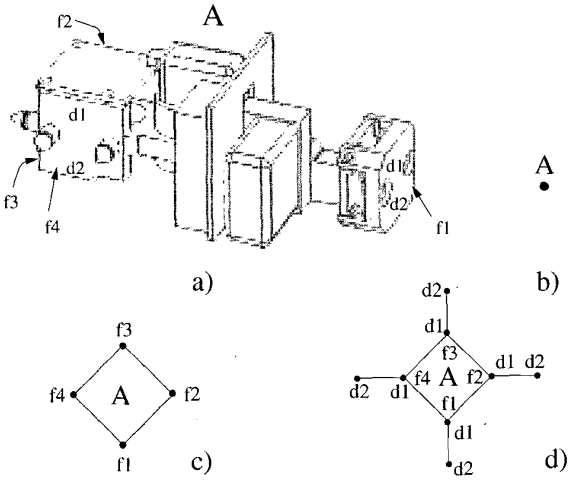

Figure 3: a) Ports and port orientations of Conro module and graphs that take into account b) connectivity, c) multiple ports and d) multiple-connection ports

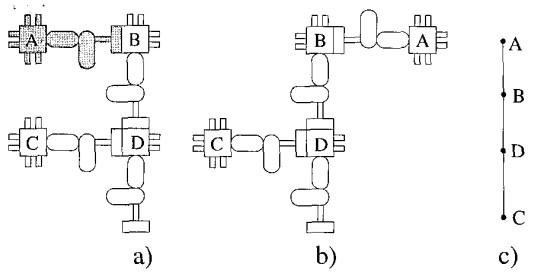

Figure 4: Top view of two Conro configurations and their connectivity graph

the n ports of another module, then there are $n!$ ways in which the two modules can be connected, many of which might be topologically different.

The Conro module is a multiport module and thus, we cannot use a connectivity graph to uniquely identify a configuration. For example, the two configurations shown in Figs. 4.a-b have the same connectivity graph shown in Fig. 4.c (a module has been colored for visualization purposes). A way to distinguish between configurations of multiport modules is to use the labels of the ports in the representation. In this case a suitable representation of the module is that shown in Fig. 3.c, where the module is represented by its ports, i.e., a vertex represents a port while an edge indicates that two ports either belong to the same module or belong to two connected modules.

Our goal is to find a graph-only representation of a robot that would allow us to distinguish between different configurations regardless of their labelings, i.e., the unlabeled version of the graph must be sufficient to identify the configuration. Unfortunately, the direct use of the labeled-graph representation of a robot with multiport modules leads to ambiguous unlabeled graphs. Figures. 5.a-b show the labeled graphs of the two configurations shown in Figs. 4.a-b. We can see that the unlabeled version of both graphs is the same.

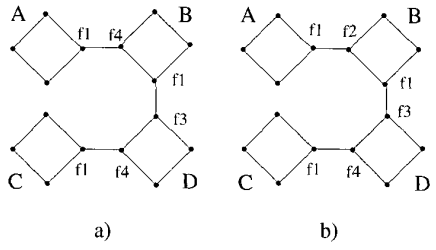

Figure 5: Labeled graph representation of the configurations shown in Figs. 4.a,b

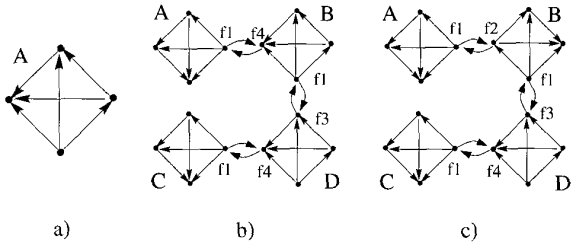

Figure 6: Multiport digraph representation of a) module and b-c) the configurations shown in Figs. 4.a-b

The problem of how to make the "label" itself a part of the graph can be solved in various ways. In our case, we mapped the labeled graph in Fig. 3.c to the digraph in Fig. 6.a by extending an edge from port f_i to port f_j for $i < j$, i.e., vertex f_i has $n - i$ edges leaving it or, equivalently, it has $i - 1$ edges arriving to it, where n is the number of ports of the module. This digraph unambiguously represents a multiport module because it is rigid, i.e., it has no automorphisms and thus, there is only one way to interpret it [11]. Hence, we can now represent robots built with multiport modules without ambiguity. For example, the two configurations shown in Figs. 4.a-b have the digraph representations shown in Figs. 6.b-c, respectively. An inspection shows that the corresponding unlabeled digraphs, indeed, are not isomorphic. The graph representation of Fig. 3.c preserves the symmetries of the module while the digraph representation of Fig. 6.a destroys it; modules that have partial symmetries can be represented using a combination of both representations.

