Interlocking Block Assembly With Robots

Yinan Zhang, Yotto Koga, and Devin Balkcom

***Abstract*— This article presents a design for interlocking blocks and an algorithm that allows these blocks to be assembled into desired shapes. During and after assembly, the structure is kinematically interlocked if a small number of blocks are immobilized relative to other blocks. There are two types of blocks: cubes and double-height posts, each with a particular set of male and female joints. Layouts for shapes involving thousands of blocks have been planned automatically, and shapes with several hundred blocks have been built by hand. This article also describes a method for assembling structures from blocks in parallel. As a proof of concept, a dual-robot system was used to assemble 48 blocks, forming an interlocking cube-like structure.**

***Note to Practitioners*—This article was inspired by existing work on interlocking joinery structures, modular robots, and construction robots. We present designs for two interlocking blocks that can be assembled into larger rigid structures. Blocks of this type are a promising future construction material. Only translation is needed to assemble the blocks, simplifying robotic assembly, and the mortarless construction allows for later disassembly and reuse of the blocks. We propose an algorithm that lays out blocks into desired shapes in series and developed a dual-robot system to assemble 48 blocks automatically. Our physical experiments show that joint manufacturing precision is critical to the ease of construction and the rigidity of the finished structure. We also present a layout algorithm that enables parallel assembly, allowing multiple robots to work on the same structure.**

***Index Terms*— Assembly, construction, interlocking structure, manipulation, manufacturing, material/parts handling, mechanisms.**

I. Introduction

THE goal of the work described in this article is to enable robotic assembly of large structures from blocks that interlock without the need for glue, cement, screws, or other connectors. Fig. 1 shows one model for which layouts and assembly plans were generated automatically by the presented algorithm. The motion of blocks is constrained by joints (See Fig. 2); later blocks reinforce and immobilize prior blocks. Each structure has a few blocks that can move, called keys. If the keys are fastened to the structure, the structure is rigidly interlocked.

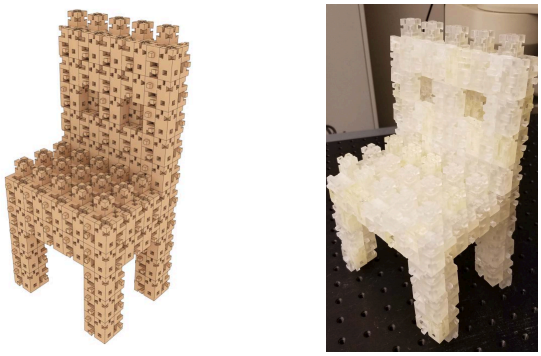
Manuscript received January 16, 2021; accepted February 23, 2021. Date of publication April 21, 2021; date of current version July 2, 2021. This article was recommended for publication by Editor L. Tapia upon evaluation of the reviewers' comments. This work was supported in part by the Seed Funding through the National Science Foundation (NSF) under Grant IIS-1813043 and in part by the Dubai Future Foundation. *(Corresponding author: Yinan Zhang.)*

Yinan Zhang and Devin Balkcom are with the Department of Computer Science, Dartmouth College, Hanover, NH 03755 USA (e-mail: yinan.zhang.gr@dartmouth.edu; devin.balkcom@dartmouth.edu).

Yotto Koga is with Autodesk AI Lab, San Francisco, CA 94111 USA (e-mail: yotto.koga@autodesk.com).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TASE.2021.3069742.

Digital Object Identifier 10.1109/TASE.2021.3069742



(a) (b)

Fig. 1. Interlocking chairs in simulation and assembled by hand. We used two kinds of blocks to build this structure. There is only one block removable on the top of the chair. (a) Chair model automatic laid out in simulation. (b) Chair assembled by hand.

A collection of rigid bodies is kinematically interlocked if there is no motion of the system that does not cause collision. For example, the pieces of a planar jigsaw puzzle are interlocked once assembled, if motion is restricted to the plane. The blocks presented in this article may be thought of as 3-D jigsaw pieces; the keys are the last pieces placed. Disassembly of the structure can be achieved by reversing the assembly order.

