Submitted to *SPIE Photonics East 2000, Sensor Fusion and Decentralized Control in Robotic Systems III*, 5-8 November 2000, Boston, MA.
Published in *Proceedings of SPIE, Sensor Fusion and Decentralized Control in Robotic Systems III*, vol. 4196, pp. 388-401, 2000.

Solutions for 3-D self-reconfiguration in a modular robotic system: implementation and motion planning

Cem Ünsal* and Pradeep K. Khosla

Institute for Complex Engineered Systems
Carnegie Mellon University, Pittsburgh, PA, 15213

ABSTRACT

In this manuscript, we discuss new solutions for mechanical design and motion planning for a class of three-dimensional modular self-reconfigurable robotic system, namely *I-Cubes*. This system is a bipartite collection of active *links* that provide motions for self-reconfiguration, and *cubes* acting as connection points. The links are three degree of freedom manipulators that can attach to and detach from the cube faces. The cubes can be positioned and oriented using the links. These capabilities enable the system to change its shape and perform locomotion tasks over difficult terrain. This paper describes the scaled down version of the system previously described in [1] and details the new design and manufacturing approaches. Initially designed algorithms for motion planning of I-Cubes are improved to provide better results. Results of our tests are given and issues related to motion planning are discussed. The user interfaces designed for the control of the system and algorithm evaluation is also described.

Keyword: self-reconfiguration, modular robots, reconfiguration planning, mechatronics, collective robotics

1. INTRODUCTION

The concept of a modular self-reconfigurable robot is a relatively new concept that extends ideas on reconfigurable robotic manipulators, modular robots, and mobile robots. Reconfigurable robots are robotic systems that can change their shape by re-locating their modular elements. The main idea behind modular reconfigurability in robotics is to design robots that are capable of locomotion in difficult terrain by extending currently available stable gaits such as using wheels, and treaded systems. On the other hand, the reconfigurability characteristics will provide a system that can adapt to its environment or to the task. Recent ongoing advances in electronics, MEMS, scale and power of computational elements and mechanical components forces the research on mobile robots toward new ideas such as self-reconfigurable systems that are mechanically feasible and small in size.

We envision a modular self-reconfiguring group of robots that consists of two modules with different characteristics. A sufficient number of modules combined as a single entity would be capable of self-reconfiguring themselves into defined shapes, which in turn would provide a new type of locomotion gait that may be combined with other capabilities. Scenarios involving such robots include a snake-like robot traveling inside a tunnel, a legged robot moving on uneven terrain, a climbing robot moving over obstacles, flexible manipulators without a fixed base for space applications, and extending structures for bridge formation.

There are several approaches to design and implementation of modular robotic systems. Early examples included manually configured manipulator or mobile systems with homogeneous or heterogeneous modules. Manipulators that can be designed according to task specification called *RMMS* [2], modularly synthesized kinematics structure family of *Tetrabots* [3], and *CEBOT* cellular system as self-organizing manipulators [4] have been proposed and realized. Recently, several prototypes of modular self-reconfigurable systems have been implemented by various researchers. Existing 2-D systems include *metamorphing hexagonal modules* [5], *self-repairing machine* [6] and *Crystalline robot* [7] moving in horizontal plane, and Inchworm [8], and self-organizing robots [9] moving in vertical plane. Modular systems that can self-reconfigure and move in 3-D include *Polypod/Polybot* [10] and *CONRO* [11] that combine different gaits, and robotic *molecule* [12], self-reconfigurable structure [13], *Proteo* [14] and *I-Cubes* [15] that are capable of changing shape using neighboring modules.

Most of these systems are homogeneous with the exception of Polypod/Polybot and I-Cubes, which consist of two different types of modules. According to the definitions given in [16], metamorphing modules, Fractum, crystalline robot, robotic molecule, self-reconfigurable structure, and Proteo use substrate reconfiguration where the modules are placed on a lattice. On the other hand, Inchworm, Polybot, and CONRO employ closed-chain reconfiguration in which "modules meet at common branch-out points to form complex configuration." The motion of I-Cube modules is closer to substrate

* Correspondence: E-mail: unsal@ices.cmu.edu; WWW: http://www.cs.cmu.edu/~unsal/research/ices/cubes

reconfiguration when the cube motions are considered. However, it is possible to view the system as a closed-chain configuration where some of the links acts as 3-DOF legs of a walking system.

Modular robotic systems have the advantage over large and complex robots. The units making up the system can be mass-produced at a lower cost. A large system consisting of many elements is also less prone to catastrophic failures due to its capability of removing the malfunctioning elements from the group and reconfiguring itself, sometimes by sacrificing additional modules [17]. Although the large number of modules guarantees there will be at least one module failing after a specified time, the same number would provide some redundancy that is useful in overcoming the problem [10]. In order to have modular self-reconfigurability, the system must have several essential properties, such as geometric, physical, and mechanical compatibility among the individual modules. Furthermore, several design issues need to be considered for the system to be autonomous. Essential properties of our particular system as well as design issues relating to the implementation are given in following sections.

The system described here is a self-reconfiguring bipartite system that separates components providing actuation for self-reconfiguration and components providing computation, sensing and power resources as well as other types of stable gait mechanism such as wheeled locomotion. In the next section, we introduce the system. Section 3 discusses the latest mechatronic design, while Section 4 introduces the user interface and communication protocol for motion planning and control. Section 5 defines our approach and current solutions to motion planning with Section 6 concluding the manuscript.

2. I-CUBES: SYSTEM DESCRIPTION

I-Cubes are a class of modular self-reconfiguring robotic system that is bipartite, i.e., a collection of independently controlled three degree of freedom manipulators, called *links* and connection elements called *cubes* not directly contributing to self-reconfiguration motions (Figure 1). Links are capable of connecting to and disconnecting from the faces of the cubes; they can therefore move (i) from one cube face to another, (ii) from one cube to another, and (iii) move the cubes. This group of links and cubes for a dynamic pseudo-graph where the links are the edges, and the cubes are the nodes. The graph may change after a link motion. The overall system should have the following properties:

- Modules can be independently controlled; only the cube attached to the non-fixed end of a moving link is affected by a link motion.
- All modules have the same characteristics and are mechanically and computationally compatible. Links and cubes are capable of permitting power and information flow to the modules they are attached.
- The connection point between the cubes and the links need to be mechanically feasible to hold multiple elements together.
- The 3-D structure formed by the cubes and the links fits a cubic lattice to guarantee interlocking of the neighboring modules, i.e., the distance between cubes while in a position to accept a link is exactly *d*.
- Links have sufficient degree of freedom to complete motions in 3-D.

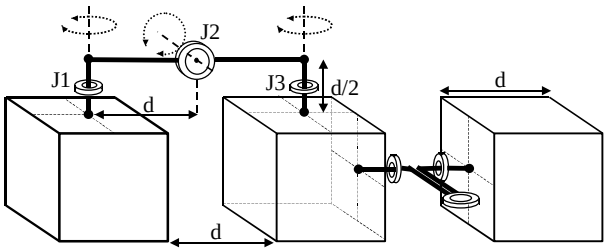


Figure 1. Geometric constraints and joint definitions for the links.

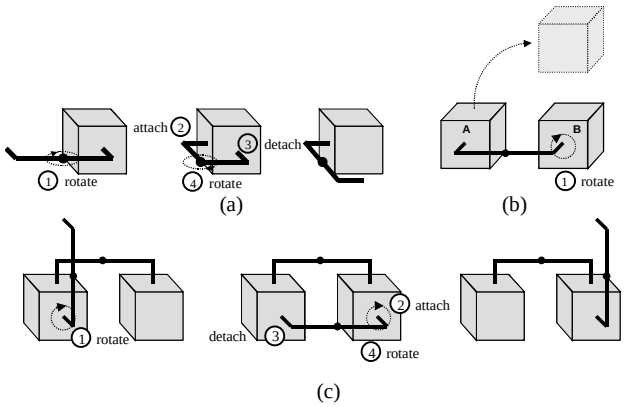


Figure 2. Possible moves for the links.

As seen in Figure 1, the size definitions for the links and the cubes are dependent. If the length of a cube edge is *d*, then the links should have four sections of length *d/2* and *d* as shown if the figure. The three rotational degrees of freedom for the links are provided by the joint J2 at the middle, and the joints located at end segments (J1 and J3). Joints J1 and J3 are both capable of providing continuous 360-degree rotations, while J2 can only rotate 270 degrees. In order to fit the cubic lattice formed by the modules, the links must be able to fully open to provide the required distance *d* between cubes attached to the link. A cube consists of six faceplates with attachment points for link connectors. Cubes do not contribute to the self-reconfiguring motions with the exception of the motion to lock the link connector in place. A cube attached to a link can be (i) rotated, (ii) translated in vertical or horizontal plane, or (iii) act as a pivot point for a moving link. The role of the cube

depends on the position and motion of the active link as well as the connections formed by all modules. We have designed our system to overcome some of the common problems in modular self-reconfiguration. Capabilities such as energy-efficient actuation, on-board power resources, information and power transfer between elements and autonomous motion are required for a feasible practical implementation and were considered during system design. Section 3 gives the details of the design for the latest version of the I-Cubes system.

