Making Self-Disassembling Objects with Multiple Components in the Robot Pebbles System

Kyle Gilpin, Kent Koyanagi, and Daniela Rus

***Abstract*— This paper describes several novel algorithms for shape formation by subtraction in programmable matter systems. These algorithms allow the simultaneous formation of multiple different shapes from a single block of host material. The resulting shapes are allowed to intertwine in arbitrarily complex ways. We also present a proof that the algorithms operate correctly to form the desired shapes. Finally, we show experimental results from close to 100 trials using both the Robot Pebbles hardware and a unique software simulator. Multiple trials of several different experiments demonstrate the algorithms operating correctly.**

I. Introduction

We present two provably-correct algorithms for shape formation by subtraction in the Robot Pebbles programmable matter system. These algorithms allow the system to simultaneously form multiple, intricate, intertwining shapes that could fill a multitude of mechanical roles by acting as joints, hinges, gears, or fasteners. In general, programmable matter systems are composed of small, intelligent modules that are able to form a variety of macroscale objects in response to external commands or stimuli. In our system, shapes are formed by deconstructing an initial block of material to eliminate unnecessary pieces. We call this approach self-disassembly or subtraction [1]. In the same way a sculptor removes the extra stone from a block of marble to reveal a statue, our system subtracts modules to form the goal structure. The system can make functional objects such as tools, containers, linkages, and support systems, or objects for entertainment like the Tetris pieces shown in Figure 1.

Our ultimate goal is to create a system of grain-sized modules that can form arbitrary structures with a variety of material properties on demand by selectively making and breaking connections. Given a bag of this Smart Sand, one can shake it in order to form arbitrary shapes with the modules. The modules inside first crystallize into a regular structure and then self-disassemble in an organized fashion to form the requested object. The object can be retrieved by reaching into the bag and brushing off the extra modules. When the user is done with the object, he returns it to the bag where it disintegrates back to its component modules. The free modules are incorporated into the universal structure of the bag and can be reused at a later time. Such a system would be useful for an astronaut on an inter-planetary mission or a scientist isolated at the South Pole. Even for the average mechanic or surgeon, having access to arbitrary, task-specific tools would be valuable for inspection and manipulation in tight spaces.

K. Gilpin, and D. Rus are with the Computer Science and Artificial Intelligence Lab, MIT, Cambridge, MA, 02139, K. Koyanagi is in the Class of 2012 at Caltech. kwgilpin@csail.mit.edu, koyanagi@caltech.edu, rus@csail.mit.edu.

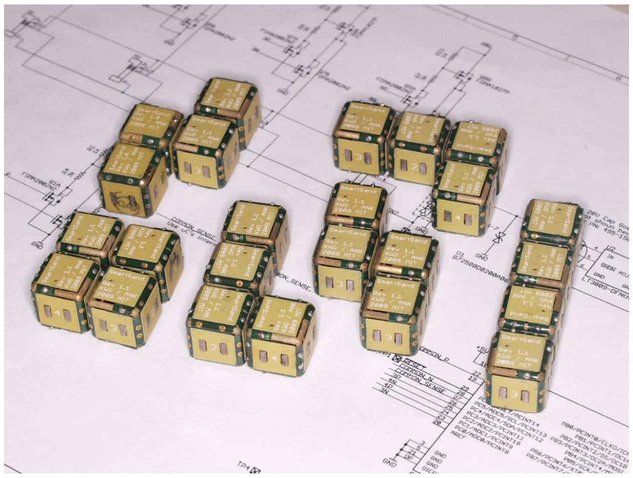

Fig. 1. Twenty-eight modules assembled in a 4-by-7 grid can be told to disassemble into all 6 possible Tetris pieces in one pass.

A. Motivation

When using programmable matter to form objects, one of the major challenges is conveying a description of the object to be formed to the relevant modules. One approach is to globally broadcast a description of the shape to be formed. In a small system, this approach is tractable, but the communication cost scales as $O(n^2)$–in a system of n modules, a total of n messages is exchanged among all neighbors, with n bits per message–regardless of the size of the object to be formed. We are aiming for a system of *Smart Sand* in which each module is on the order of a cubic millimeter. A typical claw hammer with a volume of 240cc, would require over 240,000 grains of Smart Sand to form, and the initial block of material would need to be several times this size. Given that communication requires time and power, the cost of broadcasting the shape description in such a system would be incredibly inefficient and difficult to implement.

As an alternative, this paper presents the design an implementation of shape distribution and disassembly algorithms which transmit only a minimal amount of information to a subset of all modules. These algorithms guarantee that all modules in the initial block of material will correctly be informed whether they are a part of a goal shape. Furthermore, our disconnection algorithm relies on distributed leaf-to-root deconstruction of an arbitrary spanning tree to ensure that each module breaks its connections to its neighbors in an ordered manner that guarantees the formation of the specified goal shape. The paper offers proof that the algorithms function correctly. Finally, the paper presents the results of 70 simulations and 27 hardware experiments to demonstrate that the algorithms function correctly in practice.