This article extends work presented in [1] to demonstrate the robotic construction of structures using two 6-DoF robot arms. We will explain in detail how depth cameras and deep neural networks are combined to sense block configurations and pick up blocks in random poses. This article also describes how to reorient a block to the orientation needed for assembly, by iteratively regrasping the block using the two arms.

Kinematic interlock presents some advantages over traditional connection methods, such as glue, cement, screws, nails, or friction locks. The interlocks may be structurally strong, allow simple assembly by robots, allow disassembly, and reuse of the components, and maybe suitable for underwater or other environments where adhesives are ineffective. Fabricating in parts may present some advantages over traditional 3-D printing. Individual components may be fabricated efficiently, packed for storage and transport, repaired or replaced as needed, and allow design changes.

The algorithm described in this article takes a voxelized 3-D model as input and finds an assembly plan such that the interlocked structure covers the specified voxels. There are two types of blocks: $1 \times 1 \times 1$ cubes and $1 \times 1 \times 2$ posts, with connectors arranged in a particular way. Assembly requires only translation motions.

The main insight of the block designs and layout algorithms of this article is that a collection of rigid blocks may be

interlocked if there is a cycle, or closed loop, of joints that permit translation only in conflicting directions. We design small structures containing only a handful of blocks, with the property that if the last block were to be glued to an adjacent block after assembly, an interlocked closed chain would be formed. Fig. 3(c) shows an example; block 4 in the figure is the key and may be removed only by translation out of the page.

We then show how this small structure may be extended by creating a new, overlapping cycle of blocks, such that if the key of the new cycle is glued, the new cycle attaches the prior key to the structure without the need for glue. In Fig. 3(c), the next cycle of blocks would hold block 4 in place. Larger structures are then built from these overlapping cycles.

A. Applications and Limitations

The work aims to design blocks and layout patterns that make robotic construction as easy as possible. Assembly of the blocks requires only orthogonal translations that may be accomplished in principle by robots with only three degrees of freedom. Any contiguous shape that can be described on a voxel grid can be build using the blocks.

This article focuses primarily on the geometry of a particular design of blocks and associated layout algorithms. As such, many critical issues that would need to be solved for a practical system have been neglected. Chief among these is the need for analysis of the rigidity and robustness of the final structures. Due to manufacturing limits, joints do not match perfectly, and for some shapes, small errors may add up, which leads to undesirable aggregate flexibility, as explored in [2]. The physical experiments conducted use 3-D printed blocks and are no more than a proof of concept, using a small number of blocks. Given the success of Lego blocks, we do believe that better manufacturing processes would allow extension to very large structures.

Although the algorithm presented can lay out essentially arbitrary voxelized structures, overhanging components of layers are not interlocked until a second identical layer is placed above. This means that some external, though temporary, means of support are needed during construction, just as in 3-D printing. Algorithms for placing support material efficiently can be found in [3]–[5]. Unlike 3-D printing, block assembly only requires temporary support until a substructure is interlocked. This observation could inspire special position-holding robots to provide necessary support for overhanging layers.

We would also like to gain a better understanding of how to design and analyze block types and layout algorithms. Effectively, the block types designed are the result of creative thought to generate small overlapping cycles, with posts that allow connections to the succeeding layer. Other forms of interlock might allow even simpler assembly or more robust structures; a primary objective of future work is to design mathematical or mental tools to find other block designs.

II. Related Work

Interlocking structures have a long history. Wood joints, such as the dovetail and mortise and tenon, are used in carpentry around the world; in China and Japan, complex interlocking designs have permitted the construction of wooden buildings with no screws or nails [6]. In the paleontology community, evidence has recently been presented that supports a hypothesis that the backbones of theropod dinosaurs interlocked to provide support for the extremely large body mass [7].

