Journal of Southeast University (English Edition) Vol. 22, No. 2, pp. 185 – 190 June 2006 ISSN 1003—7985

Motion simulation and experiment of a novel modular self-reconfigurable robot

Wu Qiuxuan[1] Cao Guangyi[1] Fei Yanqiong[2]

([1]Research Institute of Fuelcell, Shanghai Jiaotong University, Shanghai 200030, China)

([2]Research Institute of Robot, Shanghai Jiaotong University, Shanghai 200030, China)

Abstract: Based on the character of the modular self-reconfigurable (MSR) robot, a novel homogeneous and lattice MSR robot, M-Cubes, was designed. Each module unit of the robot has 12 freedoms and is composed of six rotary joints and one cubic link. An attached/detached mechanism was designed on the rotary joints. A novel space transmitting system was placed on the inner portion of the cubic link. A motor separately transmitted torque to the six joints which were distributed equally on six surfaces of the cubic link. The example of a basic motion for the module was demonstrated. The result shows that the robot is concise and compact in structure, highly efficient in transmission, credible in connecting, and simple in controlling. At the same time, a simulator is developed to graphically design the system configuration, the reconfiguration process and the motion of cluster modules. The character of local action for the cellular automata (CA) is utilized. Each module is simplified as a cell. The transition rules of the CA are developed to combine with the genetic algorithm (GA) and applied to each module to accomplish distributed control. Simulation proves that the method is effective and feasible.

Key words: modular self-reconfigurable robot; structure design; motion simulation; distributed control

The modular self-reconfigurable (MSR) robot is a new concept that appeared in 1980s. The robots belong to the area of reconfigurable modular robots and combine the idea of autonomous mobile robots. MSR robots are metamorphic systems that can autonomously change their logical or physical configurations (such as shapes, sizes, or formations), as well as their locomotion and manipulation, based on the mission and the environment at hand. Because of their modularity, versatility, self-healing ability and low cost reproducibility, such robots provide a flexible approach for achieving complex tasks in unstructured and dynamic environments. They are well suited for applications such as search and rescue, reconnaissance, self-assembly, inspections in hazardous environments, and exploration in space and oceans. They also pose fundamental research challenges for robotics and other major branches of computer science, mechatronics and control theory.

Recently, many self-reconfigurable robots have been studied. They can be classified into two categories: chain-type robots and lattice-type robots. The chain-type robots are designed for fixed-shape locomotion and an occasional self-reconfiguration, mainly including: CEBOT[1], PolyBot[2], Conro[3], etc. The lattice-type is suitable for constructing various static configurations, but it is difficult to generate group motion. The lattice-type robots are designed mainly for self-reconfiguration, which include 3-D SR robot[4], Crystalline[5], I-Cube[6], Proteo[7], etc.

A modular robotic system (M-Cubes) has been designed, which has advantages of both the metamorphic capability and the generation of robotic motion. A simulator for this system that provides interactive graphical user interface (GUI) was developed to assist the design process of reconfiguration. By using the simulator, reconfiguration planning and motion generation can be designed and tested.

1 Module Design Principle

The robotic module unit has been proposed, which has a base cube at the center and six connecting joints attached to its six surfaces as shown in Fig. 1(a). Each joint can rotate independently, and has a connecting mechanism by which the unit can connect to or disconnect from its adjacent unit. By the connection, an arbitrary cubic structure can be realized.

The structure changes its configuration by means of the rotation of the connecting joint and changes in the unit connection. At the same time, a communication channel is connected. The unit can carry a neighbour unit by rotating one of its connecting joints. The former unit is called a carrier module and the latter a

Received 2005-12-16.

Foundation item: The National Natural Science Foundation of China (No. 50305021).

Biographies: Wu Qiuxuan (1978—), male, graduate; Cao Guangyi (corresponding author), male, professor, Caogy@sjtu.edu.cn.