B. Related Work

Our research builds on previous work in programmable matter, self-assembly, and self-reconfiguring robotics, all of which grew out of the modular robotics field. Typically, modular robots have been categorized as either chain-style [2]–[4], or lattice-style [5]–[7]. Often the categorization of a robot is ambiguous and there are other systems, like the Digital Clay project [8], that lack any innate actuation capability and rely on a user to rearrange the modules. There are many interesting systems which rely on stochastic self-assembly with rigid modules to create shapes [9]–[12]. More recent research [13] has investigated scaling the size of a self-assembled object based on the number of modules available. Unlike our system, these approaches focus on shape formation by self-assembly rather than self-disassembly.

Other research has focused more directly on the concept of programmable matter. The catoms concept [14], proposes using spheres to form reconfigurable 3D structures. Existing catom prototypes use electromagnets affixed to the circumference of cylindrical PCB assemblies [14] or electrodes patterned on miniature hollow $Si0_2$ cylinders [15] to achieve 2D reconfiguration. Theoretical research has previously investigated the use of sub-millimeter intelligent particles as 3D sensing and replication devices [16]. Finally, the system described in [17] demonstrates 'virtual' programmable matter by forming a paintable display.

A limited amount of past research has focused specifically on the algorithms employed in self-disassembling systems [16], [18], [19]. The work presented here builds on our previous work in [1], [20] on the Robot Pebbles hardware and algorithms. There are unique hardware challenges associated with the Pebbles, but the work presented here shows how algorithms can be designed to overcome these challenges to use self-disassembly as a process to create multiple, interlocking objects from a single block of raw material.

II. Experimental Context and Capabilities

As this paper is concerned primarily with new subtraction algorithms for multiple shape distribution and disconnection in programmable matter systems, we will only give a brief summary of the module hardware and system architecture.

The algorithms in this paper use the Robot Pebbles system described in great detail in our ICRA 2010 paper [1]. The most important aspect of each module is that it must be able to autonomously communicate and bond with its immediate neighbors. More specifically, the Pebbles, as shown in Figure 2, are 12mm cubes that weigh 4.0g. Each is formed by wrapping a flexible printed circuit around a brass frame. Each Pebble contains an ATMega328 microprocessor, an energy storage capacitor, and four electropermanent (EP) magnets

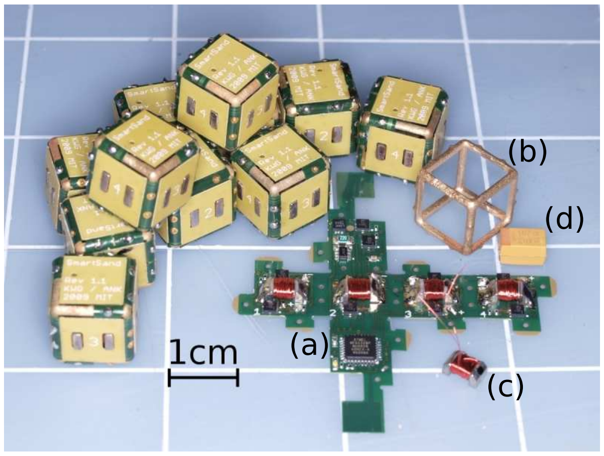

Fig. 2. The Robot Pebbles are 12mm cubes with four active faces. They are formed by wrapping a flex circuit (a) around a brass frame (b). Four electropermanent (EP) magnets (c) provide latching, communication, and power transfer capabilities to each module. Since the modules lack batteries, a $100\mu F$ capacitor (d) fills the interior to provide a reservoir of charge.

that are used for inter-module latching, communication, and power transfer among neighboring modules. The EP magnets are crucial to the Pebbles' operation. They are solid-state, genderless connection mechanisms capable of supporting over 80 times the weight of a single module and only consume power when switching on or off. Once switched, they consume no power and maintain their state for years.

The EP magnets also form the only inter-module communication link in the system. When two EP magnets are mated, they form a 1:1 isolation transformer that enables inductive communication at 9600baud. One limitation in our design is that a Pebble, due to coil driver space constraints, can only communicate with one neighbor at a given time. As a result, the Pebbles randomly split their time communicating with all neighbors. Because a given Pebble is never guaranteed to be listening for incoming messages on a given face at a given time, a module typically must make several attempts to successfully transmit a message.

The Pebbles do not contain batteries. Instead, electrical power is passed from one module to its neighbors through the two poles of each EP magnet. Because each inter-module electrical connection has a resistance of approximately 3Ohms, the $100\mu F$ energy storage capacitor inside of each module forms a charge reservoir that is utilized when switching the state of their connectors, which can only be done while the module is part of a connected system.