The concept of interlocking block assembly was previously presented in [8]. However, the technical work in this article is effectively entirely new. New block designs and layout algorithms enabled the reduction of the types of blocks needed from nine to two and have allowed structures that appear to be more robust and easier to assemble. This article also explores the construction of physical structures much larger than previously built (406 pieces compared to 64), as well as a more convincing demonstration of robotic assembly. New theoretical contributions include an analysis of how blocks may be assembled in parallel, speeding up assembly.

The present work is closest in spirit to Song *et al.* [9]–[14], which considers the problem of designing reusable components to be assembled into different forms relying on geometric constraints; the primary contribution of the current work is a universal block design and layout algorithm that allows construction of arbitrary geometries. Yao *et al.* [15] proposed a method for interactively designing joints for structures and analyzing the stability. Kong and Kimia [16] applied curve matching techniques for finding solutions for assembly of 2-D and 3-D interlocking puzzles; the layout algorithms considered in this article generate assembly motions together with the design. Wang *et al.* [17] proposed an algorithm to assemble convex blocks into a given freeform surface. Although the blocks presented have no joints or external connectors, they still found structurally stable and globally interlocking assemblies possible.

Robotic construction research dates back to the 1990s when Andres *et al.* [18] created a prototype, ROCCO, capable of gripping and laying bricks. The same robotic system was later applied to site assembly operations by Balaguer *et al.* [19]. More recent works include DimRob, a system with an industrial robot arm mounted on a mobile platform [20] used for construction tasks. This prototype was later developed into a mobile robot, In situ Fabricator, for construction at 1:1 scale [21]. Roombots [22] are another example of modular robots that can self-reconfigure into various furnitures, along with other manipulation and gripping capabilities. Kubits [23] is another example of 3-D self-reconfiguring modular robots.

Willmann *et al.* [24], for example, used autonomous flying vehicles to lift and position small building elements. Augugliaro *et al.* [25] demonstrated a system of multiple quadrocopters precisely laying out foam blocks forming a desired shape. Lindsey *et al.* [26] built cubic structures using quadrocopters. Augugliaro *et al.* [27] explored another approach of construction: quadcopters assembled a rope bridge capable of supporting people. Keating *et al.* [28] built a large mobile 3-D printer using a robot arm to extrude adhesive materials.

Instead of focusing on the robot control system to carry building elements, some researchers designed new building elements. Rus and Vona [29] developed crystalline, a modular robot with a 3-DoF actuation mechanism allowing it to make and break connections with other identical units; a

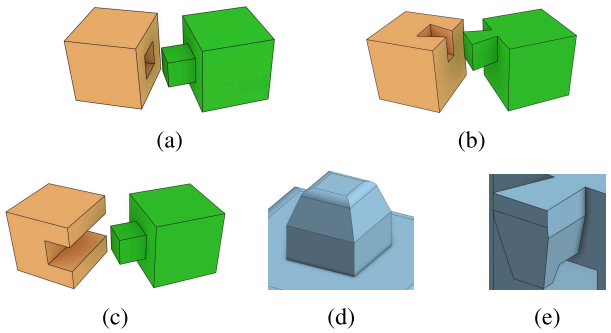


Fig. 2. Three different joint pairs and detailed design. (a) Mortise and tenon joints. (b) Dovetail joints. (c) Two-way joints. (d) Tenon joint. (e) Dovetail joint.

set of such robots form a self-reconfigurable robot system. White *et al.* [30] introduced two 3-D stochastic modular robot systems that are self-reconfigurable and self-assemble-able by successive bonding and release of free-floating units. Romanishin *et al.* [31] proposed a momentum-driven modular robot. SamBot, a cube-shaped modular robot with rotation mechanism, was introduced by Wei *et al.* [32]. Daudelin *et al.* [33] present a self-reconfigurable system that integrates perception, mission planning, and modular robot hardware. Tosun *et al.* [34] created a design framework for rapid creation and verification of modular robots.