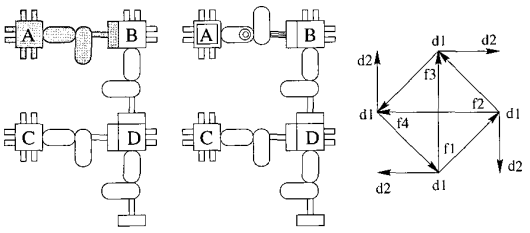

Figure 7: Top view of two Conro configurations and the module representation used to distinguish them

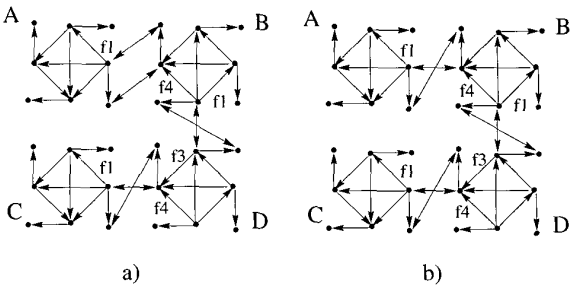

Figure 8: Multiconnection digraph representation of the configurations shown in Figs. 7.a-b

The same approach used to distinguish configurations of multiport modules can be used to distinguish configurations of modules with multiconnection ports. A multiconnection port is one that allows two ports of two modules to be connected in different ways. For example, the Conro modules can be connected to each other while they are both lying on their "bellies" or we can turn one of them "belly-up" and still have a legal physical connection. This situation is shown in Figs. 7.a-b where module A is attached to module B while both lying on its belly and after been turned upside-down. In this case, the multiport digraph representation of a module is insufficient to uniquely represent both configurations as both share the same such digraph, namely that of Fig. 6.b.

In the same way that we disambiguated multiport modules by representing the module as a set of ports, we can disambiguate multiconnection multiport modules by representing the ports as a set of connections, i.e., each possible connection in each port needs to be labeled. In the case of the Conro module, a given port of a module can be connected to a given port of another module in one of two relative orientations: belly-down or belly-up. Hence, each port has only two associated connections $d1$ and $d2$ as shown in Fig. 3.a. The corresponding labeled graph of this module representation is that of Fig. 3.d. As before, the labeled graph of the module is replaced by a digraph to make the label itself a part of the graph. This multiconnection digraph is shown in Fig. 7.c. Using this representation of the module, the configurations in Figs. 7.a-b are represented by the graphs shown in Figs. 8.a-b. which have nonisomorphic unlabeled graphs.

The module representations described in this section are quite general and can be used to represent other homogeneous modular robots. The extension to heterogeneous modular robots is straightforward as different types of modules can be labeled by attaching to a given vertex of each digraph a tag that identifies the type of the module (e.g., [11], pg. 7).

4 Configuration discovery

The configuration discovery process is used to obtain a representation of the robot configuration. In Section 3 we discussed three different module representations: the connectivity graph and the multiport and multiconnector digraphs. Since the Conro module is a multiport module, it is possible to have two configurations with the same topology (as shown in Figs. 4.a-b) and thus, we cannot use the connectivity graph to represent Conro robot configurations. On the other hand, although physically the Conro module is a multiconnector module, functionally it is not. Indeed, as shown in Figs. 7.a-b, a port can be connected to a module in either a "belly-up" or "belly-down" configuration. However, a working "belly-up" Conro module cannot be connected to a "belly-down" module because the modules would not be able to communicate, i.e., the infrared pairs will be paired emitter-to-emitter and transmitter-to-transmitter. Hence, the multiport digraph representation is all that is needed to obtain an unambiguous representation of a working Conro robot.

We can represent a digraph using its adjacency list or its incidence or adjacency matrix. In this paper we will use the adjacency matrix representation because it will help us with the identification of the configuration. The adjacency matrix P_A of the multiport digraph $G(A)$ of the module A shown in Fig. 6.a is

$$P_A = \begin{array}{c} \\ f1 \\ f2 \\ f3 \\ f4 \end{array} \begin{array}{c} \begin{array}{cccc} f1 & f2 & f3 & f4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \quad (1)$$

where $P_A(i,j) = 1$ if there is an edge from the i-th vertex to the j-th vertex [12]. In the case of Conro, a module can be connected to another using one and only one of its ports. For example, the following matrix C represents a two-module configuration in which

the port $f3$ of a module A is connected the port $f1$ of a module B (since the relation is symmetric, port $f1$ of module B is connected to port $f3$ of module A):