2.1. Self-reconfiguration and possible scenarios

Since the actuation for self-reconfiguration is provided by the links, cubes can be used to provide computation, sensing and power resources. If the modules are designed to exchange power and information, the cubes may be equipped with batteries, microprocessors and sensors to be the decision-making elements while the links become the 'muscles' of the system. It is also possible to remove some of the attachment points on the cubes to replace them with wheeled or threaded systems for faster locomotion. Specifically, we envision small robots that can reposition themselves to form a group that is capable of self-reconfiguring in order to move over obstacles that a single robot cannot overtake. Similar scenarios that combine different gaits with shape reconfiguration include stair climbing and traversing pipes. Figure 3 gives snapshots of a possible scenario for a 4L4C group (four links and four cubes). This group is capable of moving to a higher surface by reconfiguring itself. Connections between elements are kept such that the system forms a single connected graph at any given time. There are several intervals where multiple links can move simultaneously. Cube faces that are initially on the ground and several others are not used during reconfiguration. Black circles indicate the faces that are initially oriented upward. The faces with circles are still oriented upward at the end of the sequence. To guarantee this result, the cubes need to be re-oriented in earlier phases of the solution, as seen in snapshot 2.

Another example given in Figure 4 illustrates the combination of self-reconfiguration with faster locomotion capabilities. Since the cubes do not contribute to the reconfiguration motions with the exception of the locking mechanism between the modules, they can be equipped with modules providing different gaits. In Figure 4, the leftmost robot in the first image has a sensor, but is incapable of seeing what is behind the obstacle. These wheeled robots, if they can be moved to a position to form a connected group, can self-reconfigure to lift the sensor-equipped robot. For a given initial configuration, it is possible to find a sequence of link motions that will move the robot with the sensor over the obstacle. The links can be used help the robot to scan its environment with its sensors, or the robot can detach from the system to continue its surveillance mission. It is also possible for a larger group to self-reconfigure into a tower in order to lift the surveillance robot to see beyond the relatively larger obstacles [1].

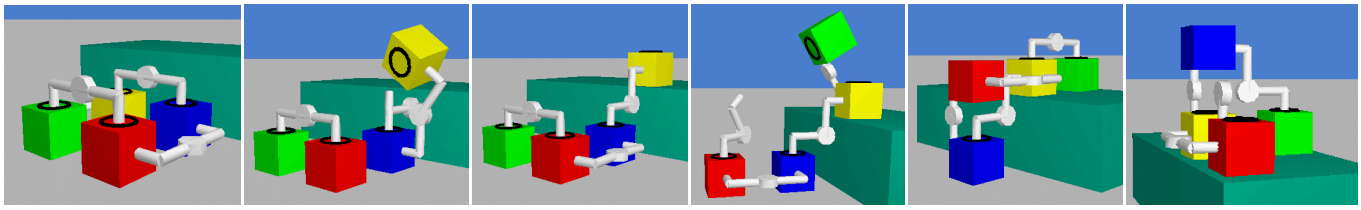


Figure 3. 4L4C group self-reconfiguring to climb an obstacle.

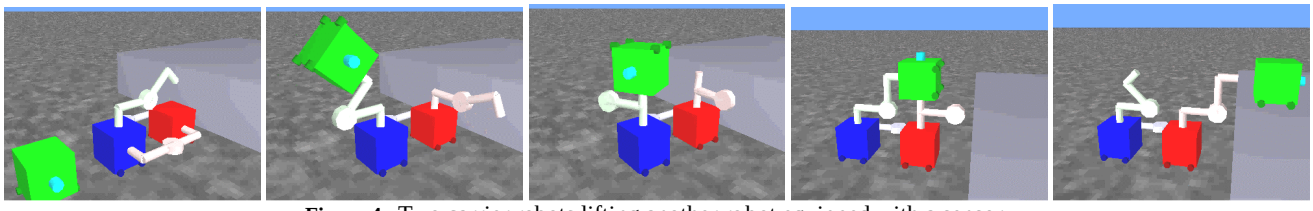


Figure 4. Two carrier robots lifting another robot equipped with a sensor.

Our previous prototypes included link and cube bodies that are manufactured using a fused deposition modeling (FDM) machine with a cube size of d = 8cm. The details of this design and the experiments are given detailed in our previous manuscripts [1, 15]. We re-iterated our design in order to scale down the module size and obtain modules that are more capable. Our initial experiments also showed that our tests on hardware implementation and software for control and planning algorithms required a more intuitive and capable graphical interface. Initial results of current iteration are given in the following sections.

3. MECHATRONIC DESIGN

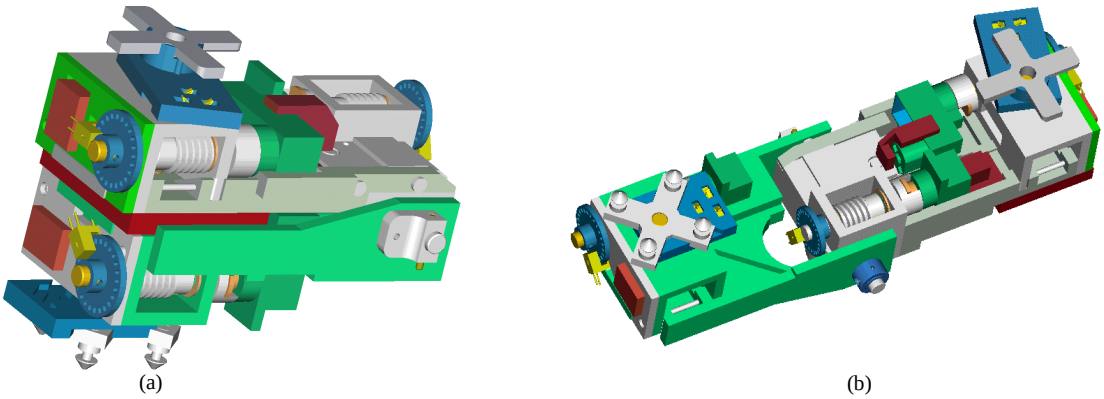
Previous implementation of the I-Cubes (See Figure 5 for a 1L1C experiment) was similar to our last prototype. However, it lacked several important features for complete reconfiguration capabilities. Our hardware experiments indicated that the idea was feasible even for a smaller size, if several design changes could be made. These changes include (i) scaling down the whole system from 8cm to 6cm, (ii) redefining the attachment mechanism on the cubes and the drive mechanism on the links

for the new scale, (iii) adding a motion controller circuitry that fits the space available on the link, and (iv) adding the communication and power connections between the modules.

Figure 5. A link moving from one cube face to another.

Similar to previous design, the links have three worm gear mechanisms driven by small servos to provide continuous rotation at the end joints, and 270-degree rotation and the middle joint (Figure 1 and Figure 6). The servos are coupled to the worm shafts using plastic couplers. At the end joints, the worm wheels are aligned with the cross shaped connectors that attach to the cube faces. At the middle joint, the servo and the worm are located on one side of the body, while the wheel shafts is coupled with the other side. The normal distance from wheel shafts at the ends to the wheel shaft in the middle is equal to $d = 6cm$. The potentiometer feedback and the mechanical stops on the servos are removed. The control circuitry inside the servo is also removed in order to fit the servos into the new smaller link design. The motors driving the servo gearboxes are now controlled via H-bridge driver ICs connected to the PIC 16C63A microprocessor running at 20Mhz.

Scaling down the size for the links enabled us to lower the maximum torque requirement at the output of the gear mechanism. Still assuming 15% efficiency for the worm gear system, we have chosen a new gear ratio of 1:30, which in turn enabled us to choose a worm with smaller diameter. Mechanical encoders and the rotary for position feedback are now replaced with optical encoders with 24 slots on the wheels. The plastic encoder wheels are machined from plastic stock. Custom design made the attachment of an encoder wheel to the middle worm shaft instead of the worm gear possible, increasing the feedback signal's resolution significantly. With the given specifications and counting both rising and falling edges of the encoder signal, the feedback signal has a resolution of 0.25 degrees.