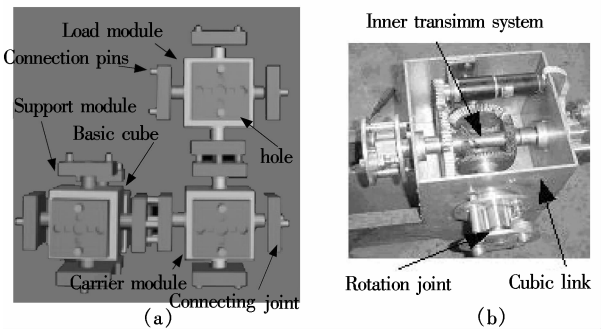

Fig. 1 Principle of M-Cubes. (a) M-Cubes unit and pair-wise motion; (b) Prototype of M-Cubes

load module in this motion (see Fig. 1). In the motion, the rotary connecting joint supported by another unit is called a support module. Both the carrier and the load module must release some of their other connections prior to the motion. A high-torque servomotor was embedded in the cube to control the rotary angles. Each connecting joint can rotate at arbitrary degrees. The M-Cubes provides a mechanical lockpin to connect each other in Fig. 2, which improves the coupling character and allows some errors in accuracy during approaching. The cone-shape mechanism guides the shaft during coupling. The principle of the attached/detached mechanism is as follows: On the one hand, the motion of connecting joints rotating independently comes from the output shaft of the basic cube, but the motion is controlled by a clutch switching on/off; on the other hand, the joint is composed of screws and nuts, the other clutch is utilized to switch on/off motion from rotation to line of pins.

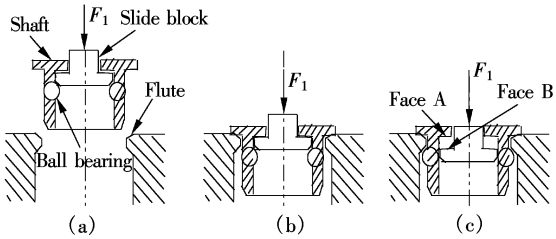

Fig. 2 Attach/detach mechanism of rotation joint. (a) Alignment; (b) Plugging; (c) Locking

Connection pins consist of slide block, shaft and ball bearing as illustrated in Fig. 2 (a), and the slide block connects with the nut. While the screw rotates and the nut moves, the nut pushes the slide block in a rectilinear motion. Contact face A and face B is a permanent magnetic whose polar gender is opposite.

When the joints are connected, the motion processes of the shaft are illustrated in Fig. 2 (b), the process is divided into two steps: ① When two joints are ready to connect, the slide block and the shaft together move because of the magnetic action of the contact face A and face B opposite polarity. ② When the shafts have been plugged into the flute, the shaft encounters the bar of flute end-face and stops forward. The slide block gets over the magnetic force between the contact face A and face B and continues to go forward. The front-end bevel of the slide block pushes and presses the ball bearing embedded in the barrel and makes the ball bearing move radically to fulfill locking of the pins and the holes.

When modules separate, the motion process of the connecting shaft is reversed to the connecting process. First, the slide block moves backward and the ball bearing relaxes the shafts, and then the slide and the barrel together move backward to accomplish the separation between the shafts and the flutes. Though a unit cannot change its position by itself, reconfiguration is possible by this pair-wise motion. Note that each unit's role is not fixed but changes dynamically during the reconfiguration process. A prototype is shown in Fig. 1(b).

2 Simulator of MSR Robot

2.1 Function of simulator

An interactive simulation system is developed as shown in Fig. 3. A sequence of reconfiguration is designed, and several motions to the modules are given. The simulator is developed on a PC with Windows 2000/XP using Java/Java 3D. The main functions of the simulator are as follows:

① Graphical display of the process. Motion of the modules is animated;

② Java editing of initial configuration;

③ Java editing of the motion sequence;

④ Display of the modules' state;

⑤ GUI facilities: including zooming in and out, changing the view point, and toggle change of visibility of other objects such as floors and obstacles;

⑥ The users can design any initial configuration by indicating the position and orientation of each module using simple mouse operations. Then the simulator allows users to design a sequence of module motions in a java file or text file. The interface checks the connectivity of all modules and alerts the designer if some system parts are disconnected. Collision of modules is

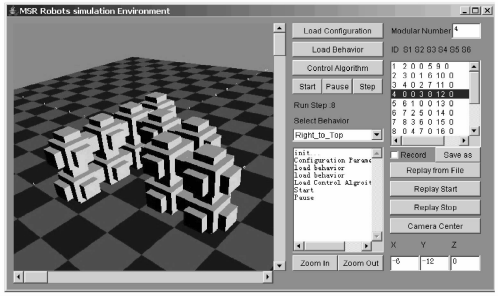

Fig. 3 Overview of simulator

also checked automatically.

2.2 Principle of distributed control

Distributed control is scalable, supporting parallelism and is better suited for operation in unstructured environments. Distributed algorithms are naturally suited for controlling self-reconfiguring robots because they take advantage of modularity, allowing the system to be more robust to failures of individual modules and communications and supporting partitionings of the robot.