$$C = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

This matrix can be written as a block matrix as

$$C = \begin{bmatrix} P_A & S_{AB} \\ S_{AB}^T & P_B \end{bmatrix}, \; S_{AB} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \quad (2)$$

where the S_{ij} matrix indicates the connection between modules i and j. The general form of a configuration matrix of a N-module robot is

$$C = \begin{bmatrix} P_A & S_{AB} & \cdots & S_{AN} \\ S_{AB}^T & P_B & \cdots & S_{BN} \\ \vdots & \vdots & \ddots & \vdots \\ S_{AN}^T & S_{BN}^T & \cdots & P_N \end{bmatrix}$$

where $S_{IJ} = 0$ if modules I and J are not adjacent. A matrix S_{IJ} such that $S(u,v) = 1$ is denoted as $S_{u,v}$. Since Conro is a homogeneous robot, all its modules are identical and thus, $P = P_i$ for all i.

The discovery of the configuration matrix has three stages: the determination of the adjacency list of each module, the collection of these lists and the construction of the configuration matrix using the multiport digraph representation of a module. Assuming that each module has a unique identification (ID), then these processes need only two commands: GETID() and GETADJLIST(). The command GETID() received by a module through its i-th port asks the module to transmit its ID to the module attached to the same port i. The command GETADJLIST(m-ID, s-ID) requests the module with ID s-ID (i.e., the slave) to send its adjacency list to the module with ID m-ID (i.e., the master). On power up, each module sends a GETID() command through each of its ports and thus, finds the IDs of all the modules to which it is connected. This provides each module with a local adjacency list.

The second step of the configuration discovery, the collection of the adjacency lists of the modules, is done by the master. In order to put the process in a context, we will assume that the master is an external host computer that has communication with a module

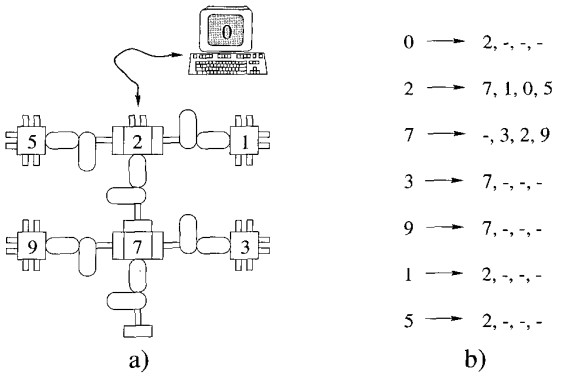

Figure 9: a) Quadruped and b) its adjacency list

of the robot through one of its ports, as shown in Fig. 9. To the modules, the host computer looks like a module with $ID = 0$. Since the host does not know in advance how many modules are in the robot, we use a linked list of dynamically allocated structures called *modules*. If *mod* is a *module* then *mod.ID* is a field that contains the ID of the module, *mod.port*[i] is a field that contains a pointer to the module connected to the i-th port of module *mod.ID* and *mod.idx* is a field that contains the position of *mod* in the list of *modules*. Since there is a one-to-one correspondence between a module and its representation, we will refer to both simply as the module. The collection of the local adjacency lists is done, in a depth-first search fashion, using the following pseudo-code.

```
BUILDADJLIST ()
1  masterMod ← NEWMODULE ( 0, 0 )
2  list ← APPEND ( Λ, masterMod )
3  id ← GETID ()
4  if id ≠ ∅ then
5     adj ← NEWMODULE ( id, 1 )
6     list ← APPEND ( list, adj )
7     list ← AUXADJLIST ( adj, list, 2 )
8     masterMod.port[1] ← adj
9  return list
```

The list is initialized with the module that represents the computer, the master module with $ID = 0$, in the 0-th position. In line 3, the master queries the serial port to see if it is attached to a robot. If there is a module adjacent to the master, a new module is created, initialized and appended to the list. The call in line 7 adds to the list all the modules attached to this new module. Finally, in line 8, the new module is set as the module connected to the master through the first port.

The pseudo-code of AUXADJLIST() is as follows.

```
AUXADJLIST ( mod, list, idx )
1  [id1,id2,id3,id4] ← GETADJLIST ( 0, mod.ID )
2  for i ← 1 to 4
3     if id_i ≠ ∅ then
4        adj ← INLIST ( list, id_i )
5        if adj = ∅ then
6           adj ← NEWMODULE ( id_i, idx )
7           idx ← idx+1
8           list ← APPEND ( list, adj )
9           list ← AUXADJLIST ( adj, list, idx )
10       mod.port[i] ← adj
11 return list
```

In line 1 the master gets the adjacency list of the module mod.ID. In line 3 the master determines if the i-th port of module mod.ID is attached to another module. If there is an adjacent module, in line 4 we search the list of modules to check whether this adjacent module is already in it or not. The command INLIST() returns a pointer to the module if the module is found. Otherwise, we create a new node, initialize it with the information of the adjacent module and append it to the list. In line 9 we add to the list all the modules attached to this newly-created module using a recursive call. At the return of this call, the node of the adjacent module is defined and, in line 10, we can create a pointer that describes the connection between modules mod and adj.