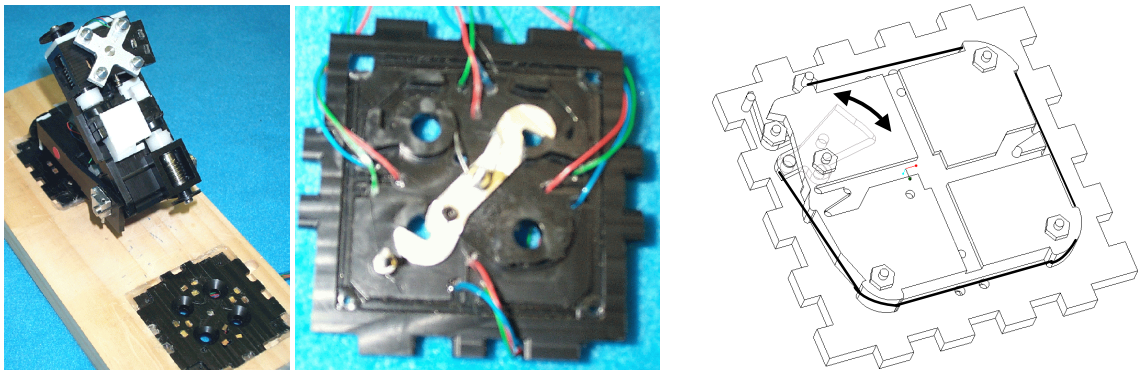
(a) (b)

Figure 6. ProE drawings of the 6cm link: (a) idle joint fully open and (b) at zero degrees.

While updating our previous attachment mechanism for the new scale, we have also revisited some of our previous designs in order to test their feasibility with the new scale and new manufacturing capabilities. Extending the cone-shaped connector idea [1] using a locking mechanism similar to the one given in [11], we have tested a connector with four pegs and a latch under the faceplate (Figure 7a). Although the locking mechanism with SMA actuation worked fine, the surface shapes required to guarantee faceplate clearance forced us to decrease the support surfaces for the attachment points. Extending support surfaces for the better results prevented the pegs from clearing the faceplate during approach. However, the cross-shaped connector with sliding pegs still clears the cube faceplate in the new design. This twist-and-lock connection mechanism is designed for the connector to enter and rotate to its locked position inside an opening on the cube faceplate (Figure 9). Once the connector is in locked position, a sliding latch rotates into a position to stop the connector from turning. This single latch now replaces the two sliding pegs we have used in our previous 8cm version; the new size does not permit the use of multiple 100micron SMA wires on the faceplate. 14cm of wire provides the necessary pull to rotate the latch, which is pushed to its locked position by a torsion spring (Figure 7b). The connection mechanism also includes a 3-point electrical connections protruding away from the shaft (Figure 6 and Figure 9). These provide the necessary lines for power and serial communications with the addition of the crosspiece itself. The cube faces are populated with four groups of three connection points for four possible link orientations. The innermost points are for power line while the ones on the outside outsides are chosen for data and ownership lines for serial communications. No communication between modules is permitted while the link is completing its attachment motions (or starting detachment). During the rotation of the cross-

shaped connector, contact points on the link slide over the contact points on the faceplate, crossing serial lines. There is another connector at the bottom of the opening where the connector enters for ground connection between modules. In addition to the connection points, the faceplates also include the SMA wire and the spring to actuate the latch (Figure 7b). Each faceplate is designed to be assembled with fix others to form a cube. The depth of a faceplate is approximately 0.82cm. This leaves a cubic area of size $(4.34)^3$ cm^3, which can be used for the control module including a PIC 16C63A/73B processor, the drivers for the SMA wires and the 7.2V 300mAh rechargeable battery (Figure 8). All link motion control circuitry is realized using surface mount components in order to fit it into the available volume on the top of the link. When the middle link joint is in fully open position, the circuit attached to one side of the link body clears the other arm and motor(s) protruding from it (Figure 6a and Figure 8).

All processors are connected to a single serial bus (USART) coupled to the PC's RS232 port (Figure 10). The communication protocol and messaging format is described in the next section with the graphical interface. The controllers interpret serial messages as low-level action commands. For the cubes, there are actuation commands for the SMA wires. For the links, there are several commands for rotating the joints with or without the feedback. Desired position and speed of the joints can be defined for closed-loop control. Gains for the speed control can also be changed via the interface. The processors have 4Kb of program memory; there are 192 bytes available for program variables.



(a) (b)

Figure 7. Attachment mechanisms: (a) 4-pegs with latch at the center of the faceplate, (b) twist-and-lock mechanism with sliding latch.

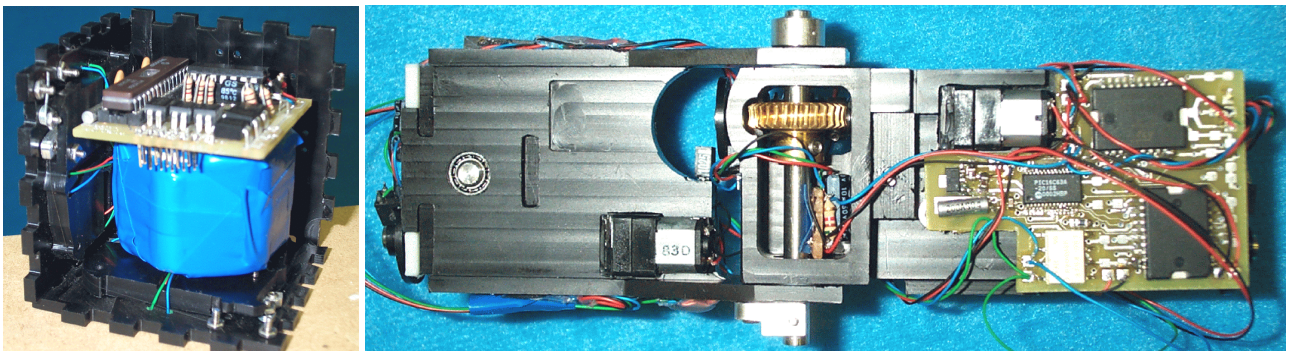


Figure 8. 6cm versions of the cube and the link.

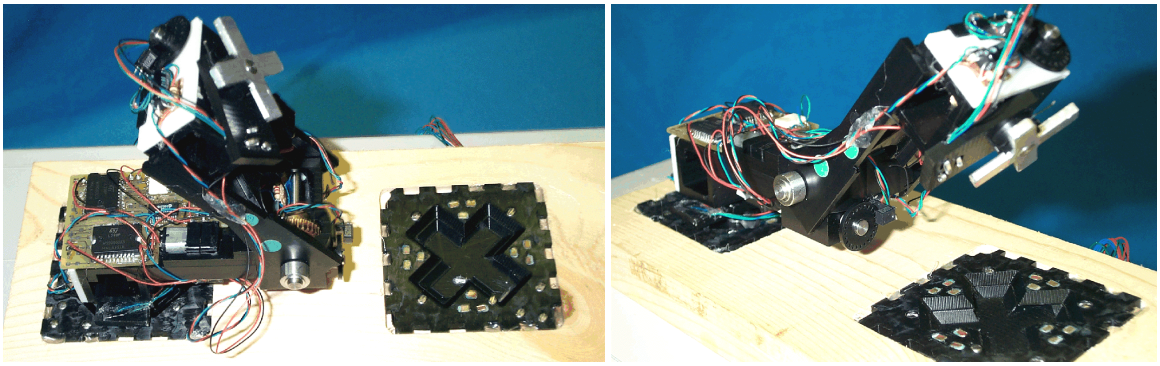


Figure 9. Link approaching the faceplate.

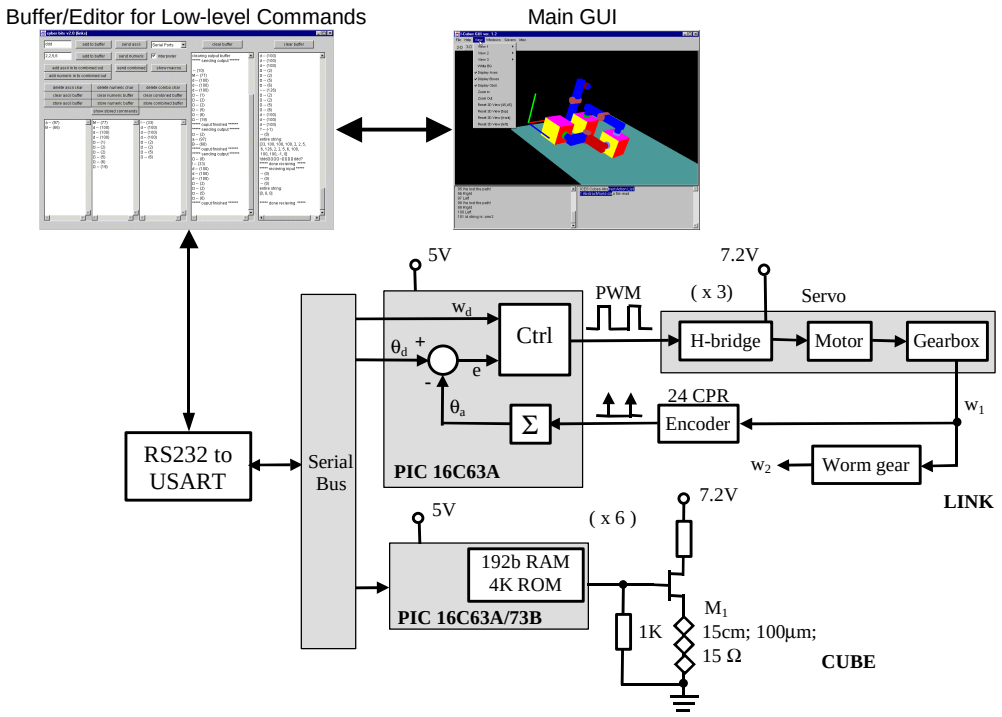
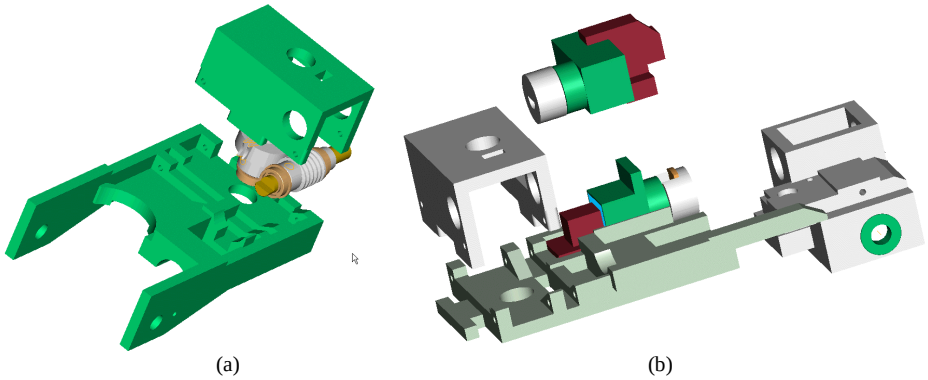