Cellular automata (CA) represents a system of distributed, locally interacting cells that evolves according to a set of rules. Based on these micro, local interactions with no global coordination, a macro or global behavior is produced[8]. Such a pattern is readily observed in nature and the universe in general.

CA can be thought of as a model of naturally existing systems. Natural systems, through the evolution, have produced highly complex systems showing globally coordinated information processing, with no global coordination. In contrast to natural systems, today many systems utilize "global criterion" that requires global coordination. We will show that, through the evolution of a cellular automaton, the global criteria of reporting cell problems are translated to local transition rules of cellular units. The cellular automaton is evolved by using genetic algorithms (GAs)[9]. Specifically, the CA transition rules are found, or evolved, using the genetic algorithm.

We represent M-Cubes robot as a collection of cells, each cell corresponding to one robot module. We view the resulting structure as a particular type of CA. The control algorithm is simply a set of local rules that run on each cell within the system. Each rule is defined by a set of preconditions on the neighborhood of the cell and a set of actions that are executed if the preconditions are satisfied. We represent the basic module of the robot as a cube, but our proposed abstraction can be replaced by other geometric structures that support the formation of lattices. Each module is assumed to be able to translate across the joint of other modules as well as a transition to other planes of motion. It is also assumed that each module is able to examine or query its neighbors and determine the presence or absence of a neighbor on all sides. Preconditions may include specific geometric configurations of neighbors and empty space, or given values of a module's internal state. The actions of a rule can include the motion of the cell, an update of an internal state, or both.

This paper presents the first distributed implementation of locomotion and self-reconfiguration on lattice-type system based CA. We propose to develop a parallel and distributed cells' planning algorithm based on a recently emerging and promising techniques of combining evolutionary computation and CA to create an "evolving" CA system. Recent results suggest that highly parallel and distributed algorithms can "evolve" using such a hybrid system, solving complex problems.

The state of the modular system is described by an array of all the module states. Each module is labeled by ID $(0\ldots n)$. Each module shape and state are defined as a class file having members as the position and the orientation of the connecting joint between the cubic and the connecting joint. By using the editor of the simulator, an arbitrary initial configuration of any number of modules is graphically designed and then stored as a text file or a java file.

The simulator checks whether all the modules can be connected, but it does not check whether each command motion causes collision between modules. By using the simulator, a program is designed and stored as a java file. Then, the simulator processes each program step at each "simulation time step", updates the states of modules and displays the motion of the modules as animation.

2.3 Activation model

Each module uses the same rule sets and independently evaluates the rules and carries out the specified action resulting from a successful rule. In traditional CA, all cells are evaluated together, so that the next configuration is generated from the current configuration by simultaneously evaluating all the cells. This model is not concerned with the physical aspects of the underlying system. Since real robots are physical systems, we modify the evaluation model for traditional CA to a sequential cell evaluation process, in which only one cell is evaluated at a time. We can easily ensure no collisions in simulation by evaluating the cells one at a time, while in hardware a local locking system is required.

An evaluation model is defined based on the relative delay of the activation of the cells. In the model, a cell can delay activation at most one cycle relative to any other cells in the system, or equivalently, one cell can activate at most twice before another cell activates once. The model can be implemented in simulation using the algorithm, although in a real robot the evaluations would be timed implicitly by the system. Even if software synchronization were possible, it could represent a significant communication cost, so the asynchronous algorithms may be preferable for systems where the modules have variable speed to avoid this overhead[10].

2.4 Software implementation

The software not only accomplishes simulating the

change in module position in general, but also fulfills metamorphism from initial configuration to goal configuration through attaching and detaching the modules. So in the simulation environment, the software simulates the components of a module to the best of its abilities and the motion of the system. Since Java and Java 3D have the characteristic of object oriented design (OOD), we decompose, separately, module, simulation process and user interface, creating corresponding classes and define particular structures of the module and the mode of motion. The purpose of the framework is to simplify the development of the simulation of modular robots. It must be general, efficient and reusable and easy to use.

The two central classes are module and simulation. The first one is abstract and must be a sub-classed for creating a new type of module. The second one is a singleton which deals with the Java 3D world (creation, stepping) and handles the collision detection with dedicated callback functions. A module has one and only one controller which must be a sub-class of the abstract class controller. This controller will calculate the new position of the module at each step. These values must be updated at each step by the module or the simulation or anything else.

