Espresso Blocks

Self-configuring Building Blocks

by

Michael Philetus Weller

A thesis submitted in partial fulfillment of the

requirements for the degree of

Master of Architecture

University of Washington

2003

Program Authorized to Offer Degree:

Department of Architecture

University of Washington

Graduate School

This is to certify that I have examined this copy of a master's thesis by

Michael Philetus Weller

and have found that it is complete and satisfactory in all respects, and that any and all revisions required by the final examining committee have been made.

Committee Members:

____

Ellen Yi-Luen Do

____

Mark D. Gross

____

Jim Nicholls

____

Wei-chih Wang

Date: ______________________

Table of Contents

List of Figures

Introduction

In the nineteen-sixties the architectural collective Archigram began to explore the design of dynamic structures, buildings that moved around and reconfigured themselves to suit the needs and whims of their inhabitants.[1] But without a system to actually construct dynamic structures research into their architectural applications has been limited. Robotics researchers have recently developed several modular robotics systems with groups of identical tiny robotic modules that reconfigure themselves to take on a variety of forms.[2] Espresso blocks is an attempt to adapt the design of modular robotics systems to the scale of a brick or concrete block to create a platform for the exploration of the design possibilities of dynamic structures.

---

[1] Ron Harron and Bryan Harvey. "A Walking City," *Archigram*, no. 5, 1964.

[2] M. Yim, D. Duff, K. Roufas. "PolyBot: a Modular Reconfigurable Robot," IEEE Intl. Conf. on Robotics and Automation (ICRA), San Francisco, CA, April 2000. pp514-519.

This document is divided into four chapters. The first chapter explains what a modular building block and a dynamic structure are and what they could be used to create. The second chapter surveys some precedents for the architectural form of dynamic structures and the technology employed to create the blocks. Our initial efforts to create a modular building block prototype are described in the third chapter. And the fourth chapter outlines the next steps to be taken towards the realization of a functional prototype espresso block system.

1 Vision

1.1 A self-configuring building block

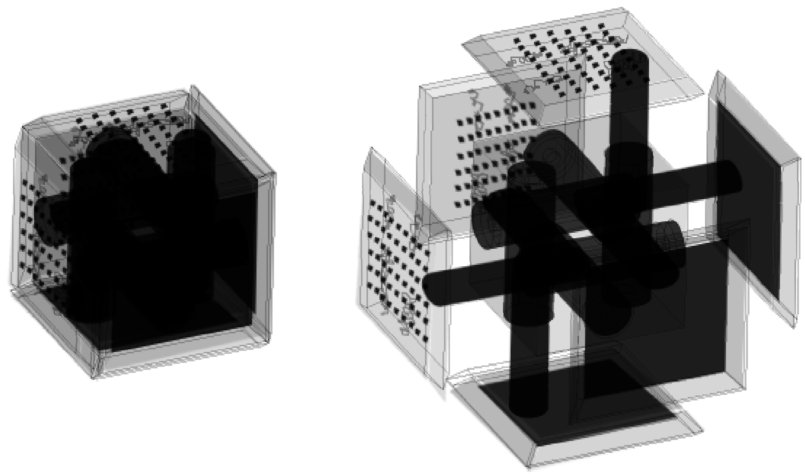

Figure 1 Espresso block expanding

The goal of the espresso blocks project is to produce a prototype self-configuring building block to explore the potential uses of dynamic structures. A self-configuring building block (Figure 1) is a module on the scale of a brick or concrete masonry unit that can be stacked to create a load-bearing structure. Unlike a traditional masonry unit, a self-configuring building block has latches, actuators and a control system. Blocks can latch to neighboring blocks and use their actuators to push each other around. Through the blocks' control systems, these movements are coordinated to produce dynamic structures.

1.2 Dynamic structures

A structure composed of self-configuring building blocks can be reconfigured throughout the day to satisfy the desires of its occupants. There are existing buildings that allow occupants to adjust their spaces to accommodate different activities through the use of moveable partitions or sliding doors. An example that is particularly reconfigurable is Shigeru Ban's 9 Square Grid House[3],(Figure 2) a fusion of the traditional Japanese house with movable screens and the modernist tradition exemplified by Gerrit Rietveld's Schroeder House[4]. These structures are not dynamic in the same sense because a limited number of configurations were designed into the structure when it was built, while a dynamic structure composed of self-

---

[3] "9 Square Grids House," *Japan Architect*, no. 30, Summer 1998. pp30-35.

[4] Lenneke Butler and Frank den Oudsten. "Schroeder House: the work of Gerrit Rietveld between myth and metaphor," *Lotus International*, 1989 no. 60. pp32-57.

configuring blocks can be redesigned to take on an arbitrary novel form as long as it observes the constraints of the blocks. The other advantage of a dynamic structure over a merely reconfigurable space is that it erects itself on the site, and never needs to be demolished. When it is given the command it loads itself onto pallets so that the blocks can be re-used.

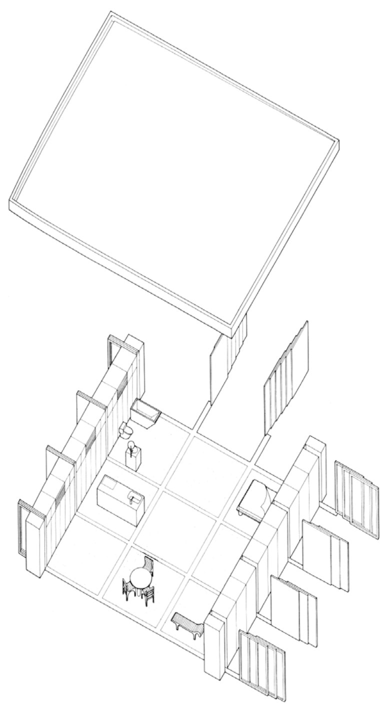

Figure 2 Shigeru Ban's 9 Square Grid House[5]

[5] "9 Square Grids House," *Japan Architect*, no. 30, Summer 1998. pp30-35.

Dynamic structures could use resources more efficiently than traditional structures. They use land more efficiently by accommodating a variety of uses within one space. They use materials more efficiently because blocks can be reused in other buildings rather than thrown into a landfill or downcycled into another building material.

1.3 Block delivery

Dynamic structures are capable of erecting themselves, so to build a dynamic structure it is only necessary to get a sufficient number of blocks and the desired accessories to the site. Blocks and accessories can be purchased at a building supply retailer, or ordered online and delivered to the site on pallets by a truck (Figure 3). For delivery to more remote locations crates of blocks can be airlifted and dropped at global positioning system (GPS) coordinates specified on the online order form. The only required accessories are a remote control, which could be a cell phone or personal digital assistant (pda) such as a palm pilot with the block control software installed on it, and a fuel cell generator or batteries to supply power. Other accessories like an incinerator toilet, a drinking water supply tank and a gray water reclamation tank may be desired.

Figure 3 Pallets of blocks delivered to a site

Once these elements have been gathered at the site it is only necessary to plug the blocks into the power supply and select a configuration from the choices supplied with the block control software. If the new occupants are not satisfied with any of the pre-programmed configurations, they can create a new configuration through the design interface on their remote control or download a configuration from the internet that someone else has designed and posted.

1.4 Block control

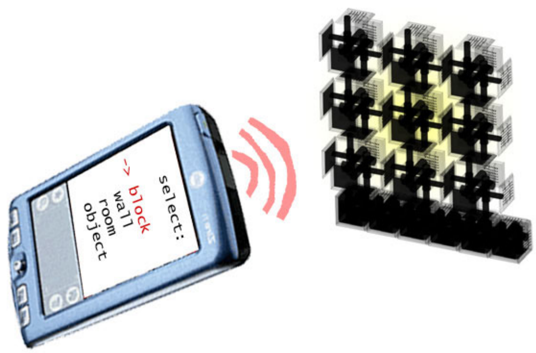

Figure 4 Single block being selected

The configuration of a dynamic structure is controlled by sending messages from a palm pilot or cell phone with an infrared port running the block control software. There are three stages to the reconfiguration process. The first stage is the selection stage (Figure 4). On the remote control, the preferred selection granularity, which can be a single block, a surface, an entire room, or an object, is indicated. When the remote control is pointed at a potential selection, the selected blocks light up. Once a selection is made a list of possible configurations is presented, beginning the second phase of the configuration process. As the occupant cycles through the potential configurations on the remote control the blocks that would have to move to realize that configuration light up green to indicate which part of the structure is going to be affected (Figure 5).

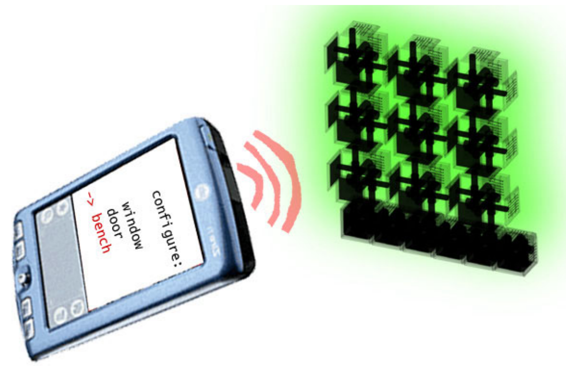

Figure 5 Blocks that would have to move glow green

Confirming a new configuration begins the final phase. All of the blocks that are going to move light up red and begin moving (Figure 6). Once a block has reached its final position, it goes dark, to indicate to the occupants that the reconfiguration has been completed.

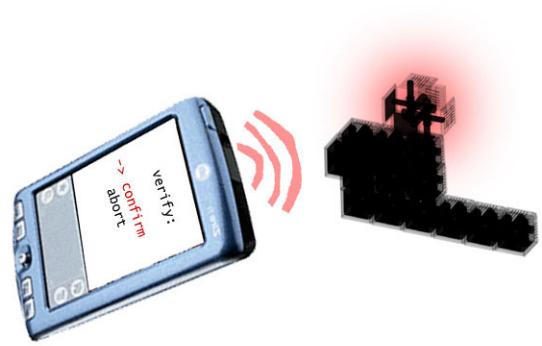

Figure 6 Blocks glow red until they are finished moving

1.5 Block configuration

The block control software comes with a variety of preset configurations. These include both entire rooms, such as an office, a bedroom, a bathroom, and an espresso stand, as well as modifications to a room such as windows and doors, and objects within a room such as a table, chair and bed. To create a new configuration, there will be a design interface in the block control program that allows blocks to be manipulated individually and in groups to create new configurations. More complex configurations can be created in a block simulation program with more sophisticated design tools that can be run on a personal computer and then downloaded onto the remote control. New configurations can then be traded between remotes or posted on web sites for others to download.

1.6 Urban form

Dynamic structures' portability and adaptability allow the creation of a variant of the espresso stand typology, the live/work espresso stand. Like a typical espresso stand, these live/work spaces are erected on semi-public land, either empty space in front of a building set back from the street or space in a pay parking lot, rented from the landlord. In enlightened municipalities dynamic structures can even be erected in on-street parking spaces rented directly from the city.

During the day these spaces function as owner-operated businesses that can quickly change locations to take advantage of shifting economic conditions. At night they house residential communities in otherwise abandoned commercial districts, or provide affordable housing in high-rent residential neighborhoods where the demand for services is high but there are few housing options for service workers.

In areas like parking lots where there is space for several dynamic structures units can share party walls to create marketplaces and housing blocks. With party walls, fewer blocks are necessary to enclose each space, and residents that do not have generators or water tanks can purchase electricity and water from their neighbors. These markets allow new inhabitants to start their own business with less capital, as they need to buy fewer blocks to enclose a space, and can purchase services from their neighbors until they have made enough money to buy their own accessory systems. Once an aspiring microcapitalist has acquired enough blocks and accessories to be self-sufficient, the structure can be moved to a new area of the city with less competition.

1.7 Outposts

Dynamic structures are also ideally suited for providing shelter in remote locations. Crates of blocks dropped from a plane can immediately assemble themselves with no additional tools. In politically unstable areas blocks can quickly assume a defensive configuration, either as a precautionary measure at night or in response to an immediate threat. Parts of a structure that malfunction or are damaged can simply be replaced by spare blocks. And if it becomes desirable to move to a new location, blocks can load themselves onto whatever form of transportation is available, from helicopters to camels.

2 Precedents

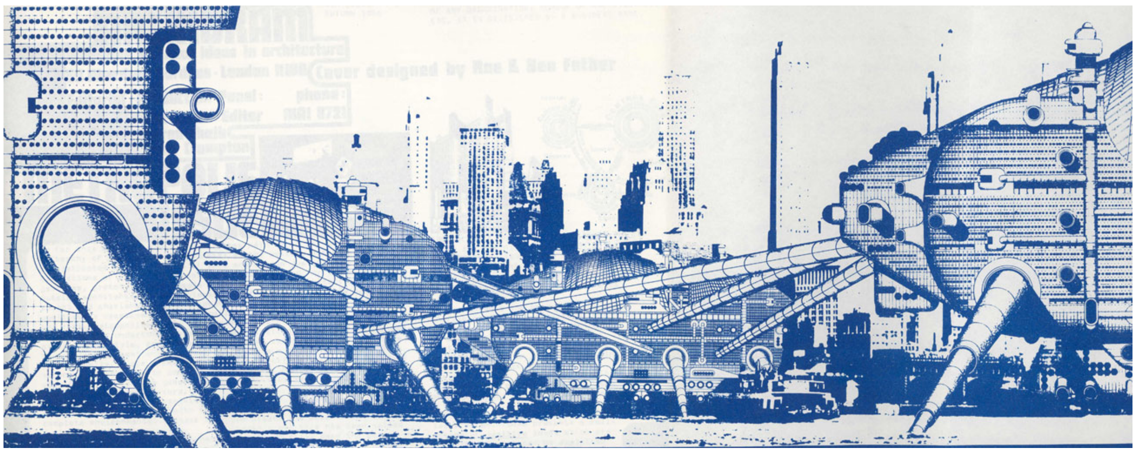

Figure 7 Archigram's "A Walking City"[6]

In their 1964 proposal for a walking city[7] (Figure 7), the architectural collective Archigram describes huge residential multi-use complexes that roam around on stilts. Having buildings walk around on legs was probably intended as a dramatization of the societal effects of capitalism on the urban population rather than a blueprint for a built work. As their jobs required them to frequently change locations, city dwellers led an increasingly nomadic

---

[6] Ron Harron and Bryan Harvey. "A Walking City," *Archigram*, no. 5, 1964.

[7] Ibid.

existence. In the four decades since “A Walking City” was published the pace of urban change has only accelerated, and we propose that dynamic structures not only illustrate this effect but enable an appropriate architectural response.

Espresso Blocks allow inhabitants to pick up and move around the city, or across the country to follow shifting economic conditions, new careers, or personal interests. Block structures can move short distances by inchworming down the street or load themselves onto a truck to travel further.

2.1 Decomposition of the shed

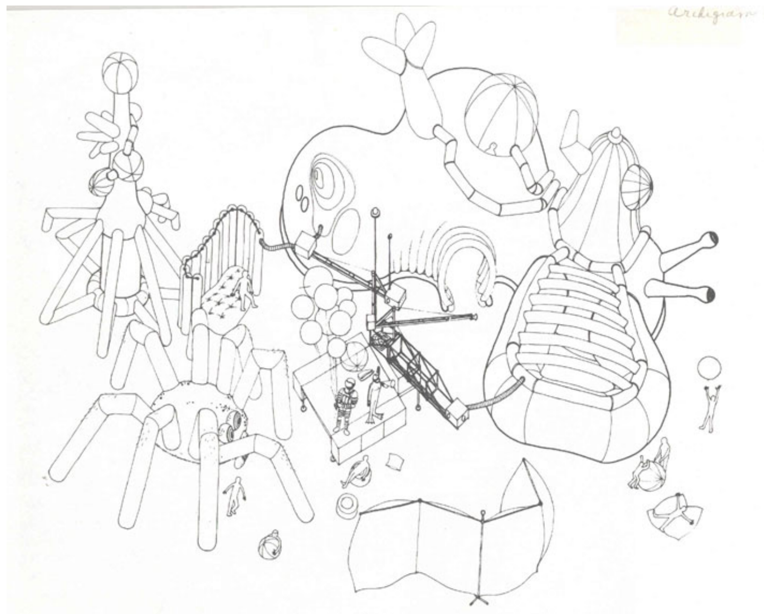

Figure 8 "Flop Out 2" metamorphic kit of parts[8]

In the late 1960s Archigram experimented with the idea that a building can be viewed as a collection of devices that each provide a certain function rather than a shed with some stuff in it. In "Flop Out 2" (Figure 8) they propose that "the single element enclosure becomes irrelevant when thinking in terms of our metamorphic kit of parts."[9] Their

[8] "Summer Collection at Woburn Alley Flop Out 2," Archigram, no. 8, 1968.