The result of running BUILDADJLIST() on a robot is an adjacency list of the modules of the robot. For example, running BUILDADJLIST() on the robot shown in Fig. 9.a generates the adjacency list shown in Fig. 9.b. Notice that the sequence in which modules were added to the list is consistent with a depth-first search.

The last step of the configuration discovery, finding the configuration matrix using the information from the adjacency list, only involves the master. This is done using the following pseudo-code:

```
BUILDCONFIGURATIONMATRIX (list)
1  for k ← 1 to LENGTH(list)
2     mod ← list[k]
3     i ← 4(k-1)
4     C[i+1:i+4][i+1:i+4] = P
5     for j ← 1 to 4
6        adj ← mod.port[j]
7        if adj ≠ ∅ then
8           r ← adj.idx
9           for s ← 1 to 4
10             if adj.port[s] = mod.ID then
11                C[i+j][4(r-1)+s] ← 1
12 return C
```

Running this algorithm returns a $4N \times 4N$ binary matrix where N is the number of modules of the robot including the master. For example, the configuration matrix of the robot shown in Fig. 9 is

$$C_1 = \begin{array}{c} \\ 0 \\ 2 \\ 7 \\ 3 \\ 9 \\ 1 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccccc} 0 & 2 & 7 & 3 & 9 & 1 & 5 \end{array} \\ \begin{bmatrix} P & S_{1,3} & 0 & 0 & 0 & 0 & 0 \\ S_{3,1} & P & S_{1,3} & 0 & 0 & S_{2,1} & S_{4,1} \\ 0 & S_{3,1} & P & S_{2,1} & S_{4,1} & 0 & 0 \\ 0 & 0 & S_{1,2} & P & 0 & 0 & 0 \\ 0 & 0 & S_{1,4} & 0 & P & 0 & 0 \\ 0 & S_{1,2} & 0 & 0 & 0 & P & 0 \\ 0 & S_{1,4} & 0 & 0 & 0 & 0 & P \end{bmatrix} \end{array}$$

where P is that of Eq. 1. Thus, consider the third row of C, i.e., the row that corresponds to the module with ID=7. The null entries of this row indicate that module 7 is not adjacent to modules 0 (i.e., the master), 1 or 5. The second entry, $S_{3,1}$, is equal to the block matrix S_{AB} in Eq. 2, and indicates that port 3 of module 7 is connected to port 1 of module 2. Likewise, the fourth entry of this row indicates that port 2 of module 7 is connected to port 1 of module 3 and so on.

The configuration matrix returned by BUILDCONFIGURATIONMATRIX() contains information about the master who might or might not be part of the robot. If the master is a module, then C_1 represents the configuration of the robot. However, if the master is an external computer that is passing as a module, as in the case shown in Fig. 9.a, then C_1 contains information regarding the relationship between the host and the robot that is not part of the robot configuration. This information is useful in certain contexts (e.g., scheduling of messages, balancing of network loads) but it is a liability when we are trying to identify the robot itself, whose configuration is independent of the particular relationship to the external host. In the latter case, the correct robot configuration matrix C_2 is obtained by removing the information of the host. For example, in the case of the robot in Fig. 9 we obtain

$$C_2 = \begin{array}{c} \\ 2 \\ 7 \\ 3 \\ 9 \\ 1 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 2 & 7 & 3 & 9 & 1 & 5 \end{array} \\ \begin{bmatrix} P & S_{1,3} & 0 & 0 & S_{2,1} & S_{4,1} \\ S_{3,1} & P & S_{2,1} & S_{4,1} & 0 & 0 \\ 0 & S_{1,2} & P & 0 & 0 & 0 \\ 0 & S_{1,4} & 0 & P & 0 & 0 \\ S_{1,2} & 0 & 0 & 0 & P & 0 \\ S_{1,4} & 0 & 0 & 0 & 0 & P \end{bmatrix} \end{array}$$

which is the principal submatrix of C_1 obtained by removing its first four rows and columns.