The serializer abstract class offers the possibility to read/write from a file, the structure of the whole robot. As the information needed strongly depends on the module, it is necessary to write a specific sub-class for each type of module. Several classes offer some useful services like the capabilities of loading from an .obj file (it will be created using Proe) supplied by GeometryLoader or a lot of functions for the matrix calculations or the I/O from/to a file supplied by the utilities class[11].

The unit module was decomposed into module entity, connector and controller. Module class, connector class and controller class were created. Module class finished the calculation and storage of position of the module and connector utilizing Java 3D and the judge of self-status and motion. Controller class defined the function of module controller, including justifying the connecting status, based on communication messages module controller planning the path of module and produced motion sequence. On the one hand, the connector class accomplished the estimation and storage of connecting status, and on the other hand, it accomplished sending, receiving and storage of connector messages. The framework of class is shown in Fig. 4 (Rectangles are framework classes; rhombuses are examples for a new module; ellipses are files used by the software).

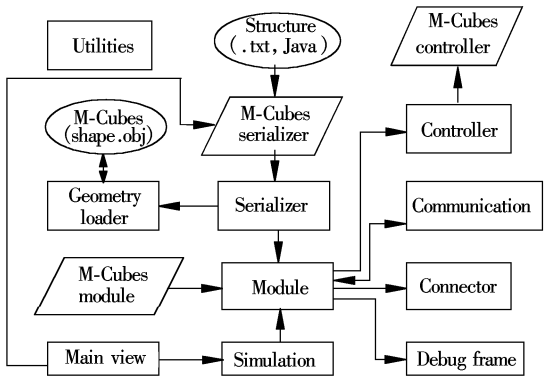

Fig. 4 Design of the framework

The simulation class performs main action including constructing the M-Cubes system, the initializing system and the real-time simulation. User interface is divided into main view and debug frame. The main view class is the viewer that is responsible for visioning and listening to simulation process. The debug frame class defines a pop-up dialogue which displays connecting status and debug information. The software has two threads: the main thread MSR3D responsible for GUI and displaying motion process, the other thread MSREngine responsible for producing motion sequence. The flowchart of the main thread and the simulation thread are shown in Fig. 5.

3 Reconfiguration and Motion Procedure

3.1 Basic motions

Though each module has limited degrees of freedom, a cluster of the modules has the ability of reconfiguration. Two basic motions are introduced here to explain the reconfiguration process. For example, we assume that the motions are carried out on a plane tiled by many other modules.

The basic motion of M-Cubes is implemented by the rotation of the connecting joints, and it has overturn and level movement. From the same view point, the level movement is the motion in which the moving module runs on the surface of other modules; overturn means that the moving module rotates to the surface of other modules; the motion needs the help of other modules to accomplish the motion, and the module can be divided into a load module, a carrier module and a support module according to the function of the modules. In Fig. 6(a), M_3, M_1, M_2 are all cell modules, if M_1 wants on the top surface of M_2, then M_1 is the load, M_2 is the carrier, M_3 is the support. By connecting M_1 and M_2, connecting M_3 and M_2, M_1 can be accomplished on the top surface of M_2 by the connecting joint of M_3 rotating 90°. By the same method, M_3 is also on the top surface of M_2. This is the principle of the level motion in Fig. 6(b). But the topology structure of the