Figure 10. Microcontrollers for links and cubes connected to a graphical user interface via serial line.

Link body is manufactured using Delrin; the five body parts are assembled using pins and screws. The body is populated using off the shelf mechanical components, servomotors, and gearboxes removed from hobby servos and electrical circuits (Figure 11). The servos are held in place with custom designed FDM plastic holders. The faceplates for the cubes are also manufactured on a CNC machine using the same material. The use of a stronger material for the link body and employing better manufacturing methods such as CNC provided a better alignment in the component assembly, which improved the gear efficiency. Backlash and gear problems as well as mechanical noise are reduced significantly. In the current version, a link weights approximately 195gr while a faceplate is around 20gr. The battery and the controller in a cube are measured to be approximately 85gr. These result in a cube weight of 205 gr. The total weight of a link is scaled from 370gr [15] to 195gr, i.e., by approximately 47%. Approximate dimensions of the link side with two servos changed from 10.9 x 3.4 x 2.3cm to 8.5 x 3.7 x 1.8cm. The 33% change of volume in the link body does not carry over the change in mass as theoretical calculations suggest [18]. The discrepancy is of course due to the loose definition of the dimensions and the heterogeneous nature of the links. However, we were able to reduce the ratio of the worm gear from 1:40 to 1:30 for which worms with smaller diameter are available. This provides faster motion for the links, as well as a chance to scale the width of the links due to smaller worm. By scaling down the link and cube size, we were able to decrease the maximum torque requirement on the servos, as well as the stress on the gear structure.



(a) (b)

Figure 11. Five pieces are manufactured and assembled with mechanical components such as (a) gear structures and (b) servo motors.

4. GUI AND COMMUNICATION PROTOCOL

The graphical user interface designed for I-Cubes is used for testing planning algorithms, generating solutions to given problems (Figure 12, A) and manually or automatically control the links and the cubes (B). The interface enables the user to

load problems from data files (A), choose solvers for motion planning (D), identify subroutines and parameters to be used (E), and animate and save the generated solutions (A). The interface can also be used to define problems (C), and manually move the elements of the robotic group displayed on the screen (B). Messages from the different layers of software and from the microprocessors are displayed on the text fields in the main display (F). The earlier versions of the GUI lacked two features found to be extremely important for this application. First, the on-screen display was a simplistic 2-D representation of the links and the cubes creating problems in visualization (G). Secondly, we have used a second interface with color-coded buttons to control the link motions. This method was found to be very unintuitive for the user.

We have recently added a 3-D interface to the GUI in order to alleviate some of the problems we have encountered in visualization and control of the links and the cubes. The latest version of the interface incorporates a 3-D display of the system that can be zoomed, panned, and rotated with the mouse for more clear display of the robot position and shape (Figure 12, H). In addition, the user is now capable of detaching, attaching, and moving the links with the mouse and keyboard commands. Currently active link and link joint as well as attachment and stationary condition of the link end effectors are all indicated with color codes (I,J).

The commands are interpreted by the interface to generate a sequence of low-level action commands to be sent to the link and cube processors via the serial line. A single mouse button and key combination used to define a 'detach link' command is translated into sequence of three commands for (i) the cube to open the locking mechanism, (ii) the link to rotate its end joint, and (iii) for the cube to release the locking mechanism to its idle (closed) position. After each of these commands sent via the serial protocol defined for the I-Cube processors, an acknowledge message is received from the processor responsible for taking the action defined. After receiving messages indicating successful completion, the interface updates the graphical representation of the robot on the screen. The interface also provides the user with the capability of inspecting, editing, generating and sending action commands in their low-level format. This enables the user to test different sequences of actions and edit them in ASCII or numerical format on a separate window in order to quickly test different scenarios and message formats (Low-level buffer/editor shown Figure 10). The serial port definitions and communication speed (Figure 12, K), current display and window properties, and parameters for stability analysis (L) can also be defined using pull-down menus. The interface is implemented in Java 2 using Swing and Java 3-D version 1.2.

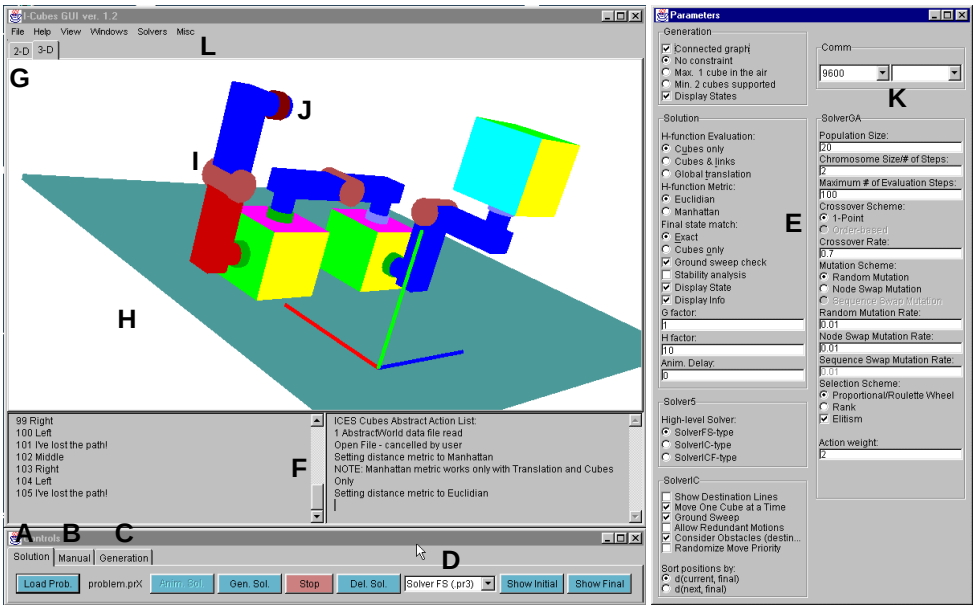


Figure 12. Main window of the graphical user interface.

The serial interface between the microprocessor and the graphical user interface runs at 9600baud. A multi-master asynchronous serial protocol is defined for communication between the cube and link processors and the graphical interface running on the PC. At the given communication speed, the communication system has a bandwidth of 120 messages per second. Given the mechanical characteristics of the system, this speed is fast enough to control link motions and attachment mechanisms on the cubes. All processors including the PC are connected to the same serial bus, which employs two lines: one for ownership, the other for data. The transmit line of each processor is tied to the receive line through a normally open relay. This receive line is tied directly to every other receive line in the system, allowing every module to hear anything on the data line. The transmission is controlled by the ownership line. When this line is pulled high, it means that a transmission is in progress. To claim the line, the processor first checks the state, and if it is low, the line is free to take possession. The processor then pulls the ownership line high, signifying that it has control, and preventing any other processor from

disrupting the data. As an additional check in order to prevent a race condition, the processor that controls the line can listen to the first byte it receives after the line goes high and compare it with the value that it has sent. In case of match, the processor knows that it is the only chip controlling the serial transmission. The messages are of variable length. There are several different types of messages used during communications: queries, replies, commands, and updates. Depending on the message type, the contents of the data bytes differ. For example, a reply message from a link to a cube or to the GUI indicating that the previously defined motion has been carried out successfully includes (i) the header that defines the message as 'motion completed,' (ii) the position and orientation of the link after the motion, and (iii) the identifier for the cube (or GUI) the reply is being sent to. Receiving the initial byte of the message all receiving processor know the length of the data to be received and whether they are required to react and/or respond to the message.