5 Configuration identification

Configuration identification is the process that determines the type of robot represented by a configura-

tion matrix. Consider the case where we have obtained a configuration matrix C that represents a robot and we want to compare it against the configuration matrices of a catalog, each one representing a robot that the master already knows how to control. The identification of the configuration can be done by simply permuting the rows and columns of the configuration matrix of the robot until it matches one in the catalog. This problem is exactly that of the classic graph isomorphism.

The graph isomorphism problem (GI) has not been solved and its complexity has not been placed either in P or NP. A way to address the graph isomorphism problem is by solving the computationally equivalent graph automorphism problem, using a program based on heuristics called Nauty [13]. This or any other heuristic used to solve GI can be used to match the configuration matrix of a robot against those of a catalog of known configurations.

6 Experimental results

Consider the case were we represent a configuration matrix with a feature vector. As features we could use the determinant of the matrix, its spectrum, the number of vertices or edges of the graph and any other graph or matrix feature that we desire. Then, we could compare the feature vector of the configuration that we are trying to identify against the feature vectors of the configurations in the catalog. Of course, if we knew of a feature vector that would uniquely identify a matrix and its permutations, we could solve GI. However, no such feature vector has been found. Thus, for any set of features, two matrices with different vectors are guaranteed to be non-isomorphic but two matrices with the same vectors are not guaranteed to be isomorphic. Still, feature vectors are, in practice, a reliable heuristic to determine isomorphism.

In our case, since we used an adjacency matrix to represent the digraph of the module, we have obtained a square configuration matrix and thus, we can use features such as its determinant and spectrum in the feature vector. We have used the described procedure to discover and identify Conro robots in quadruped, hexapod and snake configurations.

The complete setup for master-slave control of the robots requires not only the identification of the robot but a mapping between the modules of the robot and those of the cataloged configuration. This mapping is done with a heuristic that maps the configurations, first taking into account the number of connections to a given module (i.e., the degree of the vertex) and then inspecting the relative order of the adjacent modules as they are found in the adjacency list.

7 Summary and conclusions

We have presented a methodology to represent the configurations of modular robots using graphs and digraphs and have applied it to the Conro robot. Unlike other representations, the complexity of determining if two configurations are the same has been reduced to exactly that of the isomorphism-complete class.

Acknowledgments

This research is being sponsored by DARPA/MTO under contract number DAAN02-98-C-4032.

References

[1] C. Ünsal, H. Kiliççöte, and P. Khosla, "I(CES)-cubes: a modular self-reconfigurable bipartite robotic system," in *Proc. SPIE*, vol. 3839, pp. 258–269, 1999.

[2] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji, "A 3-D self-reconfigurable structure," in *Proc. IEEE Int. Conf. Robotics Automat.*, pp. 432–439, May 1998.

[3] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji, "Self-assembly and self-repair method for a distributed mechanical system," *IEEE Journal Robotics Automat.*, vol. 15, pp. 1035–1045, Dec. 1999.

[4] K. Kotay, D. Rus, M. Vona, and C. McGray, "The self-reconfiguring robotic molecule," in *Proc. IEEE Int. Conf. Robotics Automat.*, pp. 424–431, May 1998.

[5] M. Yim, J. Lamping, E. Mao, and J. G. Chase, "Rhombic dodecahedron shape for self-assembling robots," Tech. Rep. SPL-P9710777, Xerox PARC, 1997.

[6] M. Yim, "A reconfigurable modular robot with multiple modes of locomotion," in *Proc. JSME Conference on Advanced Mechatronics*, 1993.

[7] A. Castano and P. Will, "Mechanical design of a module for reconfigurable robots," in *Proc. IEEE/RSJ Intl. Conf. Intell. Robots Systems*, Nov. 2000.

[8] A. Castano, W.-M. Shen, and P. Will, "CONRO: Towards deployable robots with inter-robot metamorphic capabilities," *Autonomous Robots Journal*, vol. 8, pp. 309–324, July 2000.

[9] W.-M. Shen, B. Salemi, and P. Will, "Hormone for self-reconfigurable robots," in *Proc. Intl. Conf. Intelligent Autonomous Systems*, pp. 918–925, 2000.

[10] I.-M. Chen and J. Burdick, "Enumerating the non-isomorphic assembly configurations of modular robotic systems," in *Proc. IEEE/RSJ Intl. Conf. Intell. Robots Systems*, pp. 1985–1992, 1993.

[11] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem.* Boston, MA: Birkhäuser, 1993.

[12] N. Deo, *Graph Theory with Applications to Engineering and Computer Science.* Englewood cliffs, NJ: Prentice Hall, 1974.

[13] B. McKay, *Nauty User's Guide.* Tech. Rpt. TR-CS-90-02, Dept. Comp. Science, Australian Natl. Univ., 1990.