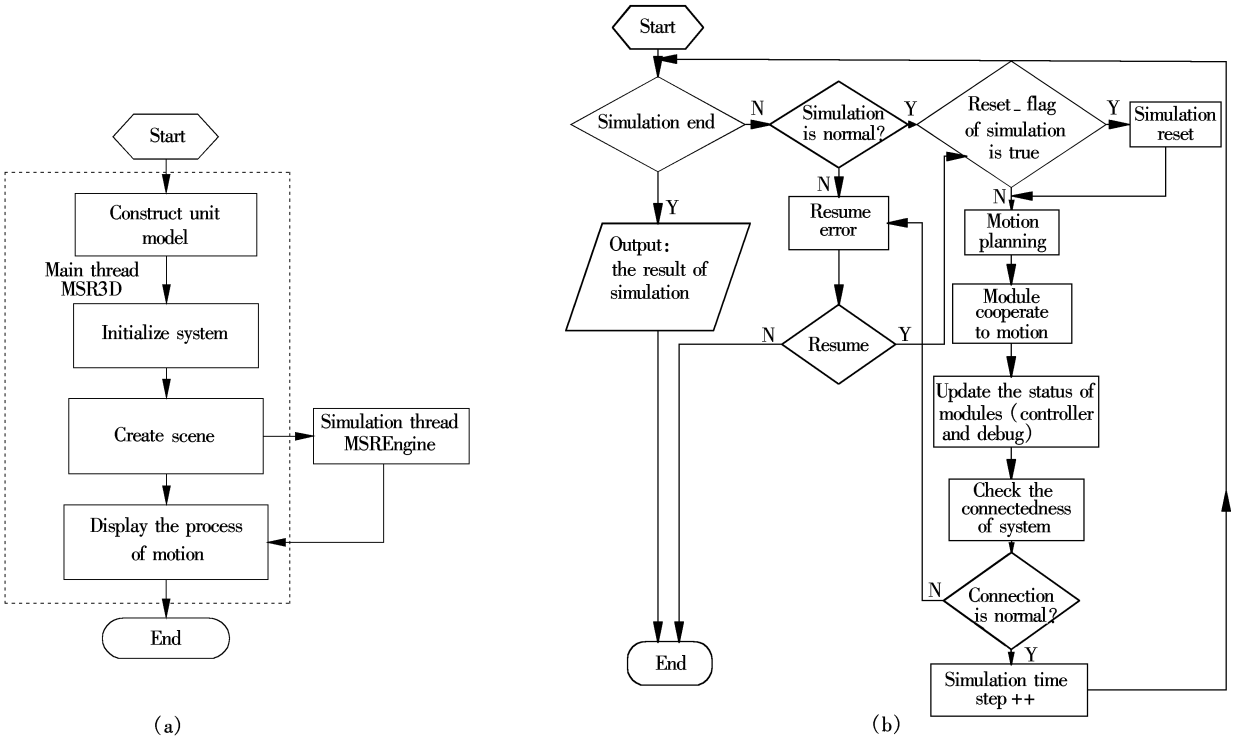

Fig. 5 Flowchart of simulation. (a) Main thread; (b) Simulation thread

robot must maintain connection during the process of motion. The module has the action of connecting and detaching. Connecting means that the pins of two neighbor connecting joints plug each other into the holes of the others and the self-lock mechanism maintains the pins fixed position. The two neighbor connecting joints are coupled as a body. On the other hand, detaching is the inverse action of connecting with the two neighbor connecting joints being decoupled in favor of a single module motion. By combining these basic motions, modules can reconstruct in a variety of structures.

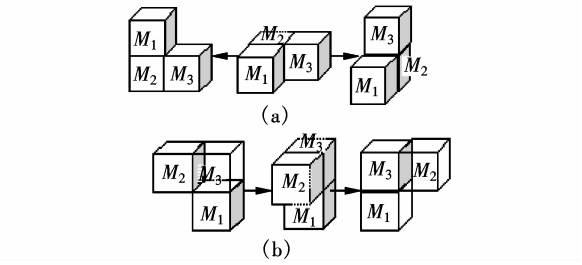

Fig. 6 Way of motion. (a) Module overturn (b) Module level move

3.2 Experiment

Examples of hand-coded reconfiguration processes are shown in Fig. 7, in which a robot shapes itself from a plane cluster of nine modules to a tower of eight modules after eight steps. Note that, in the simulation, we do not assume the floor tiled by other modules.

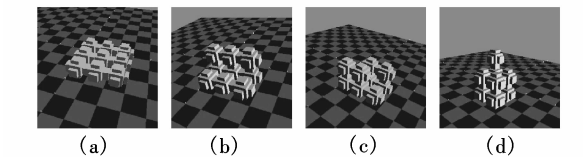

Fig. 7 Stationary self-reconfiguration. (a) Step 0; (b) Step 3; (c) Step 5; (d) Step 8

We have built five modules and finished the experiment of overturn and attaching/detaching. Shown as Figs. 8 (a) and (b), the process of a module overturned is demonstrated; Figs. 8 (c) and (d) show the process of two adjacent joints attaching and detaching.

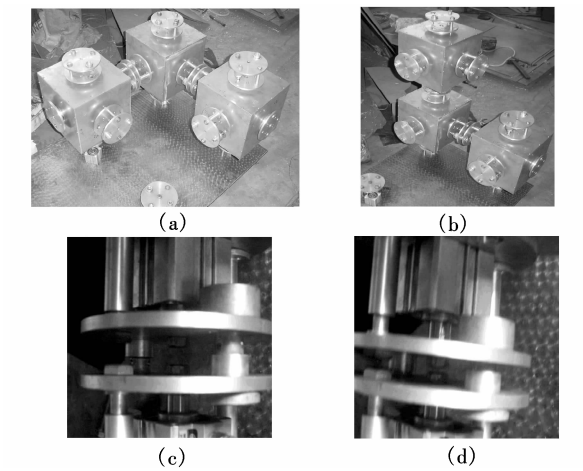

Fig. 8 Experiments of overturn and attach/detach. (a) Initial position; (b) Position of overturning 90°; (c) Plugging; (d) Locking