[9] Ibid.

L.A.W.U.N. project suggests that it would be much more pleasant to live in a park than a small box, and we could all replace our boxes with a "bottery" of "bots" and go live in the park.[10] Each bot would perform a function that had previously been provided by the house.

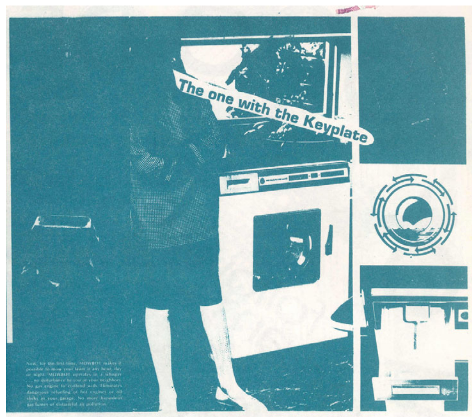

Figure 9 Keymatic bot[11]

The "keymatic" bot (Figure 9) performs domestic duties like cooking and cleaning, the "mowbot" keeps the grass neatly cut, and with "combot" people are "networked into their office in town and don't need to commute anymore."[12] All of

---

[10] David Green. "L. A. W. U. N.", *Archigram*, no. 9, 1970.

[11] Ibid.

[12] Ibid.

the bots are controlled with a small "bot call-up device" (Figure 10).[13]

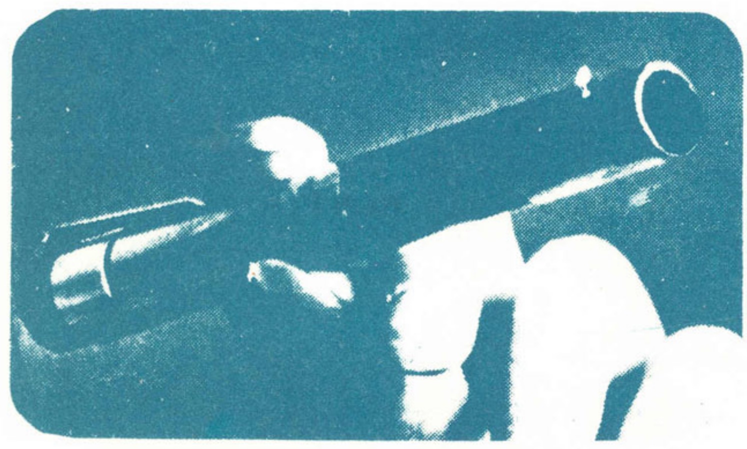

Figure 10 "Bot call-up device"[14]

In a self-configuring modular building block structure, all of the functions of a house are replaced by the block module and a small group of accessory modules. More blocks or accessory modules can be added to any unit to customize individual units. They also provide a more compact kit of parts solution because one group of blocks can take the form of several of the parts in a kit.

[13] Ibid.

[14] Ibid.

2.2 Dynamic structures and land use

Figure 11 Wheel from hamster house[15]

Dynamic structures allow one space to be used for several different functions by shifting new forms and accessories into the space rather than requiring a space for each function. This strategy for making efficient use of built space was suggested by the Viennese architecture firm Alles Wird Gut's Hamster house[16], which employs human-sized

[15] "turnOn," http://www.alleswirdgut.cc/turnon/, June 5, 2002.

[16] "Try Living in the 'Wheel' World," *Wired News*. February 18, 2002. http://www.wired.com/news/gizmos/0,1452,50243,00.html

hamster wheels (Figure 11) to provide several different functions within one space.

In the bathroom module, the toilet, sink and shower are arranged around the wheel. After a shower, you wheel around until the sink comes down and then you can brush your teeth. A dynamic structure's small footprint enables the occupation of previously unoccupiable spaces, allowing the development of underused public and semi-public space.

2.3 Modular robotics

The espresso block module adapts the design of modular robotics systems developed at PARC[17] and other labs to the scale of a building block. While a traditional robot is composed of a variety of parts assembled in a predetermined configuration, a modular robot is composed of a group of identical modules that can rearrange themselves to take on different configurations. One application that has been suggested is the design of robots for the exploration of

[17] "modular robotics at PARC," http://www2.parc.com/spl/projects/modrobots/, March 15, 2003.

distant or dangerous environments.[18] The modules are identical, so if one module fails it can be jettisoned and replaced with a spare. As words can be rearranged to form a novel sentence to describe an unprecedented experience, the blocks of a dynamic structure can be arranged into novel forms when faced with unforeseen circumstances.

2.3.1 Chain modules

Modular robots with the means to latch to one other module at either end, to create a chain of modules, are classified as "chain" modules by PARC researchers. PARC's chain module system, called 'Polybot' has two types of modules. The chain module has a latch at either end and a motor in the middle that allows the module to bend like an elbow. The hub module is a cube with latches on all six sides, but no motor. A hub cannot move, but allows several chains to link together.[19]

---

[18] M. Yim, D. Duff, K. Roufas. "PolyBot: a Modular Reconfigurable Robot," IEEE Intl. Conf. on Robotics and Automation (ICRA), San Francisco, CA, April 2000. pp514-519.

[19] Ibid.

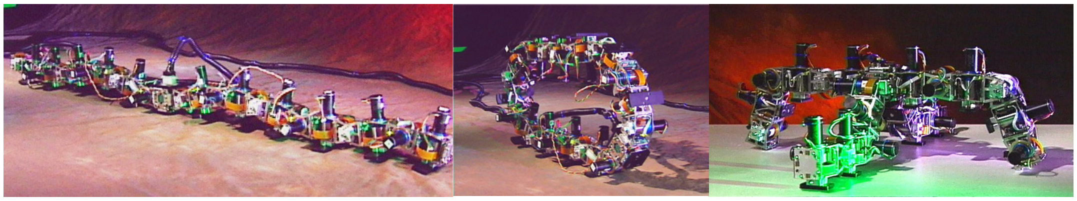

Figure 12 Polybot gaits[20]

Researchers at PARC are investigating Polybot's potential for extraplanetary exploration. A prototype system demonstrated its ability to adapt different "gaits" (Figure 12) to navigate different types of terrain. The modules, arranged in one long chain, inch across a surface like a snake. For faster movement over relatively smooth terrain the chain bites itself on the tail, latching the two ends together to create a loop, and rolls. To scramble over rocks and other obstacles the ends of the chain reach around and latch to a hub module near the middle to create four legs.[21]

---

[20] http://www2.parc.com/spl/projects/modrobots/chain/polybot/demonstrations/g2demos.html, March 15 2003.

[21] Yim, M., D. Duff, K. Roufas. "PolyBot: a Modular Reconfigurable Robot," IEEE Intl. Conf. on Robotics and Automation (ICRA), San Francisco, CA, April 2000. pp514-519.

Each latch on a Polybot module has an array of infrared sensors that the module uses to triangulate its position relative to other free latches and calculate the sort of motion necessary to bring the two latches together.[22] Although chain modules show a great deal of potential for building robot scouts, chains do not easily lend themselves to the enclosure of space, and are saddled with the computing and design overload of tracking their position in space. Modular building blocks do not need the range of motion of an ambulatory robot and would be better served by a simpler system capable of forming planar surfaces.

[22] Ibid.

2.3.2 Crystalline modules

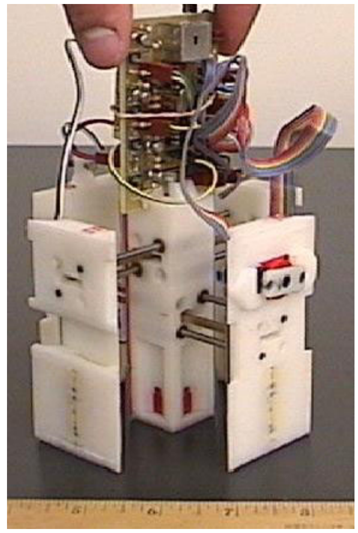

Figure 13 Crystalline atom module[23]

Marsette Vona calls the modular robotic system he and Daniela Rus developed "crystalline atoms",[24] because the modules latch together to form a lattice like the atoms in a crystal. Although the prototype he built (Figure 13) only expands in four directions, he proposed a module in which

---

[23] "The Crystalline Atomic Unit Modular Self-reconfigurable Robot," http://www.ai.mit.edu/~vona/xtal/, July 20, 2003.

[24] Daniela Rus and Marsette Vona. "Crystalline Robots: Self-reconfiguration with Compressible Unit Modules," *Autonomous Robots*, Vol. 10, Issue 1, January 2001. pp107-124.

all six faces have latches and can extend half the width of a module so that two modules can reach across a one-module space and latch together. The modules do not have to adjust their approach to the other module's position because they are constrained by the lattice so that they line up. Crystalline modules also lend themselves to the construction of planar architectural forms like floors and walls.

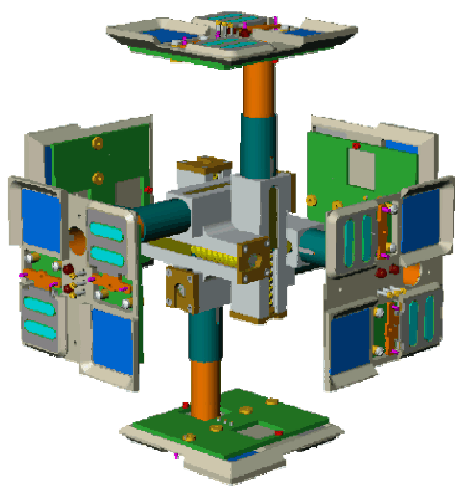

Figure 14 Telecube module[25]

[25] http://www2.parc.com/spl/projects/modrobots/lattice/telecube/, March 15, 2003.

2.3.3 Module latches

The Polybot and Crystalline Atoms both employ mechanical latches to connect to each other. Both systems report problems with latches seizing and failing to disengage while under strain. To avoid this problem, a crystalline module system being developed at PARC, the "Telecube,"[26] (Figure 14) uses a switchable array of permanent magnets to create a connection between modules. When the array is in the on position, its field extends outward and latches to a metal plate on the face of the other module. Even if two faces are not exactly lined up when they attempt to latch together the attraction of the magnetic field pulls them into line with each other. When the array is switched to the off position the magnetic field flip-flops to the opposite side of the array, releasing the metal plate, and there are no interlocking parts to seize up and prevent the separation of the modules.

---

[26] J. Suh, S. Homans and M. Yim. "Design Tradeoffs for Modular Self-Reconfigurable Robots: The Mechanical Design of Telecubes (A Case Study in Progress)," *IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Self-reconfigurable Robots*, Seoul, Korea, May 2001.

2.4 Summary

The trend towards a nomadic culture recognized by Archigram half a century ago suggests an architectural form that can quickly change its form and location to respond to its occupants' shifting needs. The emerging field of modular robotics promises to provide a structural system that meets these specifications. By adapting the modular robotic form best suited to architectural applications, a cubic crystalline module, we can create a building block for the construction of dynamic structures.

3 Design

To demonstrate how modular robotics can be adapted to create an architectural system capable of supporting dynamic structures we are building a prototype modular building block and a computer simulation for modeling block interactions. We present here the basic constraints on our block design, the first two generations of blocks and the first generation block simulation.

3.1 Crystalline module constraints

The prototype espresso block module for the construction of dynamic structures is designed as a crystalline module similar to the Telecube and Crystalline Atoms modules. It thereby avoids the complexity of modules having to align with one another and takes a simple architectural form, the load-bearing masonry wall, avoiding the complication of a skeleton and skin architectural system like a wood frame house or a steel skyscraper.

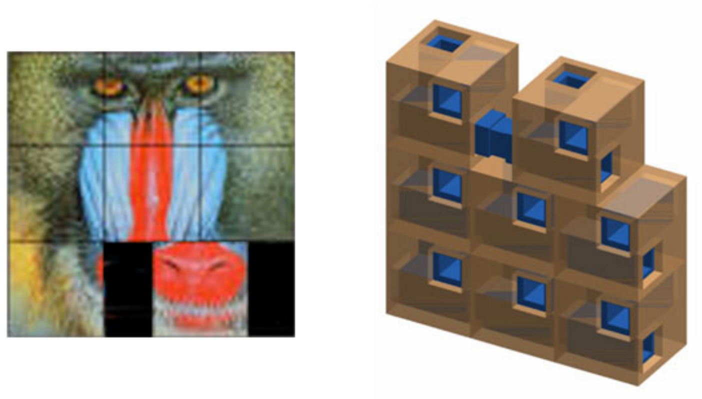

Figure 15 Block movement paradigm

The blocks move like the tiles in a nine tile puzzle game (Figure 15). A block can push away from or pull itself towards a neighboring block to slide across a one block wide space. Two blocks can reach across a one block wide space and latch together. To perform an inchworm motion, (Figure 16) where the head block is pushed forwards by the tail block, latches to another block and then pulls the tail after itself, each block must be able to lift itself and one other block. To move along one of its three axes, a block structure must be at least two blocks deep in that dimension. For this reason, the basic form of a block structure, the load bearing wall, must be at least two blocks thick to be able to extend a block from its surface.

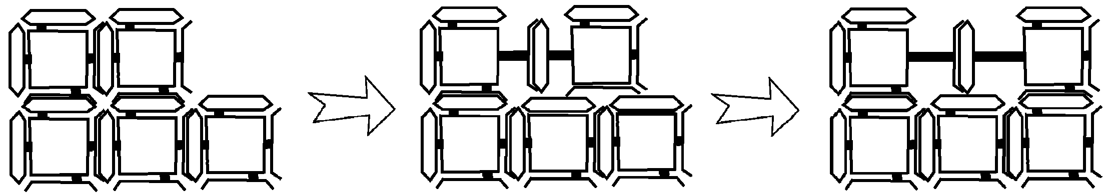

Figure 16 Block inchworm motion

The scale of a block module is born of an attempt to satisfy two opposing constraints. A block should be as large as possible to avoid having to fabricate a large number of blocks to build even a small structure. But a two block thick wall must not be so wide that it consumes as much floor area as the space it encloses. The initial goal size for a contracted block is six inches (~15cm) across, which makes a load bearing wall in a dynamic structure a foot (~30cm) thick.

3.2 First generation

In the first block design, (Figure 17) a block is composed of three tubes, with two extendable arms nested inside each tube, so that one arm extends from each face of the block. Two blocks connect by extending the small arm of one block into a socket in the large arm of the next block, engaging a mechanical latch to hold the two blocks together. The cavities left by the three tubes are enclosed with a clear plastic shell to fill out the cube, and then the entire block is wrapped in a firm plastic gel, providing a comfortable surface to sit and stand on.