Swarm robot can also be used to assemble structures at the microscale where controlling each individual robot is difficult. One approach is to use global control signals, such as gravity and magnetic field. Becker *et al.* [35]–[37] took this path and showed the feasibility of reconfiguring massive particle swarm robots with limited controls. The authors' own work [38] shows some reconfiguration tasks can be performed efficiently in a small space. Manzoor *et al.* [39] and Schmidt *et al.* [40] proposed parallel mechanisms to make the assembly even more efficient. Kilobot, a swarm of 1024 mobile robots, was introduced by Rubenstein *et al.* [41], along with algorithms for planning mechanisms allowing kilobots to form 2-D shapes.

Inspired by LEGO, Schweikardt and Gross [42] proposed a robotic construction kit, roBlocks, with programmable cubic blocks for educational purpose. With the recent development on deep learning, generative models based on graph-structured neural networks are also applied to assemble structures using LEGO [43].

III. Interlocking Blocks and Constraint Graph

In this section, we first introduce the three joint types used in the design. We also present a graph-based method for analyzing the relative movement of objects connected by these joints. Our design choices and layout algorithms are primarily inspired by this analysis. We found that globally interlocked large structures can be assembled from locally interlocked substructures and presented a high-level algorithm to achieve this goal. Details of how each part of the algorithm is implemented are discussed in later sections.

A block is a rigid body that has joints allowing assembly with other blocks, typically by sliding one block against the other using a simple translation. Fig. 2 shows three different joint pairs that we use to connect blocks. A mortise and tenon joint pair [Fig. 2(a)] allows blocks to be disassembled only in the nonpenetrating normal direction of the contact surface. A dovetail joint pair [see Fig. 2(b)] allows block motion only in a particular tangential direction. The third joint pair we use in the current design is a two-way joint [see Fig. 2(c)], which allows motions of associated blocks in both normal and tangential directions. The dovetail and mortise and tenon joints fully constrain rotational motion, but the two-way joint permits one rotational degree of freedom. These joints are chosen because of their simplicity in manufacturing and assembly effort. We manufactured the joints, so the front part in the insertion direction is thinner and thus reduces surface contacts, providing better error tolerance [see Fig. 2(d) and (e)].

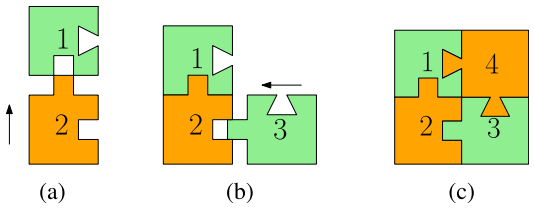


Fig. 3. Assembling a nearly interlocking four-block square. Arrow indicates the assembly direction. Block 4 is the key. (a) Tenon joint. (b) Tenon joint. (c) Dovetail joint assembled from top to down.

Blocks are assembled into a structure in order, and the last block assembled can be removed by reversing the most recent translation assembly motion. Therefore, the last block assembled must be attached to the structure using glue, friction, a screw, or some other external method; we call such a block a key.

Fig. 3 shows a 2-D projection of an interlocking structure assembled using blocks with dovetail and mortise and tenon joints. First, block 2 is assembled to block 1, using a tenon joint on the top of the blocks and moving block 2 in the positive y-direction, assuming a coordinate frame aligned with the page. Block 3 then slides in and connects with block 2 by another tenon joint. The final block is assembled from top to down, connecting blocks 1 and 3 using two dovetail joints and limiting blocks 2 and 3 to move in y negative or x positive directions.

A. Constraint Graph

To better understand how joints constrain motions, we represent a structure using a directed graph. Each vertex in the graph represents a block. A pair of directed edges are added between vertices corresponding to blocks that are in contact; $w(e_{i,j})$ denotes the set of permitted motions of j relative to i.

Consider a partition of the graph into some nonoverlapping subsets of vertices. A partition is separable if there exists a motion that satisfies constraints by all in-edges along the boundary of the subset of vertices. In this particular work, the block and joint design limit motions of every block to translations directly along axes, simplifying the analysis. (The two-way joint is used only as an auxiliary connection for blocks whose motions are already constrained to pure axis-aligned translations by other joints.)