4 Conclusion

A novel homogenous modular self-reconfigurable robot M-Cubes is presented. It has the capabilities of both reconfiguration and motion. A graphical simulator for the MSR robots system has been developed and the relation of classes and their threads are analyzed. A complicated example has been planned using the simulator. Five modules are developed. Basic experiments of hardware modules are conducted to show the feasibility of the system. It is not difficult to modify the simulator to generate actual signals to these controllers according to the designed sequence.

References

[1] Fukuda T, Kawaguchi Y. Cellular robotic system as one of the realization of self-organizing intelligent universal manipulator [A]. In: *Proc of the IEEE Intl Conf on Robotics and Automation* [C]. Cincinnati, USA, 1990. 662 – 667.

[2] Yim M, Zhang Y, Roufas K, et al. Connecting and disconnecting for chain self-reconfiguration with PolyBot [J]. *IEEE ASME Transactions on Mechatronics*, 2002, **7**(4): 442 – 451.

[3] Castano A, Will P. Mechanical design of a module for reconfigurable robots [A]. In: *Proc of the IEEE Intl Conf on Intelligent Robots and Systems* [C]. Takamatsu, Japan, 2000. 2203 – 2209.

[4] Kurokawa H, Murata S, Yoshida E, et al. A 3-D self-reconfigurable structure and experiments [A]. In: *Proc of the IEEE Intl Conf on Intelligent Robots and Systems* [C]. Victoria, Canada, 1998. 860 – 865.

[5] Rus D, Vona M. Crystalline robots: self-reconfiguration with unit compressible modules [J]. *Autonomous Robots*, 2001, **10**(1): 107 – 124.

[6] Unsal C, Khosla P K. Mechatronic design of a modular self-reconfigurable robotics system [A]. In: *Proc of the IEEE Intl Conf on Intelligent Robots and Systems* [C]. Takamatsu, Japan, 2000. 1742 – 1747.

[7] Yim M, Zhang Y, Lamping J, et al. Distributed control for 3-D metamorphosis [J]. *Autonomous Robots*, 2001, **10**(1): 41 – 56.

[8] Kondoh S, Tomiyama T, Umeda Y. Morphological design of a manufacturing system based on the concept of cellular machines [A]. In: *Intelligent Autonomous Systems* 6 [C]. Italy: IOS Press, 2000. 926 – 933.

[9] Simon M, Sean P. An overview of genetic algorithms for the solution of optimization problems [J]. *Computers in Higher Education Economics Review*, 1999, **13**(1): 16 – 20.

[10] Butler Z, Kotay K, Rus D, et al. Generic decentralized control for a class of self-reconfigurable robots [A]. In: *Proc of the IEEE Intl Conf on Robotics and Automation* [C]. Washington, DC, 2002. 809 – 815.

[11] Barthelemy H. Framework for simulating modular robots and self-organization of locomotion under water [D]. Lausanne: Swiss Federal Institute of Technology, 2005.

一种新型模块化自重构机器人的运动仿真和试验

吴秋轩[1]　曹广益[1]　费燕琼[2]

([1] 上海交通大学燃料电池研究所,上海 200030)
([2] 上海交通大学机器人研究所,上海 200030)

摘要:根据自重构机器人的特点,设计了一种新型的同构阵列式自重构机器人 M-Cubes,其每个单元模块由 6 个旋转关节和 1 个立方体连杆组成,具有 12 个自由度,旋转关节上设计了一种机械式的连接分离机构,连杆内部设计了一种空间传动系统,用一个电机分别带动 6 个空间均布的关节旋转,机构整体结构上更加简洁、紧凑. 对设计的模块进行的基本运动试验表明:传动更加高效,连接分离更加可靠,控制更加简单方便. 同时开发了一个自重构机器人仿真平台,可以图形化地设计系统的构型、模块的运动和系统的重构过程. 利用元胞自动机的局部作用特性,将每个单元模块简化为元胞,结合遗传算法来进化元胞自动机的转移规则,将转移规则作用于每个单元模块,实现分布式控制,仿真结果表明该方法是有效和可行的.

关键词:模块化自重构机器人;结构设计;运动仿真;分布式控制
中图分类号:TP242