Shape formation by subtraction is a multiple step process that moves through six major phases: localization, neighbor discovery, bonding, virtual sculpting, shape distribution, and disconnection. The details of the localization, neighbor discovery, and bonding phases are not important in the context of this paper. After bonding, the modules have assembled into a close-packed lattice that forms the initial block of material that will be sculpted. Within this block, each module knows its position and whether it has neighbors. To enable the virtual sculpting process, the modules transmit their state information through a series of reflection messages back to the user's PC so that the sculptor may choose which to include in the final structure and which to discard. The result of this sculpting is a series of inclusion messages that will convey the desired shape to the structure during the shape distribution phase. Once all modules know whether they are included in the final structure, disconnection commences and all unnecessary bonds between neighboring modules are broken in order to arrive at the desired set of goal shapes. For more information on this process, consult [18], [20]

III. Multiple Shapes

In this section we present an algorithm that controls and optimizes the formation of multiple shapes by sculpting an initial block of connected material. While prior work [18] has shown that self-disassembly can form a particular shape from an initial block of material, the previous algorithm was only able to form a single shape during each iteration of the self-disassembly process, and the resulting shape had to include a unique root module. The new algorithm removes these restrictions. It can form multiple shapes that are contiguous or separated by any number of unused modules. This flexibility allows the sculpting of objects with interlocking sub-parts and internal degrees of freedom.

The shape distribution algorithm operates by transmitting a single inclusion message to each module in the initial structure that is destined to be a part of a goal shape. Modules not included in any goal shape do not receive an inclusion message. Modules assume, by default, that they are not included in the final structure. Inclusion messages originate from the sculptor's PC, and, once in the structure, they create and follow a dynamic *inclusion chain* constructed from a constant amount of information per message. The algorithm avoids encoding the detailed path that each inclusion message must follow, and it avoids flooding the system with inclusion messages.

The total communication cost of the inclusion chain algorithm is $O(n^2)$ where n is the number of modules included in the final structure. This bound arises because for each of the n modules, the inclusion message that informs each module of its status may have to travel from the root module through $O(n)$ other modules. In contrast, using a shortest-path algorithm to route a message from the root to each included module also has a theoretical communication cost of $O(n^2)$ if there are no obstacles in the structure that could form effective local minima and a gradient descent approach is employed. Once one considers broken inter-module communication links and voids within the initial structure, the communication cost of the routing algorithm increases as each message must contain more specific routing instructions. Given the uncertainty over which approach will perform better on average, we choose the inclusion chain approach for its simplicity given the hardware's limited processing capabilities.

A. Inclusion Message Distribution

Inclusion messages are generated by the system's user, often with the help of a GUI. All inclusion messages, like all other messages, enter the initial block of modules through the root module's serial connection to the user's PC. As an inclusion message moves from a module to its neighbor, it extends the tail of an *inclusion pointer chain*. Figure 3 shows how inclusion messages follow this chain for a specified distance termed the *hop count*. Once a message has traveled the specified number of hops, it branches off of the chain in the specified *branch direction*. The hop count and branch direction are pieces of information carried by the message itself—they do not come from the modules in the structure. However, the modules in the structure do store the inclusion pointer chain. Each module only needs to remember where to redirect an incoming inclusion message with a hop count greater than one. The module that the message reaches after branching is included in the structure.

Inclusion messages carry additional pieces of information. First, each message contains an *ignore* field which may be used to counteract the message's typical effect at its destination module. This module, instead of assuming to be included in the final structure, effectively ignores the inclusion message. The advantage is that a module may be part of the inclusion pointer chain without being a part of the final shape. This allows the formation of an unlimited number of disjoint shapes from one initial block of material during a single self-disassembly process.

The second auxiliary piece of information carried by an inclusion message is the *group number*. When an inclusion message reaches its destination, the group number is assigned to the module. During the disassembly phase, if two included modules have different group numbers, they disconnect. Likewise, if their group numbers are identical, they remain bonded to each other. Group numbers will allow the formation of contiguous interlocking shapes including hinges, gears, and bearings as the module size is reduced.

In practice, inclusion messages are ASCII strings: **#INC,<hop count>,<branch dir.>,<ignore>,<group>**. Each module employs Algorithm 1 when processing an incoming inclusion message. When a module receives an inclusion message, it first checks the hop count (line 6). If that value is 0, the receiving cube is the intended destination. In addition, if the ignore flag is not set, the module records the fact that it should be a part of the final structure and saves the group number included in the message (lines 7–8). A hop count of 1 indicates that one of the receiver's immediate neighbors is the message's intended destination. The receiving module uses its own known rotation and the message's branch direction to determine the face that should retransmit the message after the hop count is set to 0 (line 10–11). This inclusion pointer direction is stored as part of the module's state (line 12). Finally, if a module receives an inclusion message with a hop count greater than 1, it decrements the hop count and then retransmits the message on the face indicated by the stored inclusion pointer direction (line 14). The algorithm terminates when the module receives a disassemble (#DIS) message.