5. MOTION PLANNING

Suggested solutions for motion planning of modular self-reconfigurable systems are usually dependent of the particular system's modular properties. Metamorphing modules [19] use a reconfiguration algorithm based on simulated annealing. The analysis of complexity for the required sequence of motions providing a solution for this 2-D system shows that the order of complexity is related to the number of modules and the maximum "perimeter" defined around the module group. The analysis is applicable to some of the 3-D examples, such as the self-reconfigurable molecules [20]. Fractum [6] uses a distributed algorithm developed under constraints of homogeneity and local information available to each module. This algorithm later adapted to self-reconfigurable structure moving in 3-D and using connectivity between modules to define the desired shape, a stochastic algorithm has been defined [21]. Casal and Yim introduced a divide and conquer approach for closed-chain reconfiguration [16]. The solution applies to robots whose topology can be described by single-dimensional combinatorial topology, and uses a hierarchy of substructures for which basic reconfiguration operations are defined. Slightly different in topology than other systems, Crystalline robot uses the fact that the module are compressible and interchangeable to obtain centralized solutions that are more tractable than the solutions for other systems [22].

The facts that I-Cubes system is bipartite and that modules providing the self-reconfiguration are non-fixed base 3-DOF manipulators complicate the motion planning for large number of modules. We are geared our research toward reconfiguration solutions not only to change the shape and structure of the system, but also to provide solutions to problems involving non-overlapped initial and final conditions. This section discusses the complexity, current solutions, and other motion planning issues related to I-Cubes system.

5.1. Representation of I-Cube States

For a given group of m cubes and n links, the system can take many states, depending on the relative position and shape of the modules. A *state* can be defined as a triple $S = \{C, L, f\}$ where C is the set of cubes representing the cube positions in a cubic grid, L is the set of links representing the position and orientation of the links, and f is a binary relation between C and L defined by a connection matrix with elements indicating a cube-link connection. We also define a *node* as a couple $N = \{S, g\}$ where S is a state and g is its global position. The global position g of the state is defined as the location of the grid cell closest to the origin. A given problem with initial and final configurations can be represented by a pair $\{N_i, N_f\}$, and may have multiple solutions.

We have defined 24 actions for the links, given as 90-degree turns of one of the joints. These actions differ by (i) the joint to be moved, (ii) the direction of rotation, and (iii) the end to be kept fixed during rotation. For example, the action in Figure 2c is defined as "*detach others at A, fix B, rotate B+*" where A and B define the end joints. Actions are actually a group of low-level actuation commands to attachment mechanism on the cubes and the servos driving the joints on the links. Each action can be associated with a translation which indicates the relative position of the new state formed after action completion with respect to the previous state. For a given initial position and shape of a *mCnL* group, it is possible to generate a *state transition diagram* that is a representation of all states as a graph, with actions and associated translations between the states. The size of this transition graph will obviously grow as the number of links and cube increase as discussed in the next section. For a more detailed description of abstract states and related definitions, see [23].

5.2. Complexity

As seen from the description given above, the number of possible states will grow rapidly with the number of cubes and links. While solving a path-planning problem with initial and final nodes, it is not possible to obtain all possible nodes for large number of modules. For example, the total number of states for a 1L2C group is 36 while the number is 4518 for a 2L2C group. Note that the search will be done over the graph of nodes which is significantly larger that the graph of states with the addition of position information. Finding a sequence of link actions for a given cube motion (or more generally for a given problem) is very difficult due to the number of possibilities. The total number of link actions will depend on the number of cubes, the number of links, and the relative position and shapes of the initial and final configurations.

If we only consider the motions of the cubes, it is possible to define an upper bound on the number of cube moves required to reach the final state. Following the discussion of [24], we know that the total number of cube moves required will be at most $O(n.P_{max}(n))$ where $P_{max}(n)$ is the size of the largest perimeter of any connected configuration of n cubes. The perimeter is this case is defined as the neighboring cells that are surrounding the cells occupied by the cubes. These neighbors are *2d* away from the cubes using Manhattan metric, and basically define the set of cells that a cube need to occupy in order to move to another position without breaking the connection between the links and the cubes. Similar to the 2-D case, we have found the size of the maximum possible perimeter for n modules (*i.e.*, cubes) to be a polynomial function of n: $P_{max}(n) = 10.n + 8$. Therefore, the total number of cube moves required for a given problem is at most $O(n^2)$ where n is the number of cubes. The total number of possibilities for a sequence of cube moves providing a solution for a given problem is $(n.k)^s$ where k is the total number of possible cube motions to be considered, and s is the total number of cube motions required to reach the final configuration.

Given a cube move, the total number of link motions required to move this cube to its desired position is at most $O(n.m)$ where m is the number of links. For each cube motion, there are m links that can be used. Although the number of links that are activated for a given cube move is limited to few links near the cube, the upper limit on the number of links that needs to be activated is bounded by the actual number of links. For each link involved in a single cube move, there may be a maximum of n-1 'jumps' from one cube to another. This value is the upper limit for non-repeating link actions; the simulations show around two jumps per link on the average. For larger groups, if involved links are close to the cube that needs to be moved, the number of jumps will be significantly less. A jump from one cube to another is defined as the sequence of actions where a link attached to a cube ends up with its other end attached to another cube (the end could be initially free or attached to a third cube). The maximum possible number of link actions for a jump is 4. While attached to a single cube, a link can reach any position with a maximum face transition of 5, although the number is usually 2 or 3 in the simulations we have seen. Considering the fact that one face of the cube should be attached to another link keeping it connected to the group, a maximum of 4 face transitions seems more feasible. Each face transition can be completed in 5 link motions. Once all the links are in position, the cube can be moved by a limited number of links (t < m) with a limited number of actions (r).

Combining all these bounds, the total number of link motions for a single cube move is at most $O(m.(n-1).4.5+t.r) = O(n.m)$. Note that the discussion of link motions here does not consider interleaved link actions. However, we are assuming that the number of actions required to move a cube would be much less with interleaved link motions. We also consider the possibility of exchanging link roles, such as links handing a cube to another link in order to keep the group connected, and moving it toward the cube to be moved.

5.3. Solvers

Heuristic search methods such as A* are found to provide fast solutions for problems involving small number of modules. If the cost of moving from one node to another (g) is associated with the number of link motions, and the cost of reaching the goal (h) with the sum or average distance (Euclidian or Manhattan) of the modules to their final position, the search returns fast solutions. However, the method cannot be applied to problems involving large number of modules or to the cases where a large number of link motions are needed to reach the goal. We have designed several multi-level "solvers" that approach the problem with divide-and-conquer method. Since obtaining all possible states is time consuming if not impossible, we have taken two approaches when the states are considered. One type of solver uses saved transition diagrams of states for small groups such as 1L1C, 1L2C, 2L2C relying on higher-level planners to define sub-problems that can be evaluated quickly [25]. Another type of solver starts with a small transition diagram and extends it whenever necessary. To illustrate the advantages and weaknesses of these solvers, we will present some scenarios.

Figure 13 shows a scenario where a group of four links and four cubes (4C4L) forms a tower. Not considering the stability issues, several solver are capable of finding solution to this problem. SolverG which starts with the state representation of the initial and final conditions and extends the transition diagram while solving for link motions using heuristic search (namely A*), finds a sequence of link motions for this problem. However, SolverG evaluates a large number of nodes that are not necessary for the solution because these states are closer to the goal node when sum of cube distance to the goal with Manhattan metric is used. Furthermore, for this solver cannot provide a solution for groups larger than 4C4L in feasible amount of time. A combination of SolverIC/SolverG performs much better. SolverIC is an algorithm that considers only the cube motions required to reach the final positions of the cubes.

At each step, the cubes are first assigned a destination from the set of cube positions in the final state. Cubes try to reach a goal position that is closer to them; once a cube reaches a position that is indicated as destination, it stops moving. If a cube's destination is already filled, this cube finds the next best position in the final configuration and updates its destination. Once a destination is chosen, each cube calculates the best possible motion in order to reach the destination. Possible moves are evaluated and filtered using a set of constraints such as availability of pivot cubes or keeping the group of cubes connected at

all times. Once all cubes define their next moves, these actions are filtered using the definition of the pivot cubes. If a cube is defined as pivot by another with higher priority, its move is prohibited for the following step. The moves are also sorted using the distance of cube position to the destination(s). The distances can be sorted in increasing or decreasing order; current or next positions can be compared. These are user-defined variables. If distance comparison of current (next) positions returns a tie, the next (current) positions are compared. In the case of another tie, cube moves are sorted based on the number of cells they need to travel over the perimeter of the group. The number of cubes allowed to move at each step, and the metric used in distance calculations can also be defined. When Manhattan metric instead of Euclidian is used, the path is calculated with the cubes and obstacles taken into consideration.