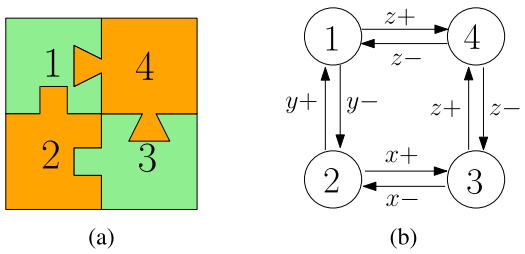


Fig. 4. Four-block interlocking structure and its graph representation. (a) 2×2 interlocking structure. Numbers indicate the order. Block 4 is the key. (b) Graph representation. Each edge allows some motions for associated blocks.

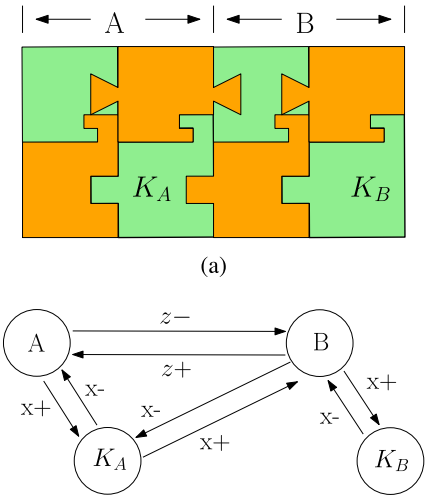


Fig. 5. Any interlocking substructure can be viewed as two nodes in the graph, which simplifies the graph representation. (a) Two interlocking substructures connected by a dovetail joint and a mortise & tenon joint. (b) Forming a larger interlocked structure from two interlocked structures.

Fig. 4(b) shows an example of a constraint graph for the previous example of a four-block interlocking structure. Consider partitioning the structure into two parts $\{1, 2\}$ and $\{3, 4\}$. These parts are inseparable since $w(e_{2,3}) \cap w(e_{1,4}) = \{x+\} \cap \{z+\} = \emptyset$. By checking more partitions of the structure, we find that only $\{1, 2, 3\}$ and $\{4\}$ are separable. If block 4 is attached to either of its neighbors, the structure is rigid.

We call a structure k-interlocked, if when k keys are attached to neighbors, no partition is separable. The example structure is 1-interlocked with block 4 as the key.

Analyzing a large structure gets difficult when there are a large number of blocks because the number of possible partitions on the graph increases exponentially. Fortunately, proof that a complete structure is interlocked can be accomplished in a hierarchical fashion, by first showing that smaller components are interlocked and then using those components to build larger interlocking structures.

Fig. 5 shows an example of how a larger interlocking structure can be built from smaller interlocking substructures. Interlocking substructures A and B are similar to those shown in Fig. 4(a); the careful eye may note some additional geometry on each block representing dovetail joints attached from the side; these joints provide some redundant constraints that add rigidity to the final structure.

The keys of the substructures are K_A and K_B and are not considered to be part of A and B. To show that the entire structure is 1-interlocked by K_B, it is sufficient to consider only partitions that separate K_A from A or K_B from B since A and B act as rigid bodies if their keys are not separated. Fig. 5(b) shows the graph representation.

B. Overview of the Layout Algorithm

The example above suggests an approach to constructing large interlocked structures. We can build four-block interlocked squares and use a second interlocked square to build an eight-block rectangle [see Fig. 5(a)]. Inductively, we can extend the rectangle as far as we like by adding additional squares to the end; we call such a structure a segment.

Intuitively, some additional connections might be added to connect segments to form a flat structure that we will call a layer. The 3-D volumes may then be constructed from stacks of layers. Fig. 6 shows a conceptual picture, with the single key block of each new larger structure shown in red.

The remainder of this article addresses the details needed to allow the implementation of this process. How should segments interconnect to form a layer? How should layers interconnect? How should joints be arranged on blocks to allow creating segments and layers from only a few types of blocks? How should layers be automatically shaped to allow construction of geometries more interesting than large cubical volumes?