Algorithm 1 Inclusion Message Processing Algorithm

```
1:  $included$ = **false**
2:  $incChainPtr$ = NULL
3:
4:  **repeat**
5:      wait for #INC msg. w/ hop count $HC$, branch dir. $BD$,
        ignore flag $IGN$, and group number $GRP$
6:      **if** $HC = 0$ **and** $IGN = 0$ **then**
7:          $included$ = TRUE
8:          $myGroup = GRP$
9:      **else if** $HC = 1$ **then**
10:         $txFace$ = branchDirToFace($BD$)
11:         queueINC($txFace$, $\infty$, 0, $BD$, $IGN$, $GRP$)
12:         $incChainPtr = BD$
13:     **else**
14:         queueINC($incChainPtr, \infty, HC-1, BD, IGN, GRP$)
15:     **end if**
16:     txQueuedMsgs()
17: **until** #DIS message received
```

The $queueXXX(txFace, retries, \ldots)$ function such as the one in line 11 places an XXX-type message in the transmit queue of the specified face. After the queues have been loaded with messages, calling the $txQueuedMsgs()$ function makes one attempt to transmit the queued messages to a module's neighbors. If the *retries* parameter passed to the queuing function is less than infinity, each call to $txQueuedMsgs()$ decrements this parameter by one. When all retries for a particular face have been exhausted, $txQueueIsEmpty(txFace)$ will return true.

Proving the correctness of the shape distribution algorithm requires a description of the shape one wishes to form. In our system, generating a description of the goal shape is facilitated by a GUI that allows the user to virtually sculpt the desired shape and then generates a list of inclusion messages that are transmitted to the root and distributed. In [18] we show that this approach is efficient and correct. While the prior proof did not incorporate the concept of ignored modules, their effect is minimal, and we will not repeat the proof here. Additionally, the group code carried in each inclusion message has no effect on the algorithm.

Figure 3 illustrates the propagation of eight inclusion messages as they form a simple wrench from a 3-by-4 block of material. As indicated by the text above the modules in Figure 3(a), the first inclusion message is **#INC,0,n/a,true,0**. The second inclusion message reaches module A with a hop count of 1, indicating that one of A's neighbors is to be included. The branch direction of this message is "up," so the hop count is decremented to 0 and the message is sent to E. Module A sets its inclusion chain pointer to E. The third message reaches A with a hop count of two, follows A's inclusion chain pointer to E, and obeys the message's branch direction by moving to the right and including module F. Jumping ahead, after Module L is included as shown in (f), the next inclusion message modifies module G's inclusion chain pointer from "up" to "down" to include C. Module K's inclusion chain pointer still points to module L, but the inclusion of modules C and D is unaffected.

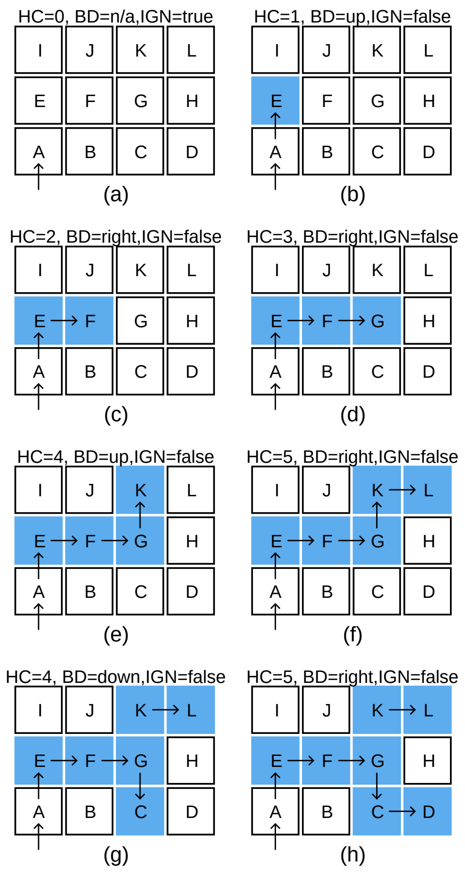

Fig. 3. Eight inclusion messages are used to create a simple wrench from a 3-by-4 block of programmable matter. The root module is labeled A. As modules are included in the final structure, they change from transparent to shaded. The arrows in the figure represent the inclusion chain pointers stored in the modules.

IV. Ordered Disconnection

One of the most challenging aspects of shape formation with the Robot Pebbles is the inter-module disconnection process that must occur after all modules know whether to remain as part of a finished object or to disconnect completely. The complexity in this disconnection process is caused by the fact that a module loses its ability to function once it breaks its connection with the neighbor supplying it with power. Furthermore, all modules that are dependent on that module for power will also lose power and will not be able to break additional magnetic bonds.

A. Parents, Children, and Neighbors

A tree can be used to represent how power is transmitted through an initial block of modules. Because it is connected to an external power supply, the module connected to the user's PC, is the root of this power transfer tree. Every other module in the tree has one parent, P. This parent is the neighbor that supplies the module with power. Conversely, every module to which a module supplies power is a child. Children of a module are denoted by the set C. Parents and children are both subsets of a module's magnetically bonded neighbors, N. In practice, current often follows many different paths from the root to any other implying that a module should have multiple parents. We disallow multiple parents by definition because they only serve to complicate the disassembly process. The key concept is that although different neighbors could also supply it with power, a module will never lose power as long as it is connected to its parent. These child and parent relationships are defined during the assembly process. A module is not allowed to become the parent of another until it has a parent of its own.