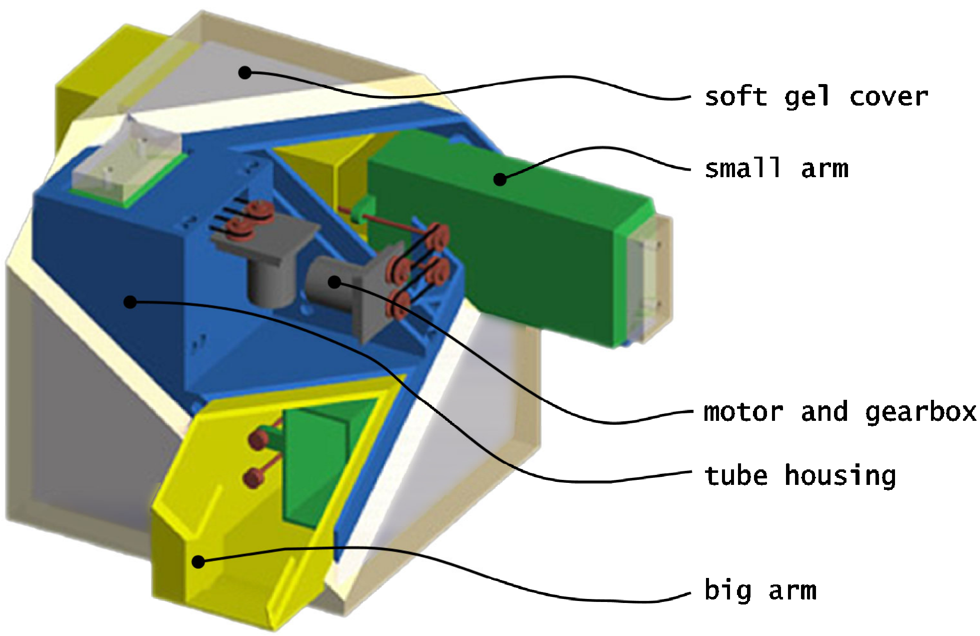

Figure 17 Cut-away axonometric of first generation block

3.2.1 Latch

Two blocks achieve a rigid connection when they latch through the overlap of the big arm and the small arm (Figure 18). The theoretical maximum overlap, if each arm is the full length of a block and the two arms must reach across a one block wide space to latch, and there is an equal overlap with the containing tube and the arm being latched to, is one third of the width of a block. For the initial design the goal was an overlap of one quarter of the width of a block. To allow the big and small arms to latch even if they are not precisely aligned, and to avoid having to extend the small arm of one block to make room for the small arm of the immediately adjacent block to latch to it, both the end of the small arm and the socket to accept it into the big arm are angled.

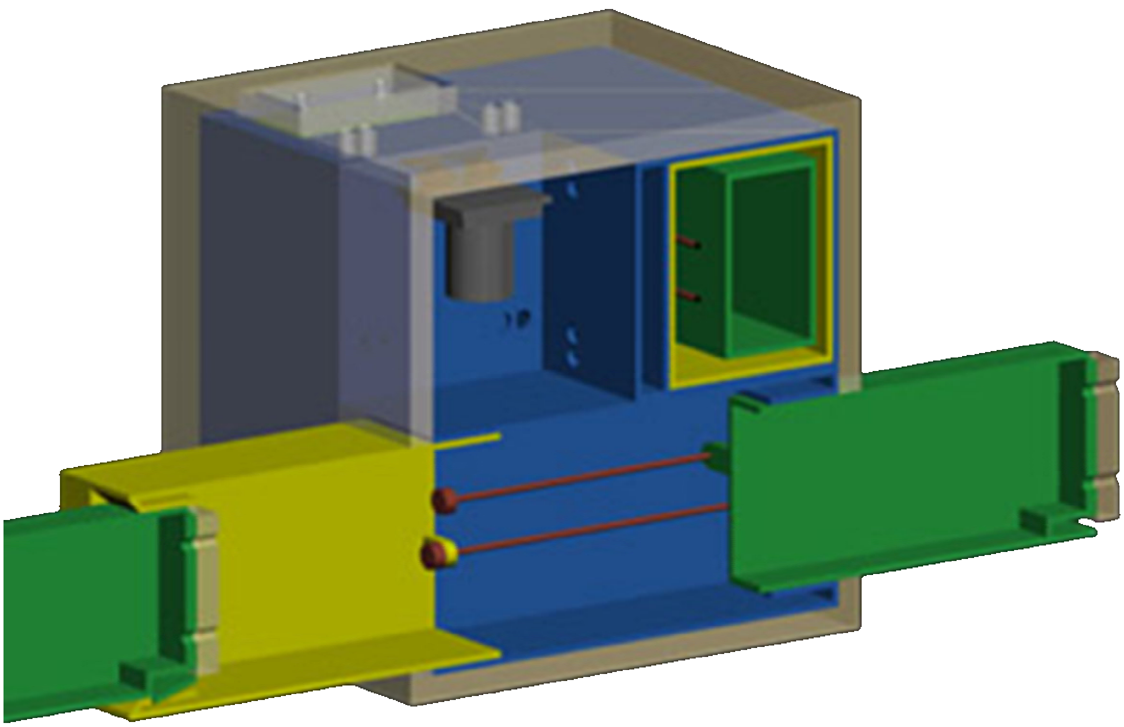

Figure 18 Green small arm latching with yellow big arm

The mechanism that holds the two arms together is a spring-loaded clip on the end of the small arm. (Figure 19) The spring engages the latch when it is fully extended into the socket of a big arm. The latch is disengaged by running an electrical charge through a shape memory alloy (SMA) spring to depress the clip, and then retracting the small arm out of the socket.

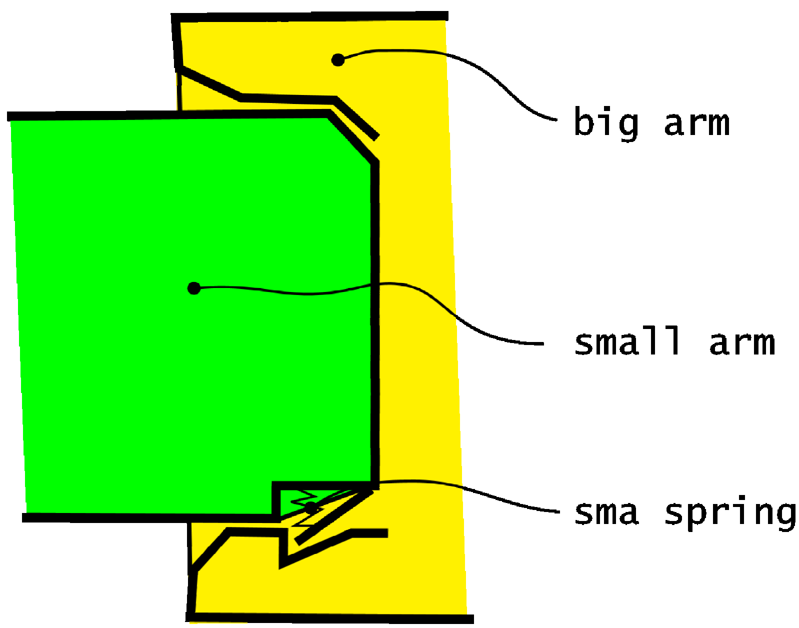

Figure 19 Detail of arms latching

3.2.2 Actuator

The extension of the arms is powered by a thread drive. A compartment at one end of each tube houses the belts that transfer power from the motor to the two threaded rods.

Each rod is threaded through a captive nut on the arm it drives and by running the motor the arms are extended and retracted. When the motor is not running the mechanism is self-braking, avoiding the need for a separate braking system and allowing a dynamic structure to maintain its configuration in the event of power supply failure.

3.2.3 Services

Each tube in a block has a compartment to house services such as drinking water and electricity. When two adjacent blocks are latched together, plugs on one face extend into sockets on the other. The services of the three tubes in each block are tied together through internal sockets and connections. The sockets on each exterior face allow block accessories or other appliances to be plugged in.

For a block to provide a connection through these sockets there must be an unbroken path of latched adjacent blocks to the source of the service, such as the power supply or water tank. Blocks must disengage their service sockets from adjacent blocks to move and would be left without power while moving around. To provide power to blocks while they are in motion there is a secondary system to transmit power through the arms of the blocks. When two blocks latch together, a connection is made between contacts on the outside end of the small arm and the inside of the socket in a big arm.

3.2.4 First generation prototype

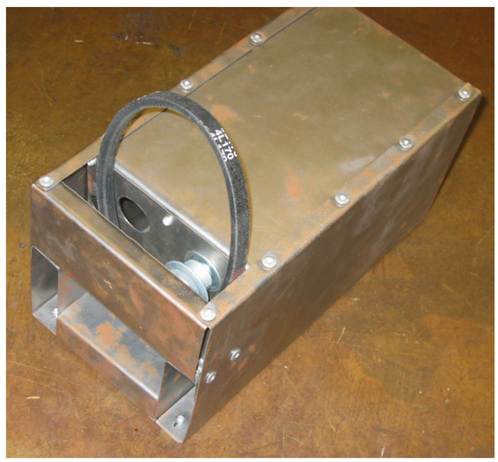

Figure 20 Prototype of first-generation tube

In the early stages of design a prototype tube (Figure 20) was constructed to explore how the thread drive and arms could fit into one tube without interfering with each other. Due to our limited available means of fabrication, the prototype is built out of sheet metal hand bent and riveted together. The goal was to build two tubes with arms and motors to demonstrate the block's latching and actuation mechanisms, but due to the time-intensive nature of the construction method it was abandoned after the completion of the first outer tube.

3.3 Second generation

After our first attempt to construct a tube of the first generation block we realized that we lacked the means to fabricate it. The second generation block (Figure 21) has been designed to take advantage of the one rapid-prototyping device available to us, a computer-driven laser cutter. A second generation block is built out of laser-cut acrylic panels tied together by standard electrical and plumbing pipes and hardware. To address the problem of mechanical latch seizure observed in the Polybot and Atomic Crystals prototypes the second generation design adopts a magnetic latching system similar to the Telecube module's. While the first generation blocks have an arm extending from each face to make a male-female connection with the arm of the next block, the entire face of a second generation block extends outwards and latches to the face of the next block with magnets instead of making a mechanical connection.

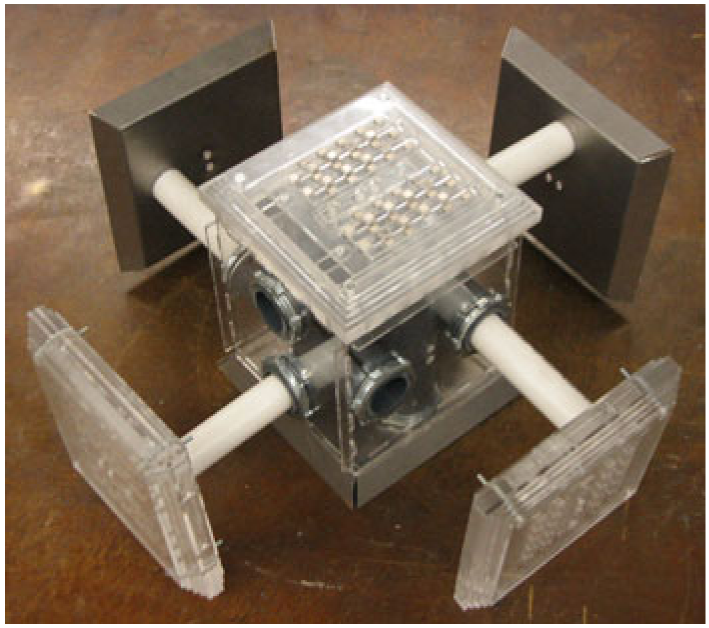

Figure 21 Second generation prototype

3.3.1 Second generation prototype

The six faces of the block (Figure 22) that extend out are mounted on the end of an arm that retracts into a tube in the block core. The core is a 4-1/2 inch (~11.5cm) cube (yellow) formed out of six laser cut acrylic panels that snap together. Each panel has two holes in it for the six five inch (~12.5cm) arm housing pipes that tie the panels together with o-rings threaded onto each end. Each arm housing tube has a 3-1/2 inch (~9cm) long one eighth inch (~3mm) wide slot in each side and a one half inch (~1.5cm) fin on the end of each arm rides in the slots and keeps the faces on track.

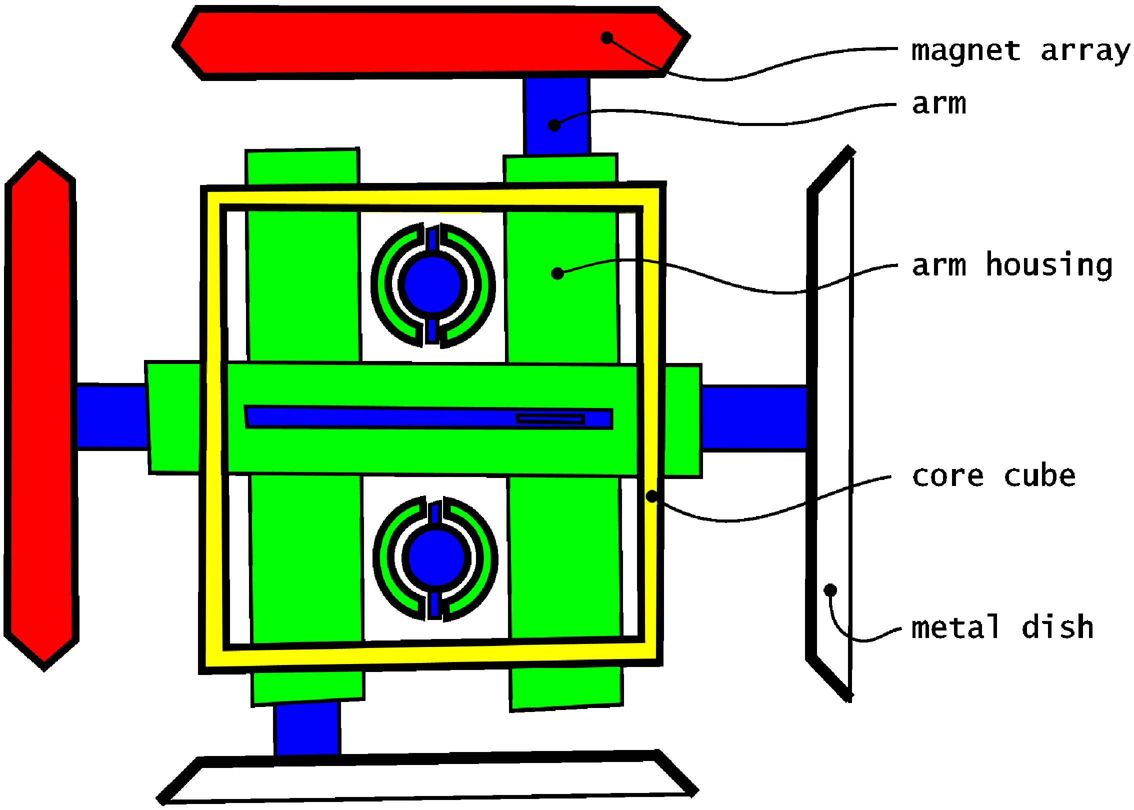

Figure 22 Section diagram of second-generation block

Three of the faces of each block are magnet arrays housed in a stack of laser cut sheets bolted together. Sheet metal dishes bent to receive a magnet array form the opposite faces. All six faces are mounted on five inch (~12.5cm) plastic pipe arms. The sheet metal faces are each one half inch (~1.5cm) deep, the magnet arrays are one inch (~2.5cm) thick and each arm has three inches (~7.5cm) of travel, making each block 6-1/2 inches (~16.5cm) wide contracted and 12-1/2 inches (~31.5cm) wide expanded. The extra half inch (~1.5cm) of each magnet array fits into the one half inch (~1.5cm) deep sheet metal face of the block it is latched to, making the blocks six inches (~15cm) on center when contracted and twelve inches (~30cm) on center when expanded (Figure 23). To maintain this goal spacing while allowing contracted blocks to move past each other the next prototype will have quarter inch (~6mm) deep dish faces, three quarter inch (~19mm) or one half inch (~13mm) deep magnet array faces and 3-1/4 inches (~8cm) of travel for each arm.

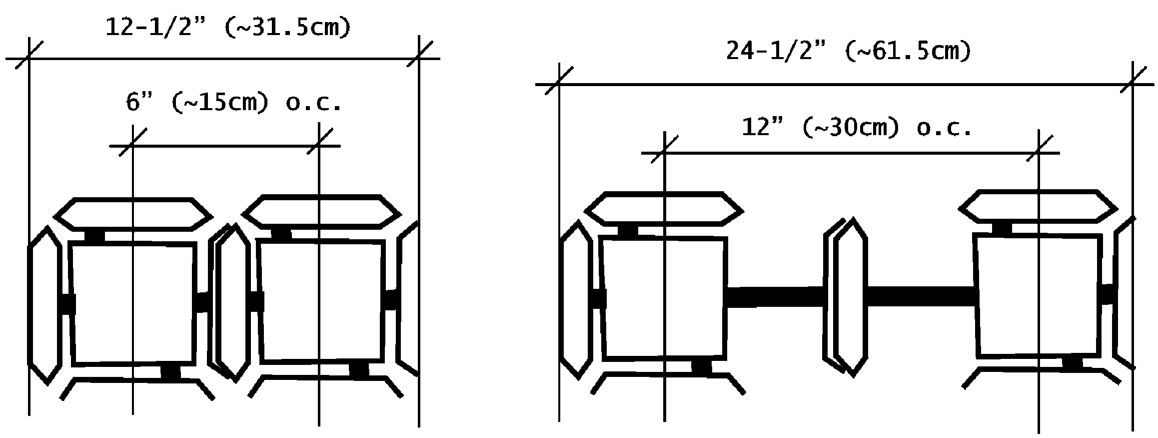

Figure 23 Block dimensions

3.3.2 Magnetic latch

The latch, adapted from the Telecube's[27], uses an array of permanent magnets to connect to a metal plate on the face of the opposite block. Electromagnets are not used because they cannot maintain a connection if the power supply fails. Permanent magnets cannot be turned off like an electromagnet, but by shifting the configuration of the array the magnetic field can be flipped from the outside of the face to the inside so that most of the field no longer reaches the metal plate on the opposite face, allowing it to be retracted.