Algorithm 1 presents an overview of the layout algorithm. A model is assembled layer-by-layer from bottom to top. Each layer itself is an interlocking substructure that also constrains the movement of a previously built layer's key(s). The details will be discussed in later sections, as indicated by the section numbering indicated in the algorithm; the reader may wish to only skim the algorithm on first reading. For now, it is worth noting that the input to the algorithm is a voxelized model describing the desired output shape. Each voxel is further subdivided into eight subvoxels; each subvoxel will effectively be instantiated by a block.

The blocks are labeled by layer and segment. Layer and segment labels allow the assignment of joint types that must connect adjacent blocks. Once the joint types have been assigned, blocks providing these joint types can be selected. The output is a sequence of block assembly orders that constructs an interlocking structure shaped as the input model. Since our model is built layer-by-layer, the final structure will have k keys, where k is the number of layers that do not have another layer on their top.

One critical observation is that joint types for a pair of blocks are selected by the layout algorithm based on the location of those blocks in the segment and layer. Since there are three joint types, each male or female, and six faces on a cube, this suggests that there might be $6^6 = 46\,656$ different types of blocks to construct. Fortunately, patterns in the segments and layers mean that not all of these block types occur. Further tricks allow the reduction of the number of block types to two. As an example, consider the blocks in Fig. 5(a). Adding a mortise joint on the right side of block

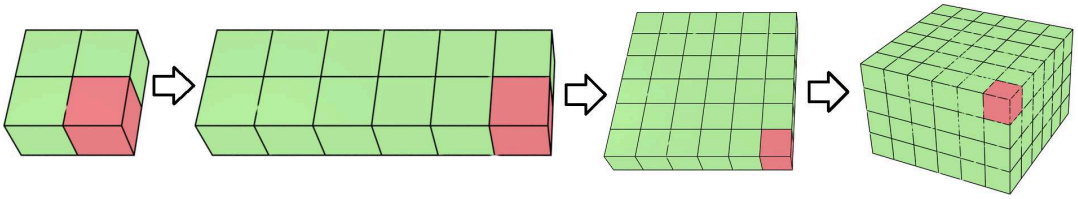


Fig. 6. Building an interlocking 3-D structure. Blocks interlock to form a square, squares form an interlocking segment, segments form a layer, and layers interlock to form the structure. The keys are marked red.

K_B makes it a copy of K_A, reducing the number of types of blocks.

It is also worth pointing out the approach we have taken to connecting adjacent layers. To provide a firm connection, we use a block of height two as a connector between layers; we call this block a post.

Algorithm 1 Algorithm Overview

```
function ConstructVoxelModel(M)
    M' ← split every voxel into eight dimension-1 cubes.
    for each layer L_i of M' from bottom to top do
        Lay out any missing posts.
        if L_i is an even layer then
            Set all segment types to X_{l-}Y_{+}. (Sec. V)
        else
            Decide the key to each component. (Sec.VI-A)
            Order segs in each component. (Sec.VI-B)
            Decide the key(s) to each segment. (Sec.VI-B)
            Decide the type of each segment. (Sec.VI-B)
            Find special cases. (Sec.VI-C)
            Modify L_i and L_{i+1} if necessary. (Sec.VI-C)
        end if
        Assemble blocks (in parallel). (Sec.VII-A)
    end for
end function
```

IV. Blocks and Squares

In this section, we will introduce the smallest interlocking structures, a square, and the two kinds of blocks that make the structure possible. Squares serve as the fundamental element of our construction. Larger structures assembly will be later introduced by connecting squares.

The layout algorithms make use of two types of blocks: a cube is a unit-cube-sized block for filling empty space in a layer and locking with existing blocks, and a post is a two-unit high block (see Fig. 7). The lower half of a postblock connects with cube blocks in the same layer, whereas the upper half of the block connects with cube blocks in the upper layer. The postblocks also act as key blocks of substructures.

We carefully analyzed segments and layers to determine how joints might be arranged on cubes and posts. A cube block has two dovetail male joints on two opposite sides that can connect, in a tangential direction, with female joints in postblocks allowing motion only in the assembly direction.