B. Child-to-Parent Disconnection

We have designed and implemented Algorithm 2 which ensures that an initial block of material can disassemble correctly—disconnecting bonds that should be broken and keeping those that should be preserved. In general, the disconnection algorithm operates by ensuring that a module has no children before disconnecting from its parent. If a module is a part of the same finished shape as its parent, the child uses a child removal message to inform its parent that it no longer needs to be considered a child. The algorithm uses sets N, P, and C to keep track of a module's bonded neighbors, parent, and children, respectively. We use two additional sets, G and K, that are initially empty. All neighbors from which a module has received group (#GRP) messages are added to G. If a neighbor's group matches the receiver's, the neighbor is added to the *keep* list, K.

The algorithm begins by waiting for a disassemble (#DIS) message from some neighbor. The face on which the message arrives is represented by the single-element set *rxFace*. When the module receives a #DIS message, it forwards it to its children (line 3). If the children do not receive this message, there is no guarantee that they will receive a #DIS from any other source. In line 4, the #DIS message is also sent to the module's neighbors that are not children to speed its propagation throughout the structure. To prevent two modules from repeatedly sending #DIS messages to each other, a #DIS message cannot be sent back to the module from which it was received. After the #DIS messages that are to be transmitted have been queued, we continue attempting to retransmit them until the transmit queues for all neighbors are empty (line 7). By passing infinity to *queueDIS*() in line 3 when the algorithm queues the #DIS messages for the module's children, the algorithm ensures that the *txQueuedMsgs*() function will never stop attempting to deliver the message until it is successful. This guarantees that module's children receive the #DIS message before the algorithm moves past line 7. In contrast, the DIS_RETRIES parameter in line 4 indicates that the *txQueuedMsgs*() function only makes a finite number of attempts to send the #DIS message to the module's non-child neighbors before the *txQueueIsEmpty*(*faceSet*) returns true. Once the children have received the #DIS message, the algorithm branches (line 9) depending on whether the module is included in any of the final structures being formed.

Algorithm 2 Disassembly Algorithm

```
1:  $G = K = \varnothing$
2:  wait for #DIS msg. to be rcvd. on face $rxFace$
3:  queueDIS($C \setminus rxFace$, $\infty$)
4:  queueDIS($N \setminus (rxFace \cup C)$, DIS_RETRIES)
5:  repeat
6:    txQueuedMsgs()
7:  until txQueueIsEmpty($N \setminus rxFace$)
8:
9:  if $included$ then
10:   queueGRP($N$, $\infty$, $myGroup$)
11:   repeat
12:     if #GRP msg. rcvd. (on face $rxFace$ specifying a
        neighbor in group $neighborGroup$) then
13:       $G = G \cup rxFace$
14:       if $neighborGroup = myGroup$ then
15:         $K = K \cup rxFace$
16:       else if $rxFace \neq P$ then
17:         queueUnlatch($rxFace$, $\infty$)
18:       end if
19:     end if
20:     txQueuedMsgs()
21:   until txQueueIsEmpty($N$) and $N = G$ and $N = (P \cup K)$
22:   if $myGroup \neq parentGroup$ or $parentGroup = \varnothing$
      then
23:     queueUnlatch($P$, $\infty$);
24:   else
25:     queueCLD($P$, $\infty$);
26:   end if
27: else
28:   queueUnlatch($N \setminus (C \cup P)$, $\infty$)
29:   queueGRP($C$, $\infty$, $myGroup$)
30:   repeat
31:     txQueuedMsgs()
32:   until $N = P$
33:   queueUnlatch($P$, $\infty$)
34: end if
35: repeat
36:   txQueuedMsgs()
37: until $N = \varnothing$
```

If the module is not included in the final structure, the relevant pseudo-code begins on line 27. It begins with the module queuing an unlatch message for all of its neighbors except its children and parent. Then, in line 29, it queues group (#GRP) messages for its children. Group messages simply inform the recipient of the transmitter's group. The infinity parameters passed to the *queueXXX*() functions in lines 28 and 29, would normally indicate that all of the unlatch and #GRP messages will be repeatedly transmitted until successfully received, but the receipt of an unlatch message purges the corresponding transmit queue; there is no point in continuing to transmit a message to a neighbor that is no longer present. (This behavior is not shown in the pseudo-code.) Now that the unlatch and #GRP messages are queued, the algorithm continually transmits them (line 30–32) until the module's only remaining neighbor is its parent. This elimination of neighbors results from the pseudo-code on line 33. Once a module's only neighbor is its parent, the module queues an unlatch message for the parent and waits (lines 35—37) until the message is received. When it is, the parent is removed from the module's list of neighbors, indicating that the module is now completely disconnected.