PARC's Telecube module latch (Figure 24) has two raised areas housing magnet arrays and two recessed areas with metal plates on each face. The raised arrays fit into the recessed area on the opposite face to prevent the magnet

---

[27] J. Suh, S. Homans and M. Yim. "Design Tradeoffs for Modular Self-Reconfigurable Robots: The Mechanical Design of Telecubes (A Case Study in Progress)," *IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Self-reconfigurable Robots*, Seoul, Korea, May 2001.

arrays from shearing off of the metal plates.[28] The espresso block module uses a simplified version of this design with one entire face raised to house a magnet array and the entire opposite face recessed to receive the magnet array, greatly reducing the number of moving parts and only requiring them in every other face.

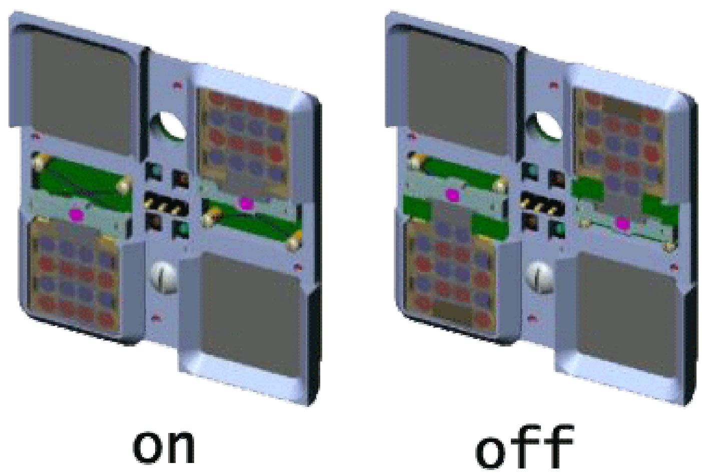

Figure 24 Telecube magnetic latch[29]

---

[28] J. Suh, S. Homans and M. Yim. "Design Tradeoffs for Modular Self-Reconfigurable Robots: The Mechanical Design of Telecubes (A Case Study in Progress)," *IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Self-reconfigurable Robots*, Seoul, Korea, May 2001.

[29] http://www2.parc.com/spl/projects/modrobots/lattice/telecube/, March 15, 2003.

The initial design of the magnetic array was modeled after the Telecube's scheme. Disk magnets are arranged in alternating north-south rows in the on configuration, and to switch to the off position pairs of columns are shifted down one row into a semi-checkerboard pattern.[30] We built a prototype (Figure 25) out of laser-cut acrylic panels with an eight by eight array of 3/8 inch diameter 1/16 inch thick neodymium magnets. Two panels in the center of the face each have two columns of the array embedded in them and slide on a track to shift between configurations.

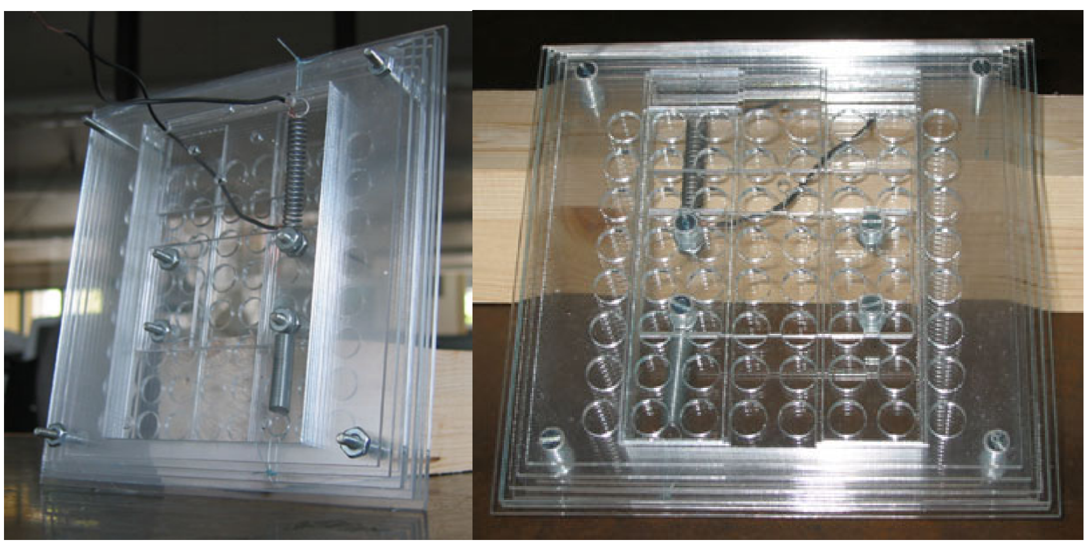

Figure 25 Springs in Telecube-style latch

[30] Ibid.

We were unable to observe any significant change in the strength of the magnetic field when the array was shifted between the nominal on and off configurations. It is unclear whether we misunderstood some essential element of PARC's design, if their design is misrepresented in their paper and on their website, or the Telecube's latch is not actually able to unlatch. We decided to abandon the Telecube's magnet array design and created a new design based on a Halbach array. [31]

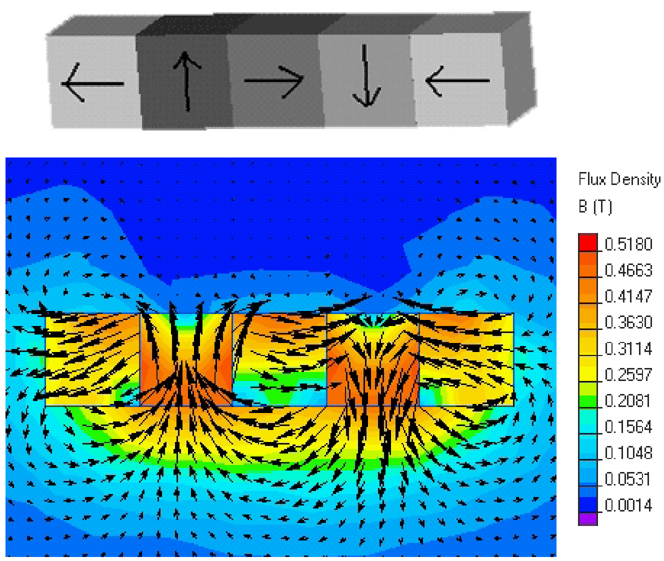

Figure 26 Halbach array flux diagram[32]

---

[31] http://www.otherpower.com/danf/halbach.html, May 22, 2003.

[32] Ibid.

While in the Telecube design the disk magnets are facing either up out of the face or down into it, a Halbach array (Figure 26) is composed of cubic magnets and each up or down magnet is flanked by two magnets facing sideways to redirect the fields of the up and down magnets to extend almost entirely to one side of the face.

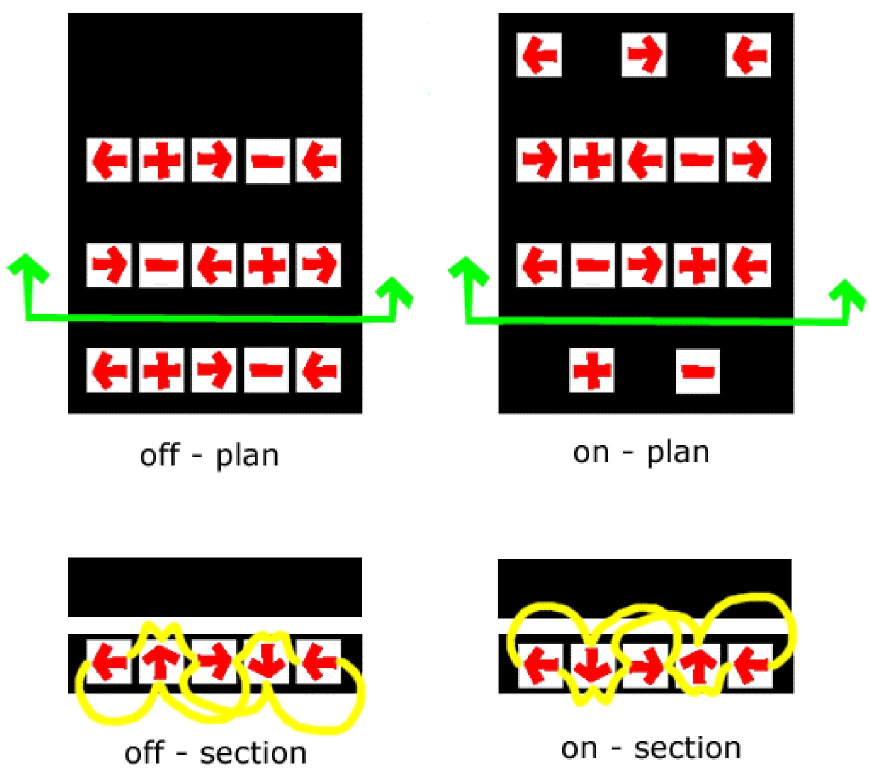

Figure 27 Halbach array latch diagram

In the off configuration of our Halbach array latch design (Figure 27) each magnet with its poles normal to the face of the block is bounded on either side by sideways facing magnets that redirect the magnetic field toward the inside of the block. The rows of Halbach arrays alternate so that when the sideways facing magnets are shifted down one row into the on configuration the fields of the arrays in the center of the face are redirected outwards and the magnets at either end no longer arranged into Halbach arrays extend their fields equally to the inside and outside of the face.

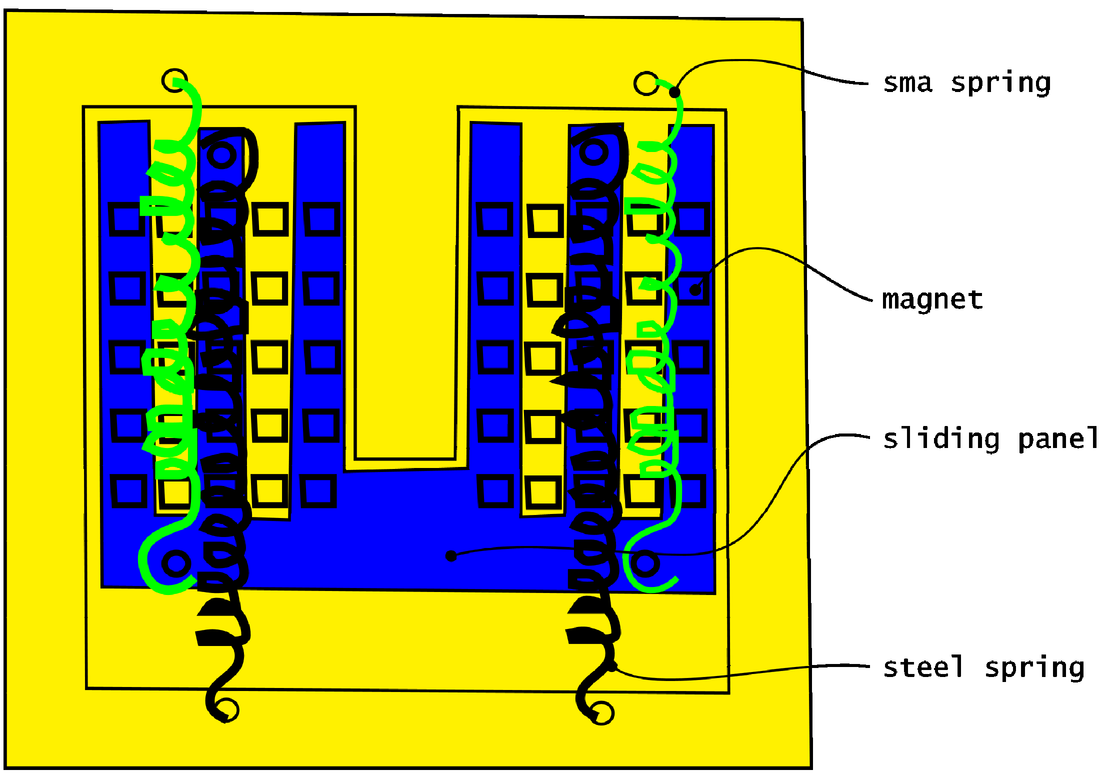

Figure 28 Halbach array latch layout

Shape memory alloy springs slide panels of magnets to shift between on and off configurations in both the latch design adopted from the Telecube (Figure 25) and our Halbach array (Figure 28) latch design. Two steel springs hold the panels in the on configuration with two SMA tension springs on the opposite side pulling the panels toward the off configuration. To unlatch the array electrical current is run through the two SMA springs and they contract, overcoming the steel strings and shifting the panel into the off configuration. When the current is stopped the SMA springs relax as they cool off and the steel springs pull the latch back into the on configuration.

In the initial disk magnet design the magnets are epoxied into holes cut for them in the outside layer of the block face. Embedding the magnets in the outside layer allows them to come directly in contact with the metal plate and exert the strongest possible force, but this has several drawbacks. Because the magnets are extremely powerful each magnet must be individually glued and held in place until the epoxy sets, a time consuming process that is not conducive to producing large numbers of blocks. Two two-layer thick panels in the center of the face house the magnets that shift to switch between configurations. The two layers of each panel are bolted together and the bolts continue through to a cavity inside the face to allow springs to be attached to the panels. When the magnets are epoxied into place some epoxy leaks between the two layers and onto the track the panels slide on, making it sticky. The magnets are not mechanically held in place, and if the epoxy fails the magnets could be pulled out of the face. We also observed a high degree of friction between the panels of magnets and the metal plate, so that it would be difficult to provide enough force with SMA springs to switch to the off configuration while latched to another block.

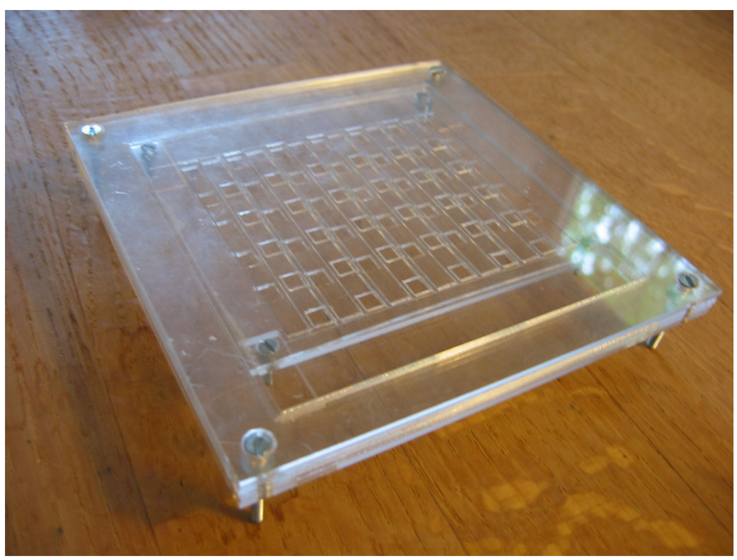

Figure 29 Second latch prototype

In the second latch design (Figure 29) the 3/16 inch cubic neodymium magnets are held in place mechanically without glue between two two-layer thick plates, (Figure 30) one fixed and one that slides. One layer of the outside fixed plate has holes to hold the magnets normal to the face and slots to allow the sideways magnets to slide, and the inside sliding plate has holes to hold the sideways magnets and slots for the magnets normal to the face. The magnets are sandwiched between the two plates so that when the inside plate slides it pulls the sideways magnets with it and leaves the up and down magnets in place shifting the configuration from on to off. (Figure 27) All of the moving parts are contained inside the face, avoiding the problems with friction between the sliding panels and metal plate observed in the disk magnet latch.