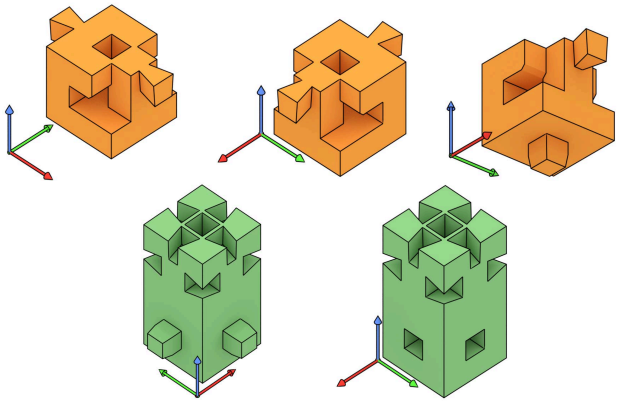


Fig. 7. Different views of cube and postblocks.

Fig. 8(a) and (b) shows how a cube's side male joints connect with a post's female joints in two different directions. A cube also has a male joint on the bottom that connects, in the normal direction, either with the top of a cube or post. The female joints on the opposite sides of the cube block allow postblocks' male joint to drop and slide to connect, which allows the postblock to disassemble only in two directions.

To describe a layout and an assembly process, some notation is helpful. For each block, we use a triplet of characters indicating the block type, orientation of the block, and assembly direction. Fig. 9 shows all of the triplets used in the assembly of structures in this article. For example, C1D means "**C**ube in orientation 1, assembled by moving **d**own." Fig. 9(b) shows all of the notation triplets used in the current approach. Not all axis-aligned orientations of cubes and posts are needed to construct structures; for example, posts only occur in the four orientations generated by rotating the post in Fig. 7 around the z-axis in 90° increments.

Block designs are crafted to allow the design of squares, segments, and layers. A square is the smallest interlocking structure we consider, composed of four blocks: two posts and two cubes. By using posts and cubes in different orientations, different squares may be constructed, as shown in Fig. 8(e) and (f) as S_a and S_b. Different squares will be used in Section VI to constrain key block motions of other adjacent segments in the same layer, allowing interlock of the layer.

Fig. 8 shows the process of assembling one kind of square. The first piece, a post, may connect to a layer below the current one. Two cubes are added to the top of the post. The second post acts as a key, and the top half of this post extends above the square to provide a connection to a square that may be later built above the current one.

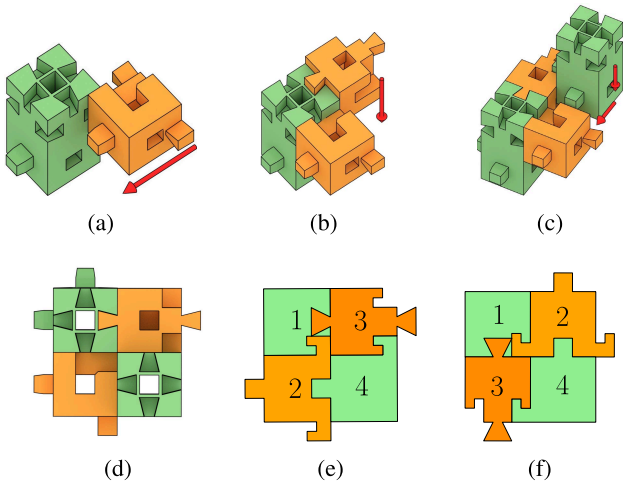


Fig. 8. Assembly process of a square and two designs for a square. (a) Slide in a C8W block. (b) Drop in a C3D block. (c) Assemble a P1W block. (d) Top view of the square. (e) S_a square. (f) S_b square.