SolverIC finds a cube motion sequence of length 8. The cube moves are first sorted using the distance between current positions and the destination points in increasing order. The distances are calculated using Manhattan metric and considering other cubes as obstacles. For each cube motion, a sub-problem is created, and SolverG is used to find a local solution. The resulting numbers of link actions for each sub-problem are 3, 3, 5, 3, 3, 14, 3, and 13. The number of nodes evaluated for each sub problem are 3, 8, 42, 8, 14, 1088, 11, and 301. A total of 3776 states is evaluated during solution. As seen in the second snapshot of Figure 13, the link on top of the rightmost cube on the ground creates an obstacle for a link that might lift the leftmost cube. The system searches many (1088) possible nodes until it finds a solution around this problem. For SolverG, each link motion has a cost g of 1. The cost h of reaching the final node from a given node is calculated using the sum of Manhattan distances of all cubes to their destinations. This value is multiplied with a weight factor of 3 to emphasize the importance of cube locations over the number of link motions.

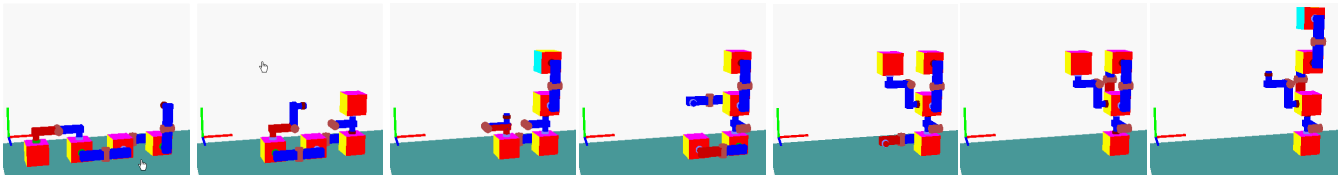


Figure 13. 4C4L group transforming from horizontal to vertical plane.

Figure 14 shows a scenario where 4C4L group moves in horizontal plane with link motions that provide a gait with overturning stability. The solution given in the figure is generated using SolverIC/SolverG combination where the parameters for SolverIC and SolverG are the same as in the example above. The only change in this scenario is that SolverG is forced to remove the nodes that are not stable from the search space. There are 3*4 sub-problems defined by SolverIC. The maximum number of actions in these sub-problems is 8, while the minimum is 3. For the most complex sub-problem (initial state is indicated with a box in Figure 14); 1642 nodes are evaluated. The total number of evaluated states for the whole problem is 2199. A SolverFS/SolverG combination (definition given below) and three-layered algorithm SolverFS/if-then/SolverG defined in [23] also return solution that are the same or very similar to the one given in Figure 14.

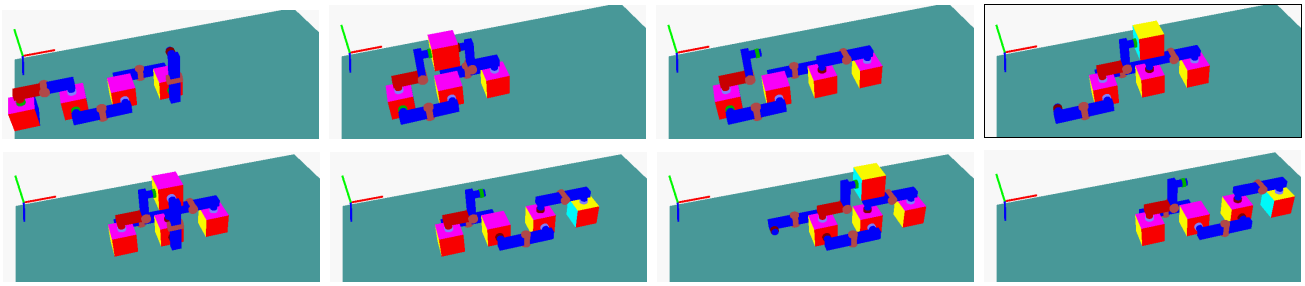


Figure 14. 4C4L group moving in horizontal plane (stable gait).

A 4C4L group that moves over an obstacle is shown in Figure 15. The problem is solved using a combination of SolverFS/SolverG. The algorithm tries to match the final position of the cubes. SolverG uses unit cost for link motions, and sum of Manhattan distances of the cubes is multiplied with a weighting factor of 4 to evaluate cost h of reaching the goal node. SolverG removes unstable nodes from the search space among others. Given a state and its global position (*i.e.*, a node), SolverFS evaluates all cube motions that are acceptable, and sorts them using the resulting positions of the cubes. Hungarian algorithm is used to match all cubes to the desired positions in order to minimize the sum of the distances to goal positions. Starting with the cube motion that provides the best result, the solver looks for acceptable pivot cubes that are needed to move the cube under consideration to calculated next position. The search is completed when a candidate cube is found, and the evaluation, sorting, and checking of cube motions continue with the next node. This algorithm is also used as the first level of the "FS/if-then/G" combination given in [23]. SolverFS returns four desired cube motions indicated with boxed images in Figure 15. The most difficult cube move is the third one. After setting the second cube over the obstacle, the

algorithm needs to find a way to move the third cube to its desired position. This move requires moving the free link attached to the cube on the ground between the cube to be moved and its pivot. The step involves 25 link motions, and requires the evaluation of 6953 nodes.

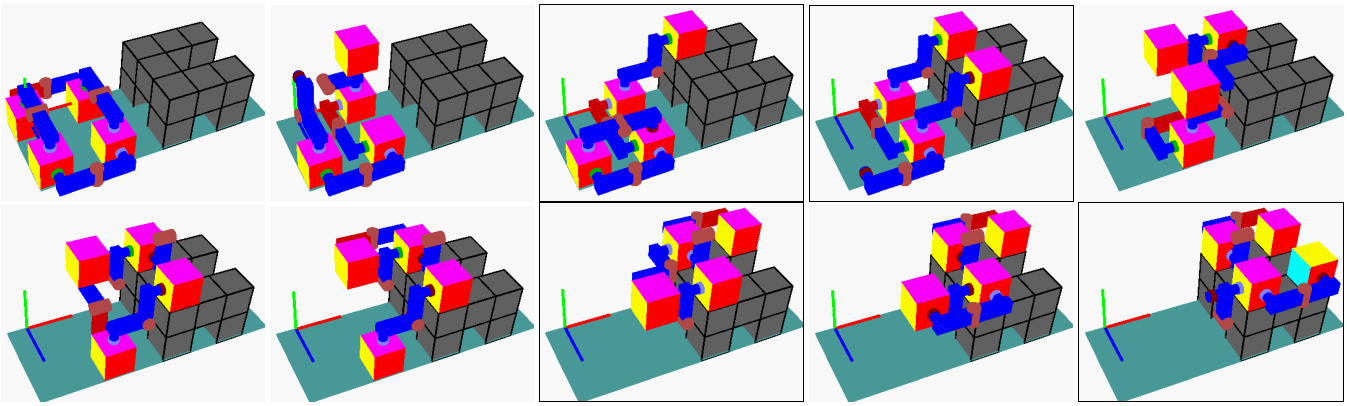


Figure 15. A 4C4L group climbing an obstacle.

There are several situations that high level solvers defined above cannot provide a solution. The main reason behind this is the decision making structure that does not consider several other alternatives to the best solution. This problem is alleviated using a (user-defined) parameter enabling the algorithm to consider cube motions that are not the best, but still acceptable. This capability enables the cubes to move around their neighbors to reach a destination point. The second reason behind SolverIC's mediocre performance is the fact that the cube that reaches another destination point while moving toward their goal do not move, once they realize that their current position is one of the desired final conditions. In addition, there is another major extension that can provide more 'intelligent' solutions to several cases involving larger number of cubes. The new SolverICF uses flags to define the destination for each cube. If there are more than one cube moving toward the same destination, the cube that is closer to this cell is given priority and the other are forced to pick other destinations. Furthermore, cubes that arrive at a destination cell are allowed to move toward their actual destination and switch destinations if their current choice causes a problem for neighboring cubes. If the current assignments are not sufficient to find a solution and the algorithms starts repeating moves, the redundant motions are allowed temporarily in order to move away from this local minimum.

Figure 16 shows a case where a group of eight cubes is required to move over a wall to reach their destinations on the other side. SolverICF only solves for cube moves required to reach the final positions, and does not consider the link motions or stability issues involving the links. Cube moves are sorted based on their current distances to desired final position. The distances are calculated Manhattan metric and taking the obstacles into account. Figure 16 gives snapshots of a solution where only one cube is allowed to move at each step. The solver also returns a similar solution if more than one cube is allowed to move at a given time step. For the first case, the number of steps is 43.