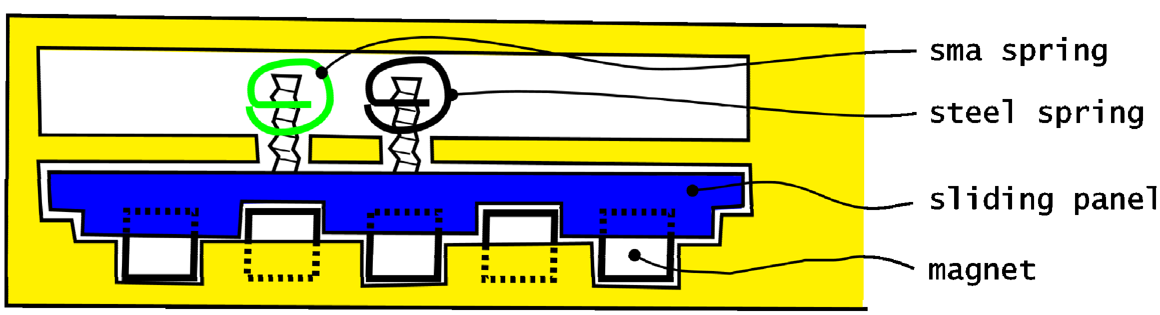

Figure 30 Magnets embedded in sliding panel

In the Halbach arrays the sideways magnets repel from the magnets normal to the face so that the two plates are pulled apart. In our first prototype of the design the face is held together by bolts in the four corners of the face, and the force of the magnets bends the center of the acrylic panels enough to allow the magnets to pop out of the plates. To remedy this problem our second prototype has an island through the center of the block with two bolts through it to prevent the center of the face from being pulled apart by the force of the magnets. The moving plate has a slot in its center to accommodate the island which also has space for an infrared sensor and electrical contact to provide communications and power between blocks.

To hold the magnets in place during assembly, strips of packing tape are placed across the holes before the plates are bolted together. The tape is strong enough to hold the magnets in place while placing the magnets in each plate, but it is very difficult to put the two plates together without pulling some magnets out of position. Even once the plates are together with all the magnets in position, it is still necessary to hold the plates tightly together while assembling the rest of the block as the six bolts that hold the plates together cannot be tightened until the whole face is assembled because they run through the entire face to nuts on its back. In the next prototype the moving plate will be slid in from the side after the containing assembly is bolted together.

Before our first Halbach array prototype self-destructed, we observed a significant drop in the strength of the magnetic field when it was switched to the off position, but have not yet had the opportunity to measure the force necessary to separate the magnet array from the metal plate in either configuration.

3.3.3 Water exclusion and reclamation

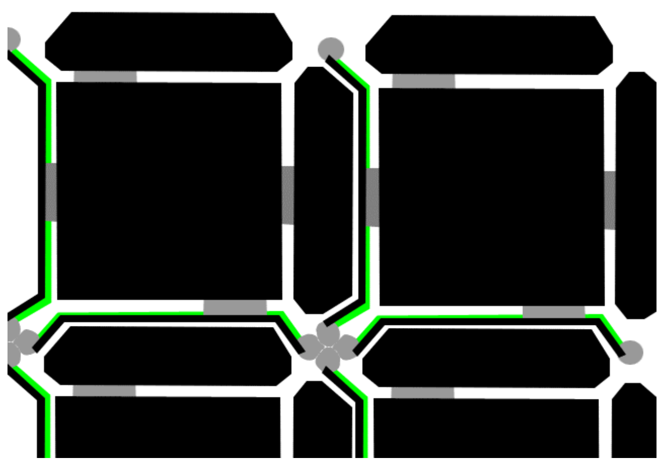

Figure 31 Gaskets and water reclamation membranes

Block structures need to be able to maintain a dry interior when it is raining and capture water used inside for washing or showering without letting it run out onto the street. In a future block prototype the sheet metal faces of each block will have a gasket running around their outside edge (Figure 31) so that when blocks latch together they will form a watertight connection on one side. A membrane on the back of each sheet metal face will absorb any water that falls onto the block from it above and direct it into the structure's graywater reclamation tank for reuse.

3.3.4 Actuator

In our initial design for the second generation block the extension and retraction of the faces is actuated by a solenoid coil. After our initial prototype of the solenoid assembly failed to induce any motion in the arm we calculated the number of coils we would need to lift a block with a reasonable amount of current. Using the Biot-Savart Law, the force in Teslas (T) of the magnetic field B produced by a current i in a circular current loop is

$$(1) \quad B = \frac{\mu_o i R^2}{2(R^2 + x^2)^{3/2}}$$

where x is distance away from the loop you are measuring the field B, R is the radius of the loop and μo is the coefficient of permittivity. If you measure at a distance much greater than the radius the equation becomes

$$(2) \quad B = \frac{\mu_o i R^2}{2x^3} \; .$$

Considering loops with N turns the equation becomes

$$(3) \quad B = \frac{\mu_o N i R^2}{2x^3} \; .$$

To produce a field with the strength of just one of the permanent magnets in our latch, .35T, as in equation 5,

$$(4) \quad .35 = \frac{(1.26\times10^{-6})Ni(0.01)^2}{2(0.15)^3}$$

$$(5) \quad Ni = \frac{.35(2)(0.15)^3}{(1.26\times10^{-6})(0.01)^2}$$

the product of the number of loops and current in Amps would have to be

$$(6) \quad Ni = 8.33\times10^6$$

Our prototype solenoid has 100 coils and we were running 10 Amps of current through them, giving us

$$(7) \quad Ni = 1\times10^3$$

falling orders of magnitude short of the force of even one permanent magnet. We realized that it is not feasible to produce the amount of force we would require with anywhere near the amount of current we intended to make available to a block. We are currently investigating using a stepper motor to drive a rack and pinion assembly for each arm.

3.4 AutoCAD Visual Basic simulation

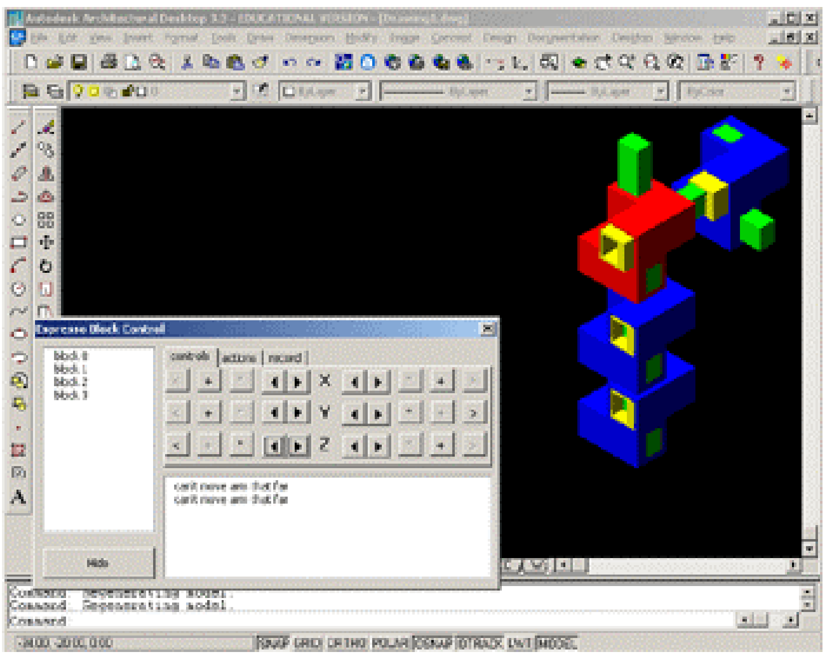

Figure 32 Screenshot of simulation

To demonstrate that block structures can transition between useful configurations despite a block's limited range of motion a block simulation was built in AutoCAD with Visual Basic (Figure 32). The simulation allows a group of blocks to be created and then manipulated by selecting a block, unlatching it from some of its neighbors to give it the freedom to move, and extending and retracting its arms. While the simulation does not impose physical constraints like gravity it makes it possible to show that the blocks' limited range of motion does not preclude the transformation between two configurations.

3.4.1 Control panel

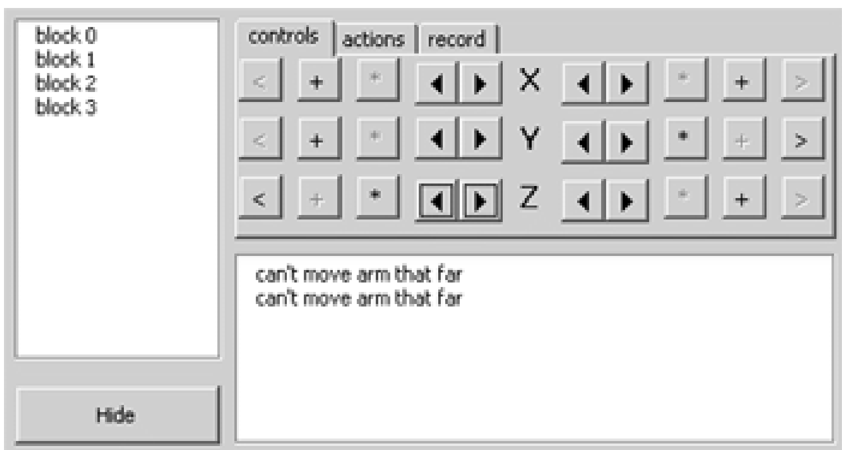

Figure 33 Block control panel

When the simulation is started, a block and a window with controls appear on the screen. (Figure 32) The view and rendering mode can be changed through the AutoCAD interface. The control panel window (Figure 33) is a mock up of the remote control interface that will be run on a palm pilot to send instructions to blocks. It displays a list of blocks, controls for each face of the selected block, and a message window. When a block is selected, it is highlighted on the screen in red, and control functions that are unavailable to that block are grayed out. The controls for each face include arrow buttons to allow the arm to be extended or retracted, a '*' button that is active if the selected block is latched to another block on that face to unlatch from the neighboring block, a '+' button to create a new block latched to this face which is only available if there is no other block in the way, and a '>' button that is active if there is another block adjacent to this face, to select that block.

If when the arm of a block is extended or retracted, that arm is attached to another block, and that block is free to move, it moves along with the arm. If the block attached to the arm is not free to move, but the block that owns it is, then the block whose arm is being extended or retracted moves and the arm stays in place. A block is free to move if it is not attached to any blocks besides the one trying to move it, or is only attached to one other block that is not attached to any other blocks. When an arm extends, if it would have to push more than two blocks to extend, it doesn't, and prints “can’t move arm that far” to the message window. If an attempt is made to extend or retract an arm that would move it past its maximum or minimum bounds, the arm does not move and a message is printed to the error window.

3.4.2 Recording actions

Moving individual blocks around is a slow process. The record tab on the control panel brings up an interface that allows a transition (Figure 34) to be recorded so that it can be played back again. An action is recorded by entering a name for the action, pressing the record button, switching back to the control interface and performing the transition, and switching back to the record tab and pressing stop when the transition is complete. In the current implementation a button must be manually created on the actions tab, but the goal is that a button will automatically appear when an action is created. When the control program is run on a palm pilot, it will then be possible to trade the action with other remote controls or upload it to a web site.

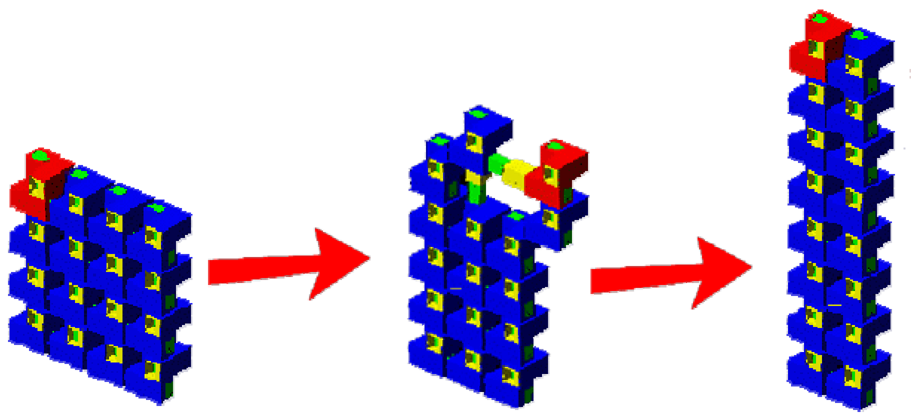

Figure 34 Pallet to wall action

3.4.3 Internal representation

The simulation program has three parts, the control panel, a collection of block projects, and a world object that maintains a hash of the positions of all of the blocks.

When a button on the control panel is pressed to extend the arm of a block it sends a call to the currently selected block to extend its arm. If the arm is not latched to another block, it calls the world object to determine if there is a neighboring block the arm being extended could latch to.

The world object's position hash (Figure 35) contains an index of all of the x positions currently occupied by a block. Each x position contains a list of all the y positions occupied by blocks at that x coordinate. Each y position has a list of z positions occupied by blocks at that x, y coordinate. And each z position has a reference to the block object that occupies that position.

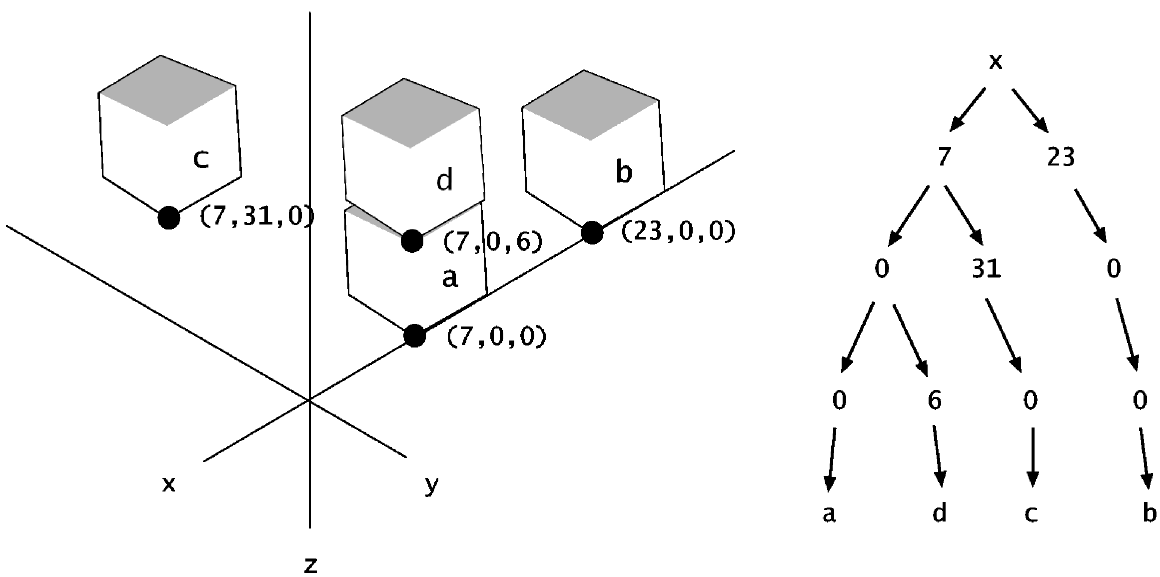

Figure 35 Position hash structure

The world object looks to see if there is another block lined up with the face of the arm being extended within one block length. If there is no block the arm is extended and the world object is called to redraw the screen and reset the control panel buttons. If there is a block within one block length and the combined extension of the two arms bridges the gap between the blocks they are latched together.

If the arm being extended is latched to another block the selected block calls the block the arm is latched to to determine if it is free. If the latched-to block is free it calls the world object to see if the space that it is going to move into is occupied by any other blocks. If the space is free the world object updates the block's position in the position dictionary, the selected block is sent the ok, and both the arm and the block it is latched to move. If there is another block in the way the selected block is sent a message, nothing moves and an error message is printed to the control panel's message window.

4 Future work

4.1 Functional block prototype

The first future goal of the project is to complete a few working prototype blocks so testing on their control program can begin. A suitable actuator needs to be added and tested in conjunction with the latch. Once the latching and actuating mechanisms are functioning each block will need infrared sensors for communications and range finding, electrical contacts to carry power between blocks and a processor to coordinate everything.