Alternatively, if the module is included in the final structure, it behaves differently. Lines 11–21 of Algorithm 2 form a repeat-until loop that eliminates all of a module's neighbors (except its parent) with group numbers that do not match its own. Before the loop begins in line 10, the algorithm queues #GRP messages for all neighbors, including the module's parent and children. The infinity parameter in line 10 ensures that these #GRP messages are sent repeatedly by the *txQueuedMsgs*() function until they are received. Once the loop beings, the algorithm checks for any incoming #GRP messages from its neighbors (line 12). If one is received, the neighbor from which it was received, *rxFace*, is added to G, the list of neighbors from which the module has received #GRP messages. If the #GRP message indicates that the neighbor's group is the same as the module's (line 14), then that neighbor is added to the module's keep list, K (line 15). If the neighbor's group number differs from the module's, and if the neighbor is not the module's parent, the module queues an unlatch message for the neighbor in line 17. This unlatch message overwrites any pending #GRP message destined for that neighbor.

This process of transmitting and receiving #GRP messages will eliminate all of a module's neighbors other than its parent and the neighbors in K. The loop ends in line 21, when the transmit queues of all neighbors have been emptied, we have received a #GRP message from each of our remaining neighbors, and our only remaining neighbors are $(P \cup K)$.

The module's children are eliminated over the course of the repeat-until loop in lines 22–26. To consider the disconnection process complete, the module only needs to inform its parent that it is no longer the parent's child. Exactly how the module informs its parent is determined by line 22. If the module's group is different than its parent's, (or if its parent does not belong to a group because it is not included in the final structure), the module queues an unlatch message for its parent. When this message is received, the two modules disconnect and the parent no longer considers the module its child. Alternatively, if the module and parent share the same group, the module sends a child removal (#CLD) message to its parent. This message simply informs the parent that the module has performed all necessary tasks and no longer requires a power source. As a result, the parent removes the module from its list of children, C. In this manner, the parent will eventually be left with no children so that it can sever the bond with its own parent.

C. Disconnection in Action

Figure 4 shows Algorithm 2 in action. In the figure, (a) represents the state of the modules after the #DIS message has been distributed and the modules have exchanged #GRP messages, but before any have begun to disconnect from their neighbors. The color of each module indicates the group to which it belongs. Module C is not included in any of the final structures. As shown by the transition from Figure 4(a) to (b), disconnection begins when the modules without children (E, F, and H) sever the relationships with their parents. In the case of E, its parent belongs to the same group, so it sends a #CLD message that breaks the parent relationship while maintaining the physical bond. Module F belongs to the same group as its neighbor, G, so G is in F's keep list. Given that all modules have already exchanged #GRP messages, F's state satisfies the conditional in line 21 of Algorithm 2. Consequently, F executes line 23 of the pseudo-code and transmits an unlatch message to its parent. Module H disconnects from its only non-parental neighbor, module D, because they are in different groups.

In subfigure (b), module D has no remaining neighbors except its parent, module C, which is not included in the structure, and therefore lacks a group code. Module D therefore satisfies the condition on line 22 of Algorithm 2, and it sends an unlatch message to C to disconnect from its parent. Without any remaining connection to the structure, the shape formed by modules D and E loses power in subfigure (c). Also shown in the transition from (b) to (c), module H sends its parent, G, a #CLD message leaving G without children.

As soon as G has no children, it disconnects from its parent because it is not included in the structure. After disconnecting, the shape formed by modules F, G, and H loses power. This disconnection is the only change as the system transitions from Figure 4(c) to (d). Once in the state shown by (d), module C realizes that it now has no children and no neighbors except its parent. Because it is not included, C can disconnect from its parent.

In Figure 4(e), module B has no children and no neighbors other than its parent, allowing it to send a #CLD message to its parent, A. It sends a #CLD message instead of an unlatch message because it knows that A is a part of the same group. When A receives this message, the parent-child bond between A and B is broken, transitioning the system to the state shown in subfigure (f). Finally, module A is left with only its parent, so it symbolically disconnects from the user's PC. At this point, all modules have lost power, but all of the necessary bonds have been maintained, and the desired shapes have been formed.

D. Correctness

The correctness of Algorithm 2 can be proven using induction on the height of the power transfer tree.

Theorem 1: Algorithm 2 results in a neighbor disconnection order that maintains power in each module until it has finished disconnecting from all unincluded neighbors and neighbors with group numbers different from its own.

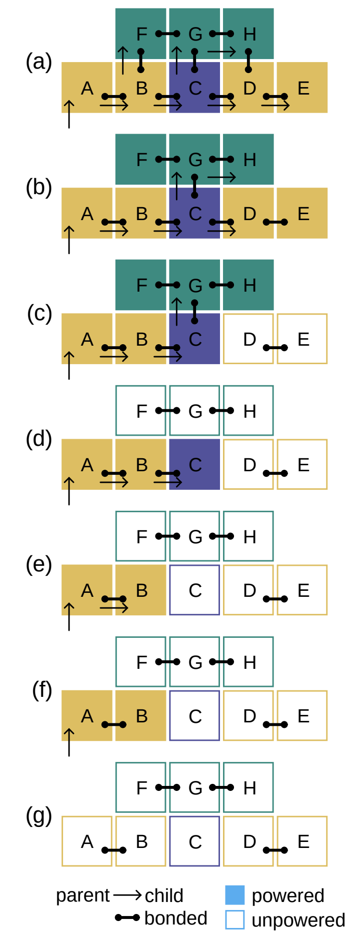

Fig. 4. Disconnection occurs in an orderly fashion. Each color of module in the figure represents an object that is to be formed from the initial block of material. As modules disconnect from the structure and lose power, they change from filled to empty. Before disconnecting, a module must ensure that all of its neighbors that depend on it for power have completed their disconnection process. Module A is the root.