For SolverIC to find a solution to this problem, the destination cells need to be prioritized using a definition based on their respective positions. If the destination cells on the ground are given priority over the others, SolverIC returns a solution similar to the one given in Figure 16. Cubes moving toward the destination will not be caught in a local minimum where all cubes trying to reach closest destination cell act as obstacles for others. Since the available space above the wall is limited to 3 cells, allowing the cube to make redundant moves does not provide a solution for this case. Since the cubes are forced to fill the ground cells first, this provides a better distribution of destination cells among the cubes. A priority function over the destination cells (however simplistic such as a linear function of one of the three axes) enforces a better distribution of the destination cells to even out the disturbance created by the obstacles in the potential function that pulls the cubes toward their goals. The priority definitions are also useful in scenarios such as those given in Figure 17.

Figure 17 show two scenarios involving large number of cubes. A group of 27 cubes moves to form another cube at another destination or a single layer on the ground. Multiple cubes are allowed to move at any given step, and the cubes are sorted in decreasing order of their distance from their destinations. For the problem shown in Figure 17a, SolverIC returns a solution of 30 steps if the comparison current positions is given priority, and a solution of 28 steps if the comparison of next positions is done first. SolverICF, due to its better destination assignment, returns solutions with 25 steps in both cases. For the problem shown in Figure 17b, SolverIC cannot reach the final state, although allowing redundant motions and introducing a priority function forcing the destination cells on the ground to be filled first provide a solution with two cubes unable to reach their destination. SolverICF on the other hand, gets very close to the final state without priority function or enabling

redundant motions; only one cube is incapable of reaching its destination. If we change the distance metric to Manhattan with cubes considered as obstacles, the solver returns a solution of 38 steps. Snapshots of this solution are given in the figure.

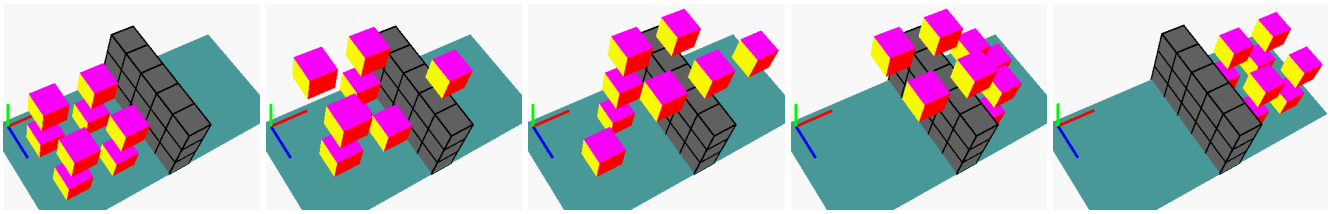
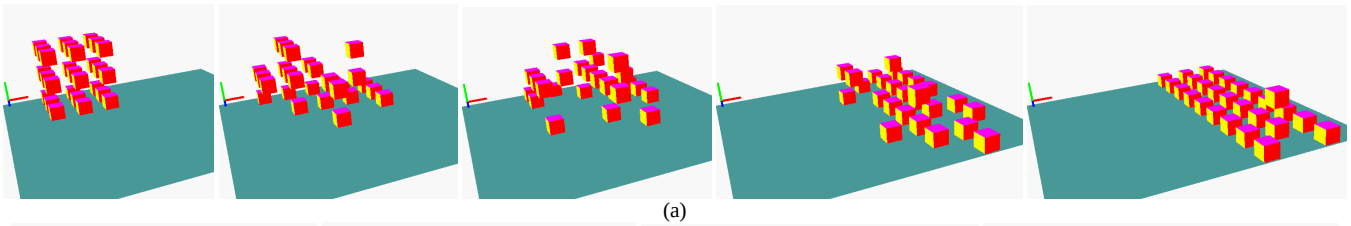
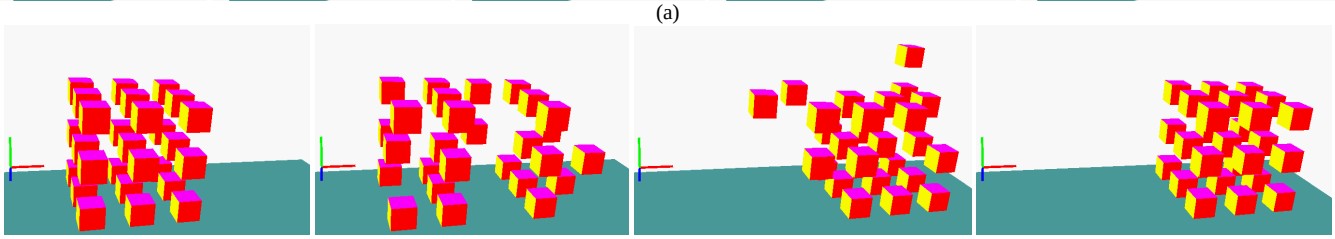


Figure 16. Eight cubes moving over a wall.



(a)



(b)

Figure 17. A group of 27 cubes forming (a) a single layer of cubes and (b) another cube at another location.

As mentioned above, the solvers use several rules to prune some of the nodes from the search space while considering cube and link motions toward the final goal. Although newly generated states are added to the transition diagram, if an action is found to be not permissible for the present node, the resulting node is removed from the list of nodes to be evaluated. Possible reasons for a node to be rejected are: (i) Link or cube does not clear obstacles; (ii) a cube is swept on the ground or slid over obstacles; (iii) total number of supported cubes is less than user-defined value; (iv) a cube moves beyond the user-defined ranges; (v) the overturning moment resulting from link motion causes instability. Note that the possibility of obtaining the cube moves provided by the high-level solvers using the links and stability of the configurations evaluated here are not considered for these large groups. However, we believe that analyzing the stability of different configurations is important during evaluation of motions in a modular self-reconfigurable systems, and run a simple analysis of for the overturning stability of the group while solving for the link motions at the lower-level. Next section describes our approach.

5.4. Stability

Stability analysis can be a part of the decision-making algorithm for the link motions if user enables the analysis from the GUI. Any move that the robot makes, e.g., a 90-degree rotation of a link joint with or without a cube attached to the moving end of the link, should not lead to an unstable configuration. This essentially implies that the group of links and cubes should not topple over under its own weight. This analysis is one step in a process to determine which component of the robot to move to achieve a particular goal. To assess a proposed move of a particular link (and possibly a cube), the move is first checked to determine if there is a conflict with respect to the geometry of the group and the environment. Then, the stability analysis subroutine is used to check for overturning stability. A specific link motion at a given step is checked to assure that it will not result in overturning the robot.

The current analysis is limited to overturning, and does not consider sliding stability. The surface on which the robot rests is assumed to be a hard stable surface capable of supporting the robot without deflection or distortion. The surface is also assumed to be flat and level. The analysis take into account cubes placed on obstacles at different elevations, but does not account for sloped surfaces. However, it is independent of the place of analysis. This implies that this analysis can be performed for any other type of force, similar to gravity, static in nature, irrespective of its direction. The stability analysis is simplified by the geometry of the robot. All of the cubes are positioned such that all the surfaces of the cubes are parallel or perpendicular to the adjacent cubes to which they are linked. The link geometry in terms of arm length are fixed, and the movements can only result in motions that end in cubes being translated a fixed distance (on a plane perpendicular to one of the default axes) and/or rotated by 90 degrees at its center.

The procedure is similar to stability analysis done for cofferdams, and spread footings. The center of mass of the robot is calculated and kept track of during each of the projected moves. The location of the center of mass of the robot, on the plane normal to the direction of gravity, is compared to the footprint or base area of the robot, on the same plane, that will be stationary during projected link motion. If the location of the center of mass of the entire robot including the moving elements falls within the base of the stationary portion of the robot during the course of the link motion, the move is found to be safe, and will not cause the robot to overturn. If during some portion of the move, the location of the center of gravity of the entire robot moves to the edges of the base or outside the base area, the move will cause the robot to turn over.

The calculation of the center of mass is done using the information provided by the initial and final abstract states of the robot for the move under consideration. The robot is viewed as a structure consisting of a moving and resting part. The centers of mass of the two parts are calculated separately. The center of mass of the whole structure is found as a function of the center of mass of the moving portion (Figure 18b). The motion of the link is discretized and the center of mass of the moving part (a link or a link and a cube) is calculated at a fixed number of intermediate positions for the move.

The effective base area of the robot is the area of the stationary portion of the robot that is defined by the boundary formed by the outermost points. These points are simply the corners of the cubes that lie on the ground, or obstacles, and are stationary for the particular move for which this analysis is done. The base area calculation involves finding the 'convex hull' of the base points that form the outer boundary of the cubes that do not move. As a first step, the base cubes are identified. From the base cubes, a collection of the points forming the convex hull is sorted out.

Note that the changes in the convex hull at each time step are due to at most one cube relocation; it involves addition or removal of a group of 4 points from the convex hull. The results of the algorithm are compared to several cases (such as those given in Figure 18 and Figure 15) evaluated in Working Model 3D. The simulations show the system turning over the edges or corners of the cubes for the cases indicated as unstable by our algorithm. The weights of a cube and the two sides of a link are user-defined values. For stability calculations, the location of the center of mass for two sides of the link are given by the user; cube center of mass is assumed to be at the center, and all modules are assumed to be identical.