4.2 Freestanding simulator

We have already begun to build a freestanding version of the block simulation program with Python and the OpenGL graphics library. The same code that will run on the prototype block's processors will be used to model the block's behavior in the simulation, allowing preliminary testing of the block control code and modeling of complex block structures.

4.3 Blocks solve goal shapes

The first generation block simulation requires the designer to specify every action each block takes. Once the basic block control functions are worked out our goal is to allow designers to specify a goal shape and have either the blocks or the remote control determine a course of action to achieve that shape.

4.4 Blocks avoid unstable configurations

Before blocks are deployed to form inhabitable structures we will to add an additional layer to the block control program that models the statics of block structures and prevents blocks from attempting to adopt unstable configurations.

4.5 Block furniture

An entire block structure will require such a large number of blocks that it will probably be unreasonable to construct one until a facility is set up to mass produce blocks. The first application for the prototype blocks will instead be pieces of furniture. Designing and testing transitions between different forms of furniture will allow experimentation with the interaction between people and dynamic structures without the danger of suspending loads overhead.

Annotated Bibliography

A. Modular Robotics

Agrawal, S.K., S. Kumar, M. Yim, J. Suh. "Polyhedral Single Degree-of-freedom Expanding Structures," *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Seoul, Korea, May 2001.

Explores the idea of building a lattice structure that has only a few moving parts that change the shape of the whole structure. A interesting and applicable idea, but they did not actually build it so there is little guidance offered on technical issues.

"Modular Robotics at PARC," http://www2.parc.com/spl/projects/modrobots/, March 15, 2003.

PARC's modular robotics site describes several interesting prototypes that they have actually built with diagrams and videos.

Roufas, K., Y. Zhang, D. Duff, M. Yim. "Six Degree of Freedom Sensing for Docking Using IR RED Emitters and

Receivers," *Experimental Robotics VII, Lecture Notes in Control and Information Sciences*, Eds. Daniela Rus and Sanjiv Singh Springer, 2001. pp271-9.

Describes a system of sensors to guide two modular robots to dock with one another. Getting this to work is one of the most difficult problems involved in having modules reconfigure themselves, and this system seems to be functional and fairly inexpensive to build.

Rus, Daniela and Marsette Vona. "Crystalline Robots: Self-reconfiguration with Compressible Unit Modules," *Autonomous Robots*, Vol. 10, Issue 1, January 2001. pp107-124.

Suh, J., S. Homans, M. Yim. "Design Tradeoffs for Modular Self-Reconfigurable Robots: The Mechanical Design of Telecubes (A Case Study in Progress)," *IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Self-reconfigurable Robots*, Seoul, Korea, May 2001.

Describes a system of cubes that attach to each other on all six faces, with faces that telescope out. A cube can extend a face and attach to another cube, and

then pull itself over to the cube it attached to. A block system is promising for architectural applications. They have built working prototypes and have a lot of useful technical information.

Yim, M., D. Duff, K. Roufas. "PolyBot: a Modular Reconfigurable Robot," *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, San Francisco, CA, April 2000. pp515-519.

Describes a modular robot with two module types. The chain type is a cube that has two faces that can attach to another cube, and a motor in between that allows it to twist. Several attached together make a snaking chain. The second type is a cubic node that can attach to another cube on all six faces, but doesn't move at all. Several snaking chains can attach to a node to create a robot with legs and walk around. Then it can bend into a circle and roll. Although the chain robot type is not suitable for producing enclosure, this project is on the third generation of robots, and has a lot of technical issues worked out.

B. Reconfigurable and Dynamic Structures

"9 Square Grids House," *Japan Architect*, no. 30, Summer 1998. pp30-35.

Shigeru Ban's 9 Square Grid House draws on traditional Japanese houses with movable screen partitions and the modernist aesthetic of house as machine to create a grid of nine spaces separated by sliding wall partitions.

Bell, Jonathan and Sally Godwin. "Transformable Architecture for the Homeless," *Architectural Design*, v. 70, no. 4, 2000, pp34-39.

For housing for refugees and the displaced to be successful, it must allow the inhabitants to customize the space to suit the chaotic and changing needs of a marginal existence. One of the central goals of this projects is to restore control of the space to the inhabitants.

Brown, Kate and David Bamford. "Manifest TentCity," *Arcade*, summer 1999, p16.

Discusses Seattle's tent city, and its marginalization by city authorities. Suggests a variety of options for developing the tent city, either small portables or a large superstructure. While this project is not necessarily geared towards housing the homeless, it aims to occupy a similarly marginalized site.

Butler, Lenneke and Frank den Oudsten. "Schroeder House: the work of Gerrit Rietveld between myth and metaphor," *Lotus International*, 1989 no. 60. pp32-57.

Schroeder House

Cook, Peter, Warren Chalk, Dennis Crompton, Ron Herron, David Green and Mike Webb. “Cut-out Puzzle,” *Archigram*, no. 7, 1966.

Two pages of cutout ‘living pods’, shed units, streets, and a triangular superstructure. The reader is invited to cut them out, design a community, photograph it and send it in. The vision of living pods that can be moved around and reconfigured resonates.

Green, David. “L. A. W. U. N.,” *Archigram*, no. 9, 1970.

Modular ‘bots’ provide for human needs, making houses unnecessary and allowing people to live in parks. The relevant idea is that services are not provided by spaces but by invisible modules that are deployed as necessary.

Herron, Ron and Bryan Harvey. “A Walking City,” *Archigram*, no. 5, 1964.

A vision of a city walking around on legs. This project proposes the same idea, but on a much smaller scale, with each building block walking around and interfacing with other building blocks to create new spaces.

Richardson, Phyllis. *XS: Big Ideas, Small Buildings*. New York: Universe Publishing, 2001.

Contains drawings, plans and a short description of a variety of projects with a small footprint. The chapter on portable designs is particularly relevant, as the description of a small sidewalk newspaper kiosk that unfolds into several different configurations and then collapses down to a small box at night.

"Summer Collection at Woburn Alley Flop Out 2," *Archigram*, no. 8, 1968.

"turnOn," http://www.alleswirdgut.cc/turnon/, June 5, 2002.

Alles Wird Gut's hamster house arranges the different elements of a room around a human-sized hamster wheel, so that when you are sitting on the toilet in the bathroom, the bathtub is over your head. To get to the bathtub, you walk up the wall, and it rotates down to you. The prefabricated rooms are arranged along a cylinder, so that you move along the cylinder from

room to room, and then rotate a room to move around within it.

Appendix A: Thesis Presentation Comments

[These are the comments made by the jurors at my final thesis review, as transcribed by Ellen Do.]

MASTERS THESIS PRESENTATIONS

Spring Quarter 2003

May 14,15,16 - ROOMS ARCH 202 AND 135

WEDNESDAY AM, MAY 14

Jurors: Ed Weinstein, Weinstein/Copeland
Anne Schopf, Mahlum
Jay Deguchi, Suyama, Peterson, DeGuchi Architects
Lucia Pirzio-Biroli, Studio Ecktypos

Moderator: Peter Cohan

12:00: Michael Weller, espresso blocks

Committee: Do, Gross, Nicholls, Wei-Chih Wang

Comments:

* The project can go 2 directions, ideal and reality.

* If going for the reality, comfort should be considered, hard brick furniture would not be comfortable to sit on and use.

* Could consider using a soft material.

* This is a thought provoking thesis. You are asking questions. There is no doubt about the intellect and talent in this project.

* The project is carried out nicely. No one would have the real answers. The thesis is getting all of us to believe it, and to think about all the possibilities.

* Your scenario is in urban setting. However, it appeared that it may be more suitable for a suburban instead of urban setting. There would be lots of space to play with different block configurations.

* Shouldn't you have more than one type of block? For example, there are different kinds of CMU blocks, not just one type.

* The scenario may be one 'container' that can have different uses. The question is what kind of rooms, places to occupy, can you create.

* The Espresso Blocks will be suitable for frontier situations, for example, outer space.

* Is your focus on efficiency? What about time saved for construction, and cost effectiveness?

* This project can be nicely combined with the previous presentation of the Tent City, easily built and portable.

* People have psychological needs. The Blocks may be able to address different environmental perceptions, but you need to consider human needs in a space.

* There is something about how we will be shaping habitation. People bring baggage into the environment.

* This project is maybe asking about the social impact of and fundamental change of how we perceive habitat.

* Think out of the box. Think about what it means for construction technique, and architectural program. What different type of social program and applications can it support/do you want to engage in?

* Think about how generic code can do multiple things. Once wireless devices get into the workspace, how would the building respond?

* This gives tremendous space for the future. For example, one can change the wall colors or texture according to your feeling. The room could be a new visionary environment. For example, it could be a conference room, and once you are in, the room understands that you are having a teleconference, it would automatically show you display screens, connect you, and you don't need to do any complicated setup. Another example: if you miss your children, the blocks will call up the school for you to check on them or bring up the images of their activities on your wall.

* This provides incredible opportunity for space layouts. You should think about modular blocks, not just one kind, but a series of different ones, so that can create different types of architecture with one single type or more. There could be complexities if for example, you have 3 elements.

* Humans are a species with psychological responses. With innovation, maybe some of the issues can be addressed.

* There is new social class that is mobile. The Archigram reference of 40 years ago is nice. It was meant to be an idea, ideology, not really intended to be built, but promote the asking of questions. This thesis has the same spirit.

* You never know. There is new urbanism. ;-)

* This project has phenomenal inventiveness. There is a new class of people that have not existed before. And mobile homes are spreading.

* If you are going for the practical route. The gasket idea is good. But it seems to only address static concerns and not all of the mechanical parts.

* What do you do when you want to replace a window? Can you have curtains, to let air in? (Transparent bricks?)

* Who will use Espresso Blocks? Would it be for cool buildings, loft apartment types? Frontier applications make sense.

* It could also potentially be used to build shelters, emergency tents, for rescue. There might be military uses for this. You should not give up once you graduate. You should keep pursuing this. There is whole future waiting for you, and I won't be surprised if you can find money, maybe the military will be interested.

* Can the Blocks address the issue of drawers, and maybe a new kind of fabric?

* The room could probably create holograms easily. The blocks can figure out where to project images, and meetings with virtual environments.

* People will always have stuff. There should be place for stuff.

* A storage room can be perceived as a big box for stuff.

* What's the possible other scale of the project? Right now it's same as a CMU, but should it be much bigger, a mobile, suitcase? Should it be much smaller, like nano blocks? What's the biggest block you can handle?

* People are more familiar with the hot desk concept.

* People are nomadic. All the students have on average moved 7 times in the last 3 years!

* How will you do temperature control? Would it become cooler inside a block?

* This project is wildly successful.

* It is fabulous. We are all convinced that it can work. We are looking at the model in front of us, and the story you told us. We keep moving into alternatives to make this work.

* This is impressive. You have a successful presentation, and animation, and powerful little thing (full mock-up). The animation showing what the blocks can do really works to demonstrate the idea.

* This may be a good tool for astronauts in outer space to control building of machines, measure the terrain, or build habitats.

* Where is the reference /is there a reference to starbucks? ;-)

* Where is the espresso?

* Have you thought about playing with the 'speed' notion of espresso? Quick, build it fast?

Appendix B: Baristas Unite!

[This is an essay I wrote for an architecture theory class that was the inspiration for the espresso blocks project.]

The automation and globalization of the industrial workforce has promoted the rise of a new class of college-educated professionals to manage the distant resources of multinational corporations, and stifled the demand for industrial labor. The majority of the citizens who do not have a professional degree or accreditation are now left with few employment options. Most of the available jobs involve working for near minimum wage at some sort of corporate service chain, a Kinko's, a Starbuck's, or, for the desperate, a McDonald's.

Due to rising density and land values in urban areas, and the lower wages paid by the corporate service industry, both parents in a typical family are forced to work full time to pay the rent on a house or apartment with a large kitchen no one has time to use. And single-parent service class families are increasingly unable to afford housing, or services. Even many professionals are forced by the rising cost of urban living to give up private practice and take jobs with large corporate firms.

There is hope that the members of the new service class can escape this dilemma by adopting the strategy of the espresso cart. By providing the services demanded by the urban economy, while occupying a minimum amount of expensive urban real estate, from a mobile platform that can move with the changing urban economy, citizens of urban areas can escape the yoke of multinational corporations by embracing capitalism on a smaller scale.

By applying the constraints of the espresso cart to their housing as well, these entrepreneurs could take advantage of a type of architecture where space is placed at a premium. Rather than having several human-size spaces to accommodate different functions, one human-size space is able to change to provide a variety of different functions. One experiment at creating this kind of space is Alles Wird Gut's hamster house[33], where the elements of several different rooms are arranged around the edge of a circle,

[33] http://www.alleswirdgut.cc/turnon/

and to go to another room, rather than leaving the space, you walk up the wall like a hamster in a wheel, and the components of the next room rotate down to the floor level.

The hamster house conserves floor space by providing only one element of each room at a time. It would be more desirable to have one whole room on the floor at a time, with all the other rooms stored away. This goal can be achieved through adopting the technologies of modular robotics to construct reconfigurable spaces.

Taking a modular robotics approach will also allow the building system to gracefully accommodate multiple occupants and higher density. Several modules could be linked together to create one larger space, or could provide larger spaces for each module by sharing walls in a hive formation. The zone of subsidized parking between the street and the sidewalk will be reclaimed by the disenfranchised to build new layer of urban infill.

Because of the marginal nature of this relationship to the city, the modules will be off of the city grid. Each module will store its own water, to be refilled either by purchasing from nearby buildings, trading with other modules in the hive, or appropriating it from hydrants or unprotected water lines. Modules will generate their own power, from a fuel cell or solar cell, or purchase it from other modules in the hive. Greywater will be filtered and reused. Sewage will be composted and either used in gardens or sold. This decentralized strategy for providing services will create a market for services within each hive, and allow a great number of modules to be absorbed into a city with minimal impact.

Appendix C: Simulation Source Code

BlockBuilder

```
Option Explicit

' build autocad block definitions to be inserted
Public Sub build(ByVal blocksize As Double)

    ' build tube block
    tubes blocksize

    ' build big arm block
    bigarms blocksize

    ' build small arm block
    smallarms blocksize

End Sub
Private Sub tubes(ByVal blocksize As Double)

    ' blue tubes
    onetube blocksize, "tubes", acBlue

    ' red selected tubes
    onetube blocksize, "selectedtubes", acRed

End Sub
' build tubes blocks
Private Sub onetube(ByVal blocksize As Double, ByVal tubename As
String, ByVal tubecolor As AcColor)

    ' block variables
    Dim tubeblock As AcadBlock
    Dim insertatzero(0 To 2) As Double

    ' array of tube boxes
    Dim tubebox(0 To 2) As Acad3DSolid

    ' create block
    Set tubeblock = ThisDrawing.Blocks.add(insertatzero,
tubename)

    ' build tube boxes for each axis
    Dim axis
    For axis = 0 To 2

        ' box variables
        Dim bigbox As Acad3DSolid
        Dim smallbox As Acad3DSolid

        ' dimension variables
        Dim center(0 To 2) As Double
        Dim length As Double, width As Double, height As Double

```

```
        ' draw tubebox
        settubedimensions blocksize, axis, center, length, width,
height
        Set tubebox(axis) = tubeblock.AddBox(center, length,
width, height)

        ' draw bigbox
        setbigdimensions blocksize, axis, center, length, width,
height
        Set bigbox = tubeblock.AddBox(center, length, width,
height)

        ' draw smallbox
        setsmalldimensions blocksize, axis, center, length,
width, height
        Set smallbox = tubeblock.AddBox(center, length, width,
height)

        ' subtract bigbox and smallbox from block
        tubebox(axis).Boolean acSubtraction, bigbox
        tubebox(axis).Boolean acSubtraction, smallbox

    Next

    ' join tubes together
    tubebox(0).Boolean acUnion, tubebox(1)
    tubebox(0).Boolean acUnion, tubebox(2)

    ' set color
    tubebox(0).color = tubecolor

End Sub
' build big arm blocks
Private Sub bigarms(ByVal blocksize As Double)

    ' create block for each axis
    Dim axis
    For axis = 0 To 2

        ' block variables
        Dim bigblock As AcadBlock
        Dim insertatzero(0 To 2) As Double

        ' box variables
        Dim bigbox As Acad3DSolid
        Dim smallbox As Acad3DSolid

        ' dimension variables
        Dim center(0 To 2) As Double
        Dim length As Double, width As Double, height As Double

        Set bigblock = ThisDrawing.Blocks.add(insertatzero,
"bigarm" & axis)

        ' draw bigbox
        setbigdimensions blocksize, axis, center, length, width,
height
        Set bigbox = bigblock.AddBox(center, length, width,
height)