Proof:

Base case: Tree height 1. A tree of height one has a single parent and multiple children that are the leaves of the tree. These children may or may not be magnetically connected neighbors. If a child has magnetically connected neighbors, it exchanges #GRP messages with them. If the groups of two neighbors are different, or if either neighbor is not included in any of the final structures, the neighbors unlatch (line 17). If they are in the same group, they do nothing. Once a leaf module handles its neighbors appropriately, the leaf severs its parent-child bond with the root. If the root and leaf are in different groups, the leaf sends the root an unlatch message (lines 22–23). Otherwise, the leaf sends the root a #CLD message that breaks the parent-child connection while maintaining the magnetic bond.

As we set out to prove in the theorem, the following occurs for each module before it potentially loses power by disconnecting from its parent: an unincluded module completely disconnects from all neighbors and then its parent; an included module with magnetically connected neighbors in groups other than its own detaches from these neighbors; and simultaneous with power loss, a module disconnects from its parent if their groups differ.

Induction: Assume the disconnection process operates correctly for trees of height n. To complete the proof, we need to show that the disconnection process works correctly for trees of height $n+1$. Following this approach, a tree of height $n+1$ can be viewed as a tree of height n with one additional set of leaves. These leaves may or may not be magnetically bonded with any other module in the entire tree. Whether or not they are bonded does not change how they act. Just as in the height 1 base case, the leaves exchange #GRP messages with their magnetically bonded neighbors and break their magnetic connection if they are in different groups or if either is not included in the final structure. Once this is complete, the leaves break (by unlatching or sending a #CLD message) their parent-child bond with their parent.

As in the base case, all leaves, before losing power, have disconnected from their neighbors as needed. Unincluded leaf modules have broken all of their magnetic connections. Included leaf modules have broken their magnetic connections with neighbors belonging to groups different than their own and maintained their connections with neighbors of the same group. With the leaves removed, the $n+1$ height tree is now an n height tree. ■

V. Experiments

We have performed a number of experiments in both simulation and hardware. The simulated experiments are used in place of hardware to test the algorithms and demonstrate scalability. Because the shape distribution and disconnection phases are distinct, we are concerned with the success of each. The first experiment we performed consisted of fully disassembling a 3-by-3 block of modules that did not contain any goal shapes. This is shown pictorially in Figure 5(a). In 12 of 15 hardware experiments, all bonds were broken as expected. In the other 3 three, there were 2, 3, and 4 unbroken bonds. In all three cases, the initial shape was poorly constructed and the modules far from the root did not align well with their neighbors. As a result, we believe communication failures, not the algorithm, led to the unbroken connections.

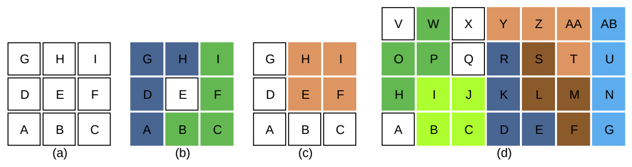

Fig. 5. We have used the Robot Pebbles to form a number of different shapes that test the ability of the hardware and algorithms to form multiple contiguous and discontiguous shapes.

We performed additional experiments with other goal shapes to test the system's ability to use the ignore and group fields of an inclusion message. The results for both the simulator and hardware appear in Table I. In the table, the shape distribution success rate is measured by observing which Pebbles know that they should be a part of goal structure. The disconnection success rate is the number of bonds that behaved as expected divided by the total number of bonds in the initial structure

TABLE I
EXPERIMENTS SHOW THE ALGORITHMS WORKING CORRECTLY.

Goal Shape(s)	*Sim / HW*	*Number Trials*	*Success Rate [%]*	
			Distribution	*Disconnection*
Figure 5(a)	Sim	15	N/A	100.0
	HW	15	N/A	95.0
Figure 5(b)	Sim	15	100.0	100.0
	HW	5	100.0	98.3
Figure 5(c)	Sim	15	100.0	100.0
	HW	5	100.0	96.7
Figure 5(d)	Sim	15	100.0	100.0
Figure 3(h)	Sim	10	100.0	100.0
	HW	2	100.0	97.1

The results in Table I show that the shape distribution algorithm works flawlessly in both simulation and hardware. We only see errors when performing disconnection experiments in hardware. Even so, the overall disconnection success rates are still good. This leads us to believe that the disconnection algorithm functions correctly, but that peculiarities of the hardware are interfering with its operation.