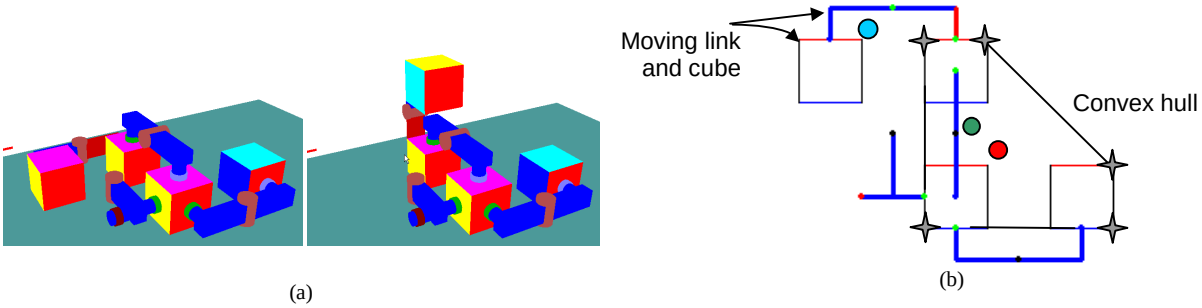


Figure 18. Given a 90-degree link motion (a), center of mass of the moving and stationary parts and effective base area (b) are calculated.

6. SUMMARY AND CONCLUSIONS

We have presented a new version of I-Cubes three-dimensional modular self-reconfiguring robotic system. This new version has power and communication connections between modules and will be powered by batteries on-board the cubes. We have found this 6cm design to be feasible and with better capabilities than our previous design. Our tests on the mechatronic modules and our work toward realizing a group of many links and cubes at this new scale are continuing. A user interface for control of the modules and testing of the planning algorithms is also presented. The interface enables the user to test different configurations, send atomic commands to the modules, and run path-planning algorithms over pre-defined problems.

Finding a sequence of link motion providing self-reconfiguration and/or motion for a given pair of initial and final positions proves to be difficult as indicated by our previous tests. The two approaches taken for solving relatively small groups of cubes and links provide solutions to the problem we have considered so far. A simple analysis of the overturning stability and other geometric and user-defined constraints are used to limit the search space. However, these algorithms are not scalable to large groups. Although, the high-level solvers can find a sequence of cube motions leading to the desired state, using this sequence to evaluate sub-problems and link actions requires a better understanding of what the limitations on the whole system are. Using semantic information on link-cube connections and designing a decision mechanism that can combine link motion sequences for consecutive sub-problems could prove to be useful. We plan to work on algorithms that take link motions and connections into consideration while evaluating cube motions, and other approaches that evaluated the link motion considering the system as a whole group.

ACKNOWLEDGMENTS

The authors would like to thank Andrew Birnbaum, Frederick Hopke, William Hein for their contribution on design and implementation of the system, Brennan Sellner and Kevin Peterson for their computer wizardry, and Dr. Mark Patton for helping us with the stability analysis. The graphic user interface is based on ideas evaluated by Darrin Filer, Brain Williams, Zhan Ye, and Daniel Yocum. This work is supported in part by Defense Advance Research Projects Agency's Microsystems Technology Office and by the Pennsylvania Infrastructure Technology Alliance, a partnership of Carnegie Mellon, Lehigh University, and the Commonwealth of Pennsylvania's Department of Economic and Community Development.

REFERENCES

[1] C. Ünsal, H. Kiliççöte, and P. K. Khosla, "I(CES)-Cubes: A Modular Self-Reconfigurable Bipartite Robotic System," *Proceedings of SPIE, Sensor Fusion and Decentralized Control in Robotic Systems II*, vol. 3839, pp. 258-269, 1999.

[2] C. J.-J. Paredis, and P.K. Khosla, "Kinematic Design of Serial Link Manipulators from Task Specifications," *International Journal of Robotics Research*, Vol. 12, no. 3, pp. 274-287,1993.

[3] B. Neville, and A. Sanderson, "Tetrabot Family Tree: Modular Synthesis of Kinematic Structures for Parallel Robotics," *Proc. IEEE/RSJ Intl. Symp. of Robotics Research*, pp. 382-390, 1996.

[4] T. Fukuda, and Y. Kawauchi, "Cellular Robotic System as One of the Realization of Self-Organizing Intelligent Universal Manipulator," *Procedings of the IEEE International Conference on Robotics and Automation*, pp. 662-667, 1990.

[5] A. Pamecha, C.-J. Chiang, D. Stein, and G.S. Chirikjian, "Design and Implementation of Metamorphic Robots," *Proc. ASME Design Eng. Tech. Conf. & Comp. in Engineering Conf*., Irvine, CA, 1996.

[6] E. Yoshida *et al*, "Experiments of Self-Repairing Modular Machine," *Dist. Autonomous Rob. Syst. 3*, Springer-Verlag, pp. 119-128, 1998.

[7] M. Vona, and D. Rus, "A Physical Implementation of the Self-reconfiguring Cyristalline Robot," Proceedings of the *IEEE Intl. Conference on Robotics and Automation*, pp. 1726-1733, 2000.

[8] K. Kotay, and D. Rus, "Self-reconfigurable Robots for Navigation and Manipulation," *Proc. Intl. Symp. on Experimental Robots*, 1997.

[9] K. Hosokawa *et. al*., "Mechanisms for self-organizing robots which reconfigure in a vertical plane," *Distributed Autonomous Robotic Systems 3*, T. Lueth *et al*, editors, Springer-Verlag, 1998, pp. 111-118.

[10] M. Yim, D. Duff, and K. D. Roufas, "PolyBot: a Modular Reconfigurable Robot," *Proc. IEEE Conf. on Rob.& Auto*., pp. 514-520, 2000.

[11] A. Castano, W.-M. Shen, and P. Will, "CONRO: Towards Deployable Robots with Inter-Robots Metamorphic Capabilities," *Autonomous Robots Journal*, vol. 8, no. 3, pp. 309-324, 2000.

[12] K. Kotay, D. Rus, M. Vona, and C. McGray, "The Self-reconfiguring Molecule: Design and Control Algorithms," *Algorithmic Foundations of Robotics*, A.K. Peters, 1998.

[13] S. Murata *et al*, "A 3-D Self-Reconfigurable Structure," *Proc. IEEE Intl. Conf. on Robotics and Automation*, pp. 432-439, 1998.

[14] H. Bojinov, A. Casal, and T. Hogg, "Emergent Structures in Modular Self-reconfigurable Robots," *Proceedings of the IEEE International Conference of Robotic and Automation*, pp. 1734-1741, 2000.

[15] C. Ünsal, and P. K. Khosla, "Mechatronic Design of a Modular Self-Reconfiguring Robotic System," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1742-1747, 2000.

[16] A. Casal, and M. Yim, "Self-Reconfiguration Planning for a Class of Modular Robots," *Proceedings of SPIE, Sensor Fusion and Decentralized Control in Robotic Systems II*, vol. 3839, pp. 246-255, 1999.

[17] S. Murata, H. Kurokawa, and S. Kokaji, "Self-assembling Machine," *Proc. IEEE Intl. Conf. on Robotic and Auto*., pp. 441-448, 1994.

[18] P. Will, A. Castano, and W.-M. Shen, "Robot Modularity for Self-reconfiguration," *Proceedings of SPIE, Sensor Fusion and Decentralized Control in Robotic Systems II*, vol. 3839, pp. 236-245, 1999.

[19] Pamecha, A., I. Ebert-Uphoff, G. S. Chirikjian, "Useful Metrics for Modular Robot Motion Planning," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 4, pp. 531-545, 1997.

[20] C. McGray, and D. Rus, "Self-Reconfigurable Molecule Robots as 3D Metamorphic Robots," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.837-842, 1998.

[21] E. Yoshida *et al*, "A Distributed Reconfiguration Method for 3-D Homogeneous Structure," *Proceedings of the IEEE/RSJ Conference on Intelligent Robotics and Systems*, pp. 852-865, 1998.

[22] D. Rus, and M. Vona, "Self-Reconfiguration Planning with Compressible Unit Modules," *Proc. IEEE Intl. Conf. on Rob. & Auto*., 1999.

[23] C. Ünsal, H. Kiliççöte, and P. K. Khosla, "A 3-D Modular Self-Reconfigurable Bipartite Robotic System: Implementation and Motion Planning," to appear in *Autonomous Robots Journal*, 2000.

[24] G.S. Chirikjian, and A. Pamecha, "Bounds for Self-reconfiguration of Metamorphic Robots," JHU Tech. Rep., RMS-9-95-1, 1995.

[25] C. Ünsal, H. Kiliççöte, Mark Patton, and P. K. Khosla, "Motion Planning for a Modular Self-Reconfiguring Robotic System," *5th International Symposium on Distributed Autonomous Robotic Systems*, 2000.