```

```
        ' draw smallbox
        setsmalldimensions blocksize, axis, center, length,
width, height
        Set smallbox = bigblock.AddBox(center, length, width,
height)

        ' move smallbox down axis
        Dim newcenter(0 To 2) As Double
        newcenter(0) = center(0)
        newcenter(1) = center(1)
        newcenter(2) = center(2)
        newcenter(axis) = center(axis) - blocksize * 1 / 6
        smallbox.move center, newcenter

        ' subtract smallbox from bigbox
        bigbox.Boolean acSubtraction, smallbox

        ' set color
        bigbox.color = acYellow

    Next

End Sub
' build small arm blocks
Private Sub smallarms(ByVal blocksize As Double)

    Dim axis
    For axis = 0 To 2

        ' block variables
        Dim smallblock As AcadBlock
        Dim insertatzero(0 To 2) As Double

        ' box variables
        Dim smallbox As Acad3DSolid

        ' dimension variables
        Dim center(0 To 2) As Double
        Dim length As Double, width As Double, height As Double

        ' create block
        Set smallblock = ThisDrawing.Blocks.add(insertatzero,
"smallarm" & axis)

        ' draw smallbox
        setsmalldimensions blocksize, axis, center, length,
width, height
        Set smallbox = smallblock.AddBox(center, length, width,
height)

        ' set color
        smallbox.color = acGreen

    Next

End Sub
' set dimensions to draw outer tube box
Private Sub settubedimensions(blocksize, axis, center, length,
width, height)
```

```
    If axis = 0 Then ' x axis tube

        ' set center
        center(0) = blocksize * 1 / 2
        center(1) = blocksize * 1 / 4
        center(2) = blocksize * 1 / 4

        ' set length, width and height
        length = blocksize
        width = blocksize / 2
        height = blocksize / 2

    ElseIf axis = 1 Then ' y axis tube

        ' set center
        center(0) = blocksize * 3 / 4
        center(1) = blocksize * 1 / 2
        center(2) = blocksize * 3 / 4

        ' set length, width and height
        length = blocksize / 2
        width = blocksize
        height = blocksize / 2

    ElseIf axis = 2 Then ' z axis tube

        ' set center
        center(0) = blocksize * 1 / 4
        center(1) = blocksize * 3 / 4
        center(2) = blocksize * 1 / 2

        ' set length, width and height
        length = blocksize / 2
        width = blocksize / 2
        height = blocksize

    End If

End Sub
' set dimensions to draw outer big arm box
Private Sub setbigdimensions(blocksize, axis, center, length,
width, height)

    If axis = 0 Then ' x axis arm

        ' set center
        center(0) = blocksize * 5 / 12
        center(1) = blocksize * 1 / 4
        center(2) = blocksize * 1 / 4

        ' set length, width and height
        length = blocksize * 5 / 6
        width = blocksize * 5 / 12
        height = blocksize * 5 / 12

    ElseIf axis = 1 Then ' y axis arm

        ' set center
        center(0) = blocksize * 3 / 4
        center(1) = blocksize * 5 / 12
```

```
            center(2) = blocksize * 3 / 4

            ' set length, width and height
            length = blocksize * 5 / 12
            width = blocksize * 5 / 6
            height = blocksize * 5 / 12

        ElseIf axis = 2 Then ' z axis arm

            ' set center
            center(0) = blocksize * 1 / 4
            center(1) = blocksize * 3 / 4
            center(2) = blocksize * 5 / 12

            ' set length, width and height
            length = blocksize * 5 / 12
            width = blocksize * 5 / 12
            height = blocksize * 5 / 6

        End If

    End Sub
    ' set dimensions to draw small arm box
    Private Sub setsmalldimensions(blocksize, axis, center, length,
    width, height)

        If axis = 0 Then ' x axis arm

            ' set center
            center(0) = blocksize * 7 / 12
            center(1) = blocksize * 7 / 24
            center(2) = blocksize * 1 / 4

            ' set length, width and height
            length = blocksize * 5 / 6
            width = blocksize * 1 / 4
            height = blocksize * 1 / 3

        ElseIf axis = 1 Then ' y axis arm

            ' set center
            center(0) = blocksize * 17 / 24
            center(1) = blocksize * 7 / 12
            center(2) = blocksize * 3 / 4

            ' set length, width and height
            length = blocksize * 1 / 4
            width = blocksize * 5 / 6
            height = blocksize * 1 / 3

        ElseIf axis = 2 Then ' z axis arm

            ' set center
            center(0) = blocksize * 1 / 4
            center(1) = blocksize * 17 / 24
            center(2) = blocksize * 7 / 12

            ' set length, width and height
            length = blocksize * 1 / 3
            width = blocksize * 1 / 4
```

```
        height = blocksize * 5 / 6

    End If

End Sub
```

EspressoBlock

```
Option Explicit

' block variables
Private blocklist As EspressoBlockList
Private blockindex As Integer
Private blockposition() As Integer ' corner of block
Private blocktubesbox As AcadBlockReference
Private blockarm(0 To 2, 0 To 1) As EspressoBlockArm ' arm array
' access arm array
Public Property Get arm(ByVal axis As Integer, ByVal isbig As
Boolean) As EspressoBlockArm
    If isbig Then
        Set arm = blockarm(axis, 1)
    Else
        Set arm = blockarm(axis, 0)
    End If
End Property
' block index
Public Property Get index() As Integer
    index = blockindex
End Property
' build block
Public Sub build(parentlist As EspressoBlockList, ByVal
startposition, ByVal startindex As Integer)

    ' set private variables
    Set blocklist = parentlist
    blockindex = startindex
    blockposition = startposition

    Dim insertpoint(0 To 2) As Double
    insertpoint(0) = blockposition(0)
    insertpoint(1) = blockposition(1)
    insertpoint(2) = blockposition(2)

    ' build tubes
    Set blocktubesbox =
ThisDrawing.ModelSpace.InsertBlock(insertpoint, "tubes", 1, 1, 1,
0)

    ' build arms
    Dim buildaxis
    For buildaxis = 0 To 2
        Dim buildisbig
        For buildisbig = 0 To 1

            Set blockarm(buildaxis, buildisbig) = New
EspressoBlockArm
            blockarm(buildaxis, buildisbig).build Me, buildaxis,
buildisbig

        Next
    Next

    ' regenerate drawing
    ThisDrawing.Regen True
```

```

End Sub

Public Property Let hilite(ByVal lite As Boolean)

    If lite Then
        blocktubesbox.name = "selectedtubes"
    Else
        blocktubesbox.name = "tubes"
    End If

    ' regenerate drawing
    ThisDrawing.Regen True

End Property
' access block list
Public Property Get list() As EspressoBlockList
    Set list = blocklist
End Property
' adds a new block to the face of this block
Public Sub addblock(ByVal addaxis As Integer, ByVal addisbig As
Boolean)

    ' new block and blockposition array
    Dim newblock As New EspressoBlock
    Dim newposition() As Double

    Me.getadjacentposition addaxis, addisbig, newposition

    ' build block
    newblock.build blockpallet, newposition

    ' add block to blockpallet
    blockpallet.addblock newblock

    ' add block to control form
    ControlForm.BlockListBox.AddItem "block " &
(ControlForm.BlockListBox.ListCount)

    ' resolve control form buttons
    ControlForm.resolvebuttons

End Sub
Public Property Get position(ByVal xyz As Integer) As Double
    position = blockposition(xyz)
End Property
Public Sub setposition(ByVal xyz As Integer, ByVal newposition As
Integer)

    ' get offset
    Dim xyzoffset As Integer
    xyzoffset = newposition - blockposition(xyz)

    ' move tubes box
    Dim frompoint(0 To 2) As Double
    Dim topoint(0 To 2) As Double
    topoint(xyz) = xyzoffset
    blocktubesbox.move frompoint, topoint

    ' remove old position from position array
```

```
    blocklist.removeposition blockindex

    ' set blockposition
    blockposition(xyz) = newposition

    ' add new position to position array
    blocklist.addposition blockindex

    ' set arm positions
    Dim setaxis
    For setaxis = 0 To 2
        Dim setisbig
        For setisbig = 0 To 1

            blockarm(setaxis, setisbig).move xyz, xyzoffset

        Next
    Next

    ' regenerate drawing
    ThisDrawing.Regen True

End Sub
Public Sub getposition(newposition)

    ' set newposition equal to blockposition
    ReDim newposition(0 To 2)
    newposition(0) = blockposition(0)
    newposition(1) = blockposition(1)
    newposition(2) = blockposition(2)

End Sub
' return true if not latched to more than one block
Public Property Get isfree() As Boolean

    ' latch counter
    Dim latchcounter As Integer
    latchcounter = 0

    ' loop through tubes and check if they are latched
    Dim axiscounter
    For axiscounter = 0 To 2
        Dim bigcounter
        For bigcounter = 0 To 1

            ' increment latchedcounter if arm is latched
            If blockarm(axiscounter, bigcounter).islatched Then
                latchcounter = latchcounter + 1
            End If

        Next
    Next

    If latchcounter > 1 Then
        isfree = False
    Else
        isfree = True
    End If

End Property
```

```
' return the one other block this block is latched to if it is free
Public Sub isalmostfree(besidesarm As EspressoBlockArm, alsoblock)

    alsoblock = Empty

    Dim axis
    For axis = 0 To 2
        Dim isbig
        For isbig = 0 To 1
            If blockarm(axis, isbig).islatched And Not besidesarm Is blockarm(axis, isbig) Then

                ' if this isn't the first latched block
                If Not isempty(alsoblock) Then
                    alsoblock = Empty
                    Exit Sub
                End If

                ' if the block this block is latched to is free
                If blockarm(axis, isbig).latchedto.block.isfree Then
                    Set alsoblock = blockarm(axis, isbig).latchedto.block
                End If

            End If
        Next
    Next

End Sub
```

EspressoBlockArm

```
Option Explicit

' arm variables to hold property values
Private armbox As AcadBlockReference
Private armblock As EspressoBlock
Private armaxis As Integer
Private armisbig As Boolean
Private armextended As Integer
Private armislatched As Boolean
Private armlatchedto As EspressoBlockArm

' build block arm
Public Sub build(parentblock As EspressoBlock, ByVal buildaxis As
Integer, ByVal buildisbig As Boolean)

    ' set private variables
    Set armblock = parentblock
    armaxis = buildaxis
    armisbig = buildisbig
    armextended = 0
    armislatched = False

    ' tube box name
    Dim blockname As String
    blockname = "smallarm" & armaxis
    If armisbig Then blockname = "bigarm" & armaxis

    ' insert point
    Dim insertat() As Double
    armblock.getposition insertat

    ' insert arm box
    Set armbox = ThisDrawing.ModelSpace.InsertBlock(insertat,
blockname, 1, 1, 1, 0)

End Sub
Public Sub move(ByVal moveaxis As Integer, ByVal moveoffset As
Double)

    ' point arrays default to zero
    Dim movefrom(0 To 2) As Double
    Dim moveto(0 To 2) As Double

    ' set moveto point
    moveto(moveaxis) = moveoffset

    ' move box
    armbox.move movefrom, moveto

End Sub
Public Property Get islatched() As Boolean
    islatched = armislatched
End Property
' move arm in or out
Public Sub extend(ByVal inorout As Boolean)
```

```
    ' if inorout is true, move arm in, if it is false move arm out
    Dim inco As Integer
    inco = 1
    If inorout Then inco = -1

    ' coefficient of bigness
    Dim bigco As Integer
    bigco = 1
    If armisbig Then bigco = -1

    ' distance to move arm
    Dim offset As Integer
    offset = inco * armblock.list.max / armblock.list.step

    ' check if new distance is within bounds
    Dim minimum As Integer
    minimum = 0
    If Not armisbig And armislatched Then minimum = armblock.list.size * 1 / 4
    If (offset > 0 And armextended = armblock.list.max) Or (offset < 0 And armextended = minimum) Then

        ControlForm.debugprint "can't move arm that far"
        Exit Sub

    ElseIf offset + armextended > armblock.list.max Then

        ' adjust offset within bounds
        offset = armblock.list.max - armextended

    ElseIf offset + armextended < 0 Then

        ' adjust offset within bounds
        offset = 0 - armextended

    End If

    ' check if this arm is latched to another
    If armislatched Then

        ' move this block or the one it is latched to
        If Not resolvelatchedto(offset * bigco) Then

            ' blocks can't move so this arm can't move
            ControlForm.debugprint "can't move arm because blocks are stuck"
            Exit Sub

        End If

    Else ' if it isn't latched

        ' check if new distance will set latch
        resolvesetlatch offset

    End If

    ' update armextended
    armextended = armextended + offset
```

```

    ' from and to points
    Dim fromposition(0 To 2) As Double
    Dim toposition(0 To 2) As Double

    ' set to position to offset distance along axis
    toposition(armaxis) = offset * bigco

    ' move arm box
    armbox.move fromposition, toposition

    ' regenerate drawing
    ThisDrawing.Regen True

End Sub
' returns true if one of the blocks is moved and false if neither
block can be
Private Function resolvelatchedto(ByVal offset As Integer) As
Boolean

    ' neither block moved yet
    resolvelatchedto = False

    ' check if this block is free
    If armblock.isfree Then

        ' if it's free, try to move block opposite to the arm
offset
        If resolvemoveblock(Me, -1 * offset) Then
resolvelatchedto = True

    ' otherwise check if latchedto block is free
    ElseIf armlatchedto.block.isfree Then

        ' try to move latchedto block offset distance along axis
        If resolvemoveblock(armlatchedto, offset) Then
resolvelatchedto = True

    Else

        Dim alsoblock
        armblock.isalmostfree Me, alsoblock
        If Not isempty(alsoblock) Then
            If resolvemovetwoblocks(Me, alsoblock, -1 * offset)
Then resolvelatchedto = True
        Else
            armlatchedto.block.isalmostfree armlatchedto,
alsoblock
            If Not isempty(alsoblock) Then
                If resolvemovetwoblocks(armlatchedto, alsoblock,
offset) Then resolvelatchedto = True
            End If
        End If
    End If

End Function
' returns true if block is moved and false if it isn't
Private Function resolvemoveblock(movearm As EspressoBlockArm,
ByVal offset As Integer) As Boolean

```

```
    'block hasn't been moved yet
    resolvemoveblock = False

    ' check for collision
    If armblock.list.resolvecollision(armaxis, offset,
movearm.block) Then

        ' no collision, so move parent block offset distance
along axis
        movearm.block.setposition armaxis,
movearm.block.position(armaxis) + offset

        resolvemoveblock = True

    End If

End Function
' returns true if block is moved and false if it isn't
Private Function resolvemovetwoblocks(movearm As
EspressoBlockArm, ByVal alsoblock, ByVal offset As Integer) As
Boolean

    'block hasn't been moved yet
    resolvemovetwoblocks = False

    Dim alsoespressoblock As EspressoBlock
    Set alsoespressoblock = alsoblock

    ' check for collision
    If armblock.list.resolvecollision(armaxis, offset,
movearm.block) _
    And armblock.list.resolvecollision(armaxis, offset,
alsoespressoblock) Then

        ' no collision, so move parent block offset distance
along axis
        movearm.block.setposition armaxis,
movearm.block.position(armaxis) + offset
        alsoblock.setposition armaxis,
alsoblock.position(armaxis) + offset

        resolvemovetwoblocks = True

    End If

End Function

' test the latchposition of block
Private Sub resolvesetlatch(offset As Integer)

    ' check if there is a block on axis within a block length
    Dim latchedtoblock
    armblock.list.resolvelatchposition armaxis, armisbig,
armblock, latchedtoblock

    ' if there is no block, quit
    If isempty(latchedtoblock) Then Exit Sub

    ' get arm to latch to
    Dim maybearm As EspressoBlockArm
```

```
    Set maybearm = latchedtoblock.arm(armaxis, Not armisbig)

    ' get distance between blocks
    Dim distance As Integer
    distance = Abs(armblock.position(armaxis) -
latchedtoblock.position(armaxis)) - armblock.list.size

    ' get offset adjustment
    Dim maybeoffset As Integer
    maybeoffset = distance - (armextended + maybearm.extended -
(armblock.list.size / 6))

    ' if arms are close enough to latch
    If maybeoffset <= offset Then

        ControlForm.debugprint "arms latched"

        ' reset offset distance
        offset = maybeoffset

        ' set this arm and latchedto arm as latched
        armislatched = True
        Set armlatchedto = maybearm
        armlatchedto.islatched = True
        Set armlatchedto.latchedto = Me

        ' resolve control form buttons
        ControlForm.resolvebuttons

    End If
End Sub

Public Property Get block() As EspressoBlock
    Set block = armblock
End Property
Public Property Get axis() As Integer
    axis = armaxis
End Property
Public Property Let islatched(ByVal setto As Boolean)
    armislatched = setto
End Property
Public Property Set latchedto(latcharm As EspressoBlockArm)
    Set armlatchedto = latcharm
End Property
Public Property Get latchedto() As EspressoBlockArm
    Set latchedto = armlatchedto
End Property

Public Property Get extended() As Integer
    extended = armextended
End Property
Public Sub latchtoggle()
    If armblock.isfree Or armlatchedto.block.isfree Then
        ControlForm.debugprint "can't unlatch from last block"
    ElseIf armislatched Then

        ' unlatch this arm and the one it is latched to
        armlatchedto.unlatch
        Me.unlatch