We have observed four particular hardware issues that affect the disconnection process. First, all modules are not exactly the same size. As a result, alignment errors can accumulate, resulting in marginal or no communication between neighbors. Second, during the assembly process, previously bonded modules are sometimes pulled out of position as new modules are added to the structure. This results in the connected modules losing power, resetting their states, and introducing inconsistencies in the system. Third, the disconnection process releases internal stresses as some of the magnets turn off. Given that we see the modules moving as they disconnect, we suspect that this may also result in modules temporarily losing power. Finally, the power supply sourcing power to the root module is current limited. When a module deactivates an EP magnet, it momentarily draws 4A. The simultaneous deactivation of many EP magnets during disconnection often pegs the power supply at its current limit, potentially preventing some modules from unlatching.

VI. Discussion

We have proposed two new algorithms for forming shapes through a process of disassembly. These algorithms efficiently distributed a description of the desired shape(s) throughout an initial block of material that is being sculpted. As programmable matter systems grow to include more and more modules, algorithms such as these will become absolutely necessary. In the future, we plan to perform experiments with additional modules, and we intend to explore different methods for shape formation including magnification of a miniaturized shape and replication of an original shape. We also plan to add 3D functionality to our system. While there are additional hardware challenges associated with 3D shape formation, the algorithms presented in this paper theoretically apply equally well in 3D. In practice, one must pay more attention to the alignment and disconnection order of a 3D system to ensure that extra modules can be removed while maintaining the structural integrity of the shape. With these algorithmic advances and additional miniaturization of hardware, we hope that *Smart Sand* will become a reality.

VII. Acknowledgments

The authors would like to thank the reviewers for their insightful feedback. This work is supported by DARPA and the US Army Research Office under grant number W911NF-08-1-0228, NSF EFRI grant number 0735953, Intel, and the NDSEG fellowship program.

References

[1] K. Gilpin, A. Knaian, and D. Rus, "Robot pebbles: One centimeter robotic modules for programmable matter through self-disassembly," in *ICRA*, May 2010.

[2] M. Yim, B. Shirmohammadi, J. Sastra, M. Park, M. Dugan, and C. Taylor, "Towards robotic self-reassembly after explosion," in *IROS*, November 2007, pp. 2767–2772.

[3] A. Castano, A. Behar, and P. Will, "The conro modules for reconfigurable robots," *Transactions on Mechatronics*, vol. 7, no. 4, pp. 403–409, December 2002.

[4] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji, "Automatic locomotion design and experiments for a modular robotic system," *Transactions on Mechatronics*, vol. 10, no. 3, pp. 314–325, June 2005.

[5] D. J. Christensen and K. Stoy, "Select a meta-module to shape-change the atron self-reconfigurable robot," in *ICRA*, May 2006, pp. 2532–2538.

[6] M. Koseki, K. Minami, and N. Inou, "Cellular robots forming a mechanical structure (evaluation of structural formation and hardware design of "chobie ii")," in *Distributed Autonomous Robotic Systems*, June 2004, pp. 131–140.

[7] G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff, "Evaluating efficiency of self-reconfiguration in a class of modular robots," *Journal of Robotic Systems*, vol. 13, no. 5, pp. 317–388, 1996.

[8] M. Yim and S. Homans, "Digital clay," WebSite. [Online]. Available: www2.parc.com/spl/projects/modrobots/lattice/digitalclay/index.html

[9] P. White, V. Zykov, J. Bongard, and H. Lipson, "Three dimensional stochastic reconfiguration of modular robots," in *Robotics Science and Systems*, June 2005.

[10] M. Tolley and H. Lipson, "Fluidic manipulation for scalable stochastic 3d assembly of modular robots," in *ICRA*, May 2010, pp. 2473–2478.

[11] S. Griffith, D. Goldwater, and J. M. Jacobson, "Robotics: Self-replication from random parts," *Nature*, vol. 437, p. 636, Sept 28 2005.

[12] N. Napp, S. Burden, and E. Klavins, "The statistical dynamics of programmed self-assembly," in *ICRA*, May 2006, pp. 1469–1476.

[13] M. Rubenstein and W.-M. Shen, "Automatic scalable size selection for the shape of a distributed robotic collective," in *IROS*, Oct 2010, pp. 508–513.

[14] S. Goldstein, J. Campbell, and T. Mowry, "Programmable matter," *IEEE Computer*, vol. 38, no. 6, pp. 99–101, 2005.

[15] M. E. Karagozler, S. C. Goldstein, and J. R. Reid, "Stress-driven mems assembly + electrostatic forces = 1mm diameter robot," in *IROS*, October 2009, pp. 2763–2769.

[16] P. Pillai, J. Campbell, G. Kedia, S. Moudgal, and K. Sheth, "A 3d fax machine based on claytronics," in *IROS*, Oct 2006, pp. 4728–4735.

[17] W. Butera, "Text display and graphics control on a paintable computer," in *Self-Adaptive and Self-Organizing Systems*, July 2007, pp. 45–54.

[18] K. Gilpin, K. Kotay, D. Rus, and I. Vasilescu, "Miche: Modular shape formation by self-disassembly," *IJRR*, vol. 27, pp. 345–372, 2008.

[19] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1996.

[20] K. Gilpin and D. Rus, "Modular robot systems: From self-assembly to self-disassembly," *IEEE Robotics and Automation Magazine*, vol. 17, no. 3, pp. 38–53, September 2010.