```

```
        ' resolve buttons
        ControlForm.resolvebuttons

        ' regenerate drawing
        ThisDrawing.Regen True

    End If
End Sub
' unlatch and retract arm
Public Sub unlatch()

    ' unlatch arm
    armislatched = False

    ' coefficient of bigness
    Dim bigco As Integer
    bigco = -1
    If armisbig Then bigco = 1

    ' move arm
    Dim frompoint(0 To 2) As Double
    Dim topoint(0 To 2) As Double
    topoint(armaxis) = extended * bigco
    armbox.move frompoint, topoint

    ' set extended to 0
    armextended = 0

End Sub
```

EspressoBlockList

```
Option Explicit

' dynamic array of all blocks by index
Private byindexlist() As EspressoBlock

' top-level x position array
Private xlist As PositionArray

' block variables
Private blocksize As Integer
Private blockspace As Integer
Private selectedblock As EspressoBlock
Private armmax As Integer
Private armstep As Integer
Private blockaction As EspressoBlockActions


' create first block and add it to list
Public Sub zero()

    ' set block variables
    blocksize = 12 ' i wouldn't touch that if i were you
    blockspace = 1 ' or this either
    armmax = blocksize * 2 / 3
    armstep = 8

    ' build block blocks
    Dim bb As New BlockBuilder
    bb.build blocksize

    ' create x position array
    Set xlist = New PositionArray

    ' load actions
    Set blockaction = New EspressoBlockActions
    blockaction.loadlist Me

    ' build first block
    ReDim byindexlist(0)
    Set byindexlist(0) = New EspressoBlock
    Dim startposition() As Integer
    ReDim startposition(0 To 2) ' defaults to 0,0,0
    byindexlist(0).build Me, startposition, 0

    ' set it as selected block
    Set selectedblock = byindexlist(0)

    ' add block to position array
    addposition 0

    ' add block to control form block list
    ControlForm.BlockListBox.AddItem "block 0"

    ' select block 0
    Me.selected = byindexlist(0)

```

```
    ' resolve control form buttons
    ControlForm.resolvebuttons

End Sub
Public Sub addblock(axis As Integer, isbig As Boolean)

    ' get selected block position
    Dim startposition() As Integer
    selectedblock.getposition startposition

    ' get distance to offset new block
    Dim offset As Integer
    offset = blocksize + blockspace
    If isbig Then offset = offset * -1

    ' offset new block position along axis
    startposition(axis) = startposition(axis) + offset

    ' expand block index list
    ReDim Preserve byindexlist(UBound(byindexlist) + 1)

    ' create and build new block
    Set byindexlist(UBound(byindexlist)) = New EspressoBlock
    byindexlist(UBound(byindexlist)).build Me, startposition,
UBound(byindexlist)

    ' add block to position array
    addposition UBound(byindexlist)

    ' add block to control form block list
    ControlForm.BlockListBox.AddItem "block " &
UBound(byindexlist)

    ' select new block
    selected = byindexlist(UBound(byindexlist))

    ' latch to adjacent blocks
    latchall

End Sub
' returns true if the space adjacent to this face is empty
Public Property Get emptyadjacent(ByVal axis As Integer, ByVal
isbig As Boolean, Optional firstblock) As Boolean

    ' no block yet
    firstblock = Empty

    Dim startpoint() As Integer, endpoint() As Integer
    ReDim startpoint(0 To 2)
    ReDim endpoint(0 To 2)

    ' set off-axis start and end positions
    Dim loopaxis
    For loopaxis = 0 To 2
        startpoint(loopaxis) = selectedblock.position(loopaxis) -
blocksize
        endpoint(loopaxis) = selectedblock.position(loopaxis) +
blocksize
    Next
```

```
    ' set axis start and end positions
    If isbig Then

        startpoint(axis) = selectedblock.position(axis) - (2 *
blocksize + blockspace)
        endpoint(axis) = selectedblock.position(axis) -
(blocksize + blockspace)

    Else

        startpoint(axis) = selectedblock.position(axis) +
blocksize + blockspace
        endpoint(axis) = selectedblock.position(axis) + 2 *
blocksize + blockspace

    End If

    ' test if adjacent space is empty
    emptyadjacent = emptyspace(startpoint, endpoint, firstblock)

End Property
Public Sub addposition(ByVal index As Integer)

    Dim bp() As Integer ' block position
    byindexlist(index).getposition bp

    ' if there is not already a y array at this x, create y and z
arrays
    If Not xlist.isat(bp(0)) Then

        ' make y array
        xlist.makelistat bp(0)

        ' make z array
        xlist.listat(bp(0)).makelistat bp(1)

    ' if there is not a z array at this y, create one
    ElseIf Not xlist.listat(bp(0)).isat(bp(1)) Then

        ' make z array
        xlist.listat(bp(0)).makelistat bp(1)

    End If

    ' add this block index to z array at this position
    xlist.listat(bp(0)).listat(bp(1)).setindexat bp(2), index

End Sub
Public Sub removeposition(ByVal index As Integer)

    Dim bp() As Integer ' block position
    byindexlist(index).getposition bp

    ' remove this block from z array
    xlist.listat(bp(0)).listat(bp(1)).removeat bp(2)

    ' if the z array is empty, remove it from this y array
    If xlist.listat(bp(0)).listat(bp(1)).nomore Then

        xlist.listat(bp(0)).removeat bp(1)
```

```

        ' if the y dictionary is empty, remove it from this x
dictionary
        If xlist.listat(bp(0)).nomore Then

            xlist.removeat bp(0)

        End If

    End If

End Sub
' return block at a point, or empty if there is no block there
Private Sub onpoint(point, pointblock As Variant)

    ' is there a block at this position?
    If xlist.isat(point(0)) Then
        If xlist.listat(point(0)).isat(point(1)) Then
            If
xlist.listat(point(0)).listat(point(1)).isat(point(2)) Then

                ' return block
                Set pointblock =
byindexlist(xlist.listat(point(0)).listat(point(1)).indexat(point
(2)))

            End If
        End If
    End If

End Sub
' find block within distance on axis to this arm
Private Sub onaxis(startpoint, ByVal axis As Integer, ByVal
distance As Integer, axisblock As Variant)

    Dim newpoint() As Integer
    newpoint = startpoint

    Dim offset
    For offset = 0 To distance

        ' shift newpoint offset distance along axis
        newpoint(axis) = startpoint(axis) + offset

        ' test this position for blocks
        onpoint newpoint, axisblock

        ' if there was a block there, stop
        If Not isempty(axisblock) Then Exit Sub

    Next

End Sub
' return true if there is no block in a space
Private Function emptyspace(startpoint, endpoint, Optional
firstblock) As Boolean

    ' no blocks yet
    emptyspace = True
    firstblock = Empty
```

```

    Dim newpoint() As Integer
    newpoint = startpoint

    ' test if there is a y array at each x position
    Dim xoffset
    For xoffset = 0 To endpoint(0) - startpoint(0)

        newpoint(0) = startpoint(0) + xoffset

        If xlist.isat(newpoint(0)) Then

            ' test if there is a z array at each y position
            Dim yoffset
            For yoffset = 0 To endpoint(1) - startpoint(1)

                newpoint(1) = startpoint(1) + yoffset

                If xlist.listat(newpoint(0)).isat(newpoint(1))
Then

                    ' test the z axis for blocks
                    onaxis newpoint, 2, endpoint(2) -
startpoint(2), firstblock
                    If Not isempty(firstblock) Then

                        ' there is a block in the way
                        emptyspace = False
                        Exit Function

                    End If
                End If
            Next
        End If
    Next

End Function
Public Property Get size() As Integer
    size = blocksize
End Property
Public Property Get space() As Integer
    space = blockspace
End Property
Public Property Get byindex(index As Integer) As EspressoBlock
    Set byindex = byindexlist(index)
End Property
Public Property Get selected() As EspressoBlock
    Set selected = selectedblock
End Property
Public Property Let selected(newblock As EspressoBlock)

    ' unhilite old block
    selectedblock.hilite = False

    ' set selected block
    Set selectedblock = newblock

    ' hilite new block
    selectedblock.hilite = True

```

```
    ControlForm.debugprint "block position: " &
selectedblock.position(0) & ", " & selectedblock.position(1) & ",
" & selectedblock.position(2)

End Property
' max distance to extend arm
Public Property Get max() As Integer
    max = armmax
End Property
' number of steps to take extending arm
Public Property Get step() As Integer
    step = armstep
End Property
' returns true if there is no block in the way within offset
distance
Public Function resolvecollision(ByVal axis As Integer, ByVal
offset As Integer, resolveblock As EspressoBlock) As Boolean

    Dim startpoint() As Integer, endpoint() As Integer
    ReDim startpoint(0 To 2)
    ReDim endpoint(0 To 2)

    ' set off-axis start and end positions
    Dim loopaxis
    For loopaxis = 0 To 2
        startpoint(loopaxis) = resolveblock.position(loopaxis) -
blocksize
        endpoint(loopaxis) = resolveblock.position(loopaxis) +
blocksize
    Next

    ' set axis start and end positions
    If offset < 0 Then

        startpoint(axis) = resolveblock.position(axis) -
(blocksize) + offset
        endpoint(axis) = resolveblock.position(axis) -
(blocksize)

    Else

        startpoint(axis) = resolveblock.position(axis) +
(blocksize)
        endpoint(axis) = resolveblock.position(axis) +
(blocksize) + offset

    End If

    ' test if adjacent space is empty
    resolvecollision = emptyspace(startpoint, endpoint)

End Function
' if there is a block wihin a block length of arm, return it
Public Sub resolvelatchposition(ByVal axis As Integer, ByVal
isbig As Boolean, latchblock As EspressoBlock, latchedtoblock As
Variant)

    ' get block position
    Dim startpoint() As Integer
    latchblock.getposition startpoint
```

```

    ' set axis start position
    If isbig Then
        startpoint(axis) = startpoint(axis) - (2 * blocksize +
blockspace)
    Else
        startpoint(axis) = startpoint(axis) + blocksize +
blockspace
    End If

    ' check for block on axis
    onaxis startpoint, axis, blocksize + blockspace,
latchedtoblock

End Sub
' select next block
Public Sub selectnext(ByVal axis As Integer, ByVal isbig As
Boolean)

    Dim nextblock
    ' if this arm is latched to another arm, select it
    If selectedblock.arm(axis, isbig).islatched Then

        ' select block
        Me.selected = selectedblock.arm(axis,
isbig).latchedto.block

        ' resolve control form
        ControlForm.resolvebuttons

    ' if there is a block next to this one, select it
    ElseIf Not emptyadjacent(axis, isbig, nextblock) Then

        ' select block
        Me.selected = nextblock

        ' resolve control form
        ControlForm.resolvebuttons

    End If

End Sub
Public Property Get action() As EspressoBlockActions
    Set action = blockaction
End Property
' latch to every adjacent block
Private Sub latchall()

    ' block to latch to in loop
    Dim nextblock

    Dim axis
    For axis = 0 To 2
        Dim isbig
        For isbig = 0 To 1

            ' if there is a block next to this one, latch to it
            If Not emptyadjacent(axis, isbig, nextblock) Then

                ' latch blocks together
```

```
                If isbig Then

                    ' extend new block arm
                    Do While Not nextblock.arm(axis,
False).islatched
                        nextblock.arm(axis, False).extend False
                    Loop

                Else

                    ' extend selected block arm
                    Do While Not selectedblock.arm(axis,
False).islatched
                        selectedblock.arm(axis, False).extend
False
                    Loop

                End If

            End If

        Next
    Next

End Sub
```

PositionArray

```
Option Explicit

Private positivelist()
Private negativelist()
' return list at a position
Public Property Get listat(ByVal position As Integer) As
PositionArray
    If position < 0 Then
        Set listat = negativelist(Abs(position))
    Else
        Set listat = positivelist(position)
    End If
End Property
' return index at a position
Public Property Get indexat(ByVal position As Integer) As Integer
    If position < 0 Then
        indexat = negativelist(Abs(position))
    Else
        indexat = positivelist(position)
    End If
End Property
' return true if there is something at this position
Public Property Get isat(ByVal position As Integer) As Boolean
    isat = True
    If position < 0 Then
        If Abs(position) > UBound(negativelist) Then
            isat = False
        ElseIf isempty(negativelist(Abs(position))) Then
            isat = False
        End If
    Else
        If position > UBound(positivelist) Then
            isat = False
        ElseIf isempty(positivelist(position)) Then
            isat = False
        End If
    End If
End Property
' set a position to an index value
Public Sub setindexat(ByVal position As Integer, ByVal index As
Integer)

    ' extend array to include position
    extendto position

    ' set array at position to index
    If position < 0 Then
        negativelist(Abs(position)) = index
    Else
        positivelist(position) = index
    End If

End Sub
' create a new position array at a position
Public Sub makelistat(ByVal position As Integer)

```

```
    ' extend array to include position
    extendto position

    ' create new position array at position
    If position < 0 Then
        Set negativelist(Abs(position)) = New PositionArray
    Else
        Set positivelist(position) = New PositionArray
    End If

End Sub
' remove position from array and shrink array to next member
Public Sub removeat(ByVal position As Integer)

    If position < 0 Then
        position = Abs(position)

        ' set this position to empty
        negativelist(position) = Empty

        If position = UBound(negativelist) Then

            ' find next non-empty position
            Do While isempty(negativelist(position)) And position
> 1
                position = position - 1
            Loop

            ' shrink array
            ReDim Preserve negativelist(1 To position)

        End If

    Else

        ' set this position to empty
        positivelist(position) = Empty

        If position = UBound(positivelist) Then

            ' find next non-empty position
            Do While isempty(positivelist(position)) And position
> 0
                position = position - 1
            Loop

            ' shrink array
            ReDim Preserve positivelist(0 To position)

        End If

    End If

End Sub
' returns true if this array contains no lists or indexes
Public Property Get nomore() As Boolean

    ' if there is more than one item on list, it's not empty
    If UBound(negativelist) > 1 Or UBound(positivelist) > 0 Then
        nomore = False
```

```

    ' if there is only one item, but it's not empty, list isn't
empty
    ElseIf Not isempty(negativelist(1)) Or Not
isempty(positivelist(0)) Then
        nomore = False

    ' otherwise list is empty
    Else
        nomore = True
    End If

End Property

' if position is outside array boundaries, extend array
Private Sub extendto(ByVal position As Integer)

    If position < 0 Then
        position = Abs(position)

        If position > UBound(negativelist) Then

            ' extend lower bound to position
            ReDim Preserve negativelist(1 To position)

        End If
    Else
        If position > UBound(positivelist) Then

            ' extend upper bound to position
            ReDim Preserve positivelist(0 To position)

        End If
    End If

End Sub

Private Sub Class_Initialize()
    ReDim positivelist(0 To 0)
    ReDim negativelist(1 To 1)
End Sub
```