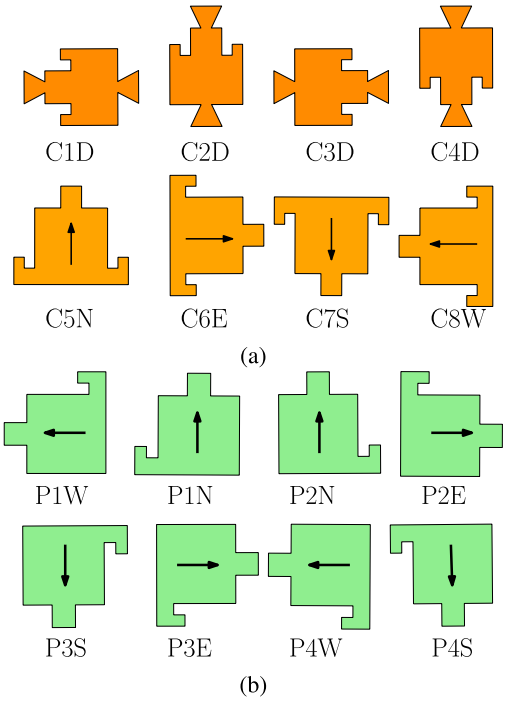


Fig. 9. Different ways to assemble cube and postblocks and corresponding notations. (a) Cube orientations and assembly directions. (b) Post orientations and assembly directions.

V. Segments

We now introduce a method to link squares into a longer interlocking structure, a segment. A segment is composed of n squares in a $1 \times n$ pattern. To build a segment, we assume that n posts have already been preplaced in the prior layer such that the top of each post appears in the same position in each square; these posts allow the segment to interlock with the prior layer.

We will discuss how to assemble a simple segment built from left to right in the direction of the x-axis, assuming that posts are in the upper (or $y+$) half of each line; other segments are symmetric and will not be discussed in detail. We denote a segment as $Y_{l+}X_+$ if the posts are in the left position of the y positive half and the segment is built toward the x positive direction with the key block at the end.

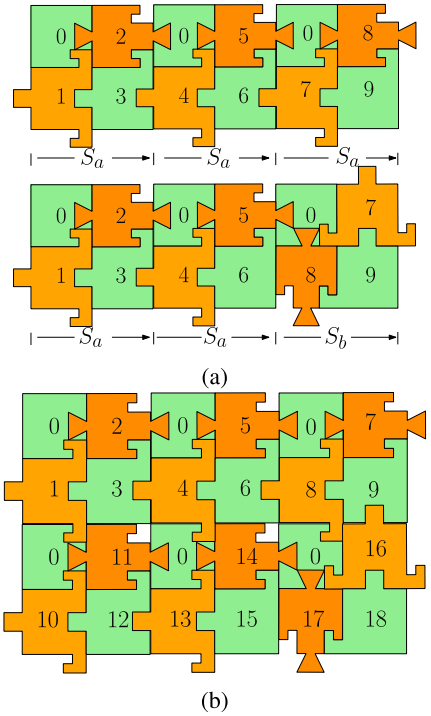


Fig. 10. Simple segment and an example layer built by connecting two segments. (a) Two $Y_{l+}X_+$ segments. (b) Layer built by two $Y_{l+}X_+$ segments.

Fig. 10(a) shows the process of assembling a $Y_{l+}X_+$ segment of three squares. We connect, from left to right, $n-1$ S_a squares. The final square of a subsegment can be of type S_a or S_b. The key piece of the segment is the last assembled postblock. A subsegment with an S_b final square is not interlocking, but when connected with previous segment(s), the S_b square prevents the adjacent block in the y positive direction from moving and interlocks the structure. Building another $Y_{l+}X_+$ segment on the y negative side will create an interlocking layer [see Fig. 10(b)]. In Section VI, we will discuss how to constrain the motion of the key in different types of segments.

A. Structure Mirrors

Knowing how to assemble $Y_{l+}X_+$ segments, one can lay out an array of segments one-by-one and create interlocking planar structures as in Fig. 10(b). However, these structures require the key to every segment to be in the x positive end and constrained by the next adjacent segment. To build more complicated planar structures, we introduce the concept of mirrors.

Definition 1 (x-Mirror): Object A is an x-mirror, $m_x(B)$, of another object B if one is a reflection of the other with reflection plane perpendicular to the x-axis.

We define an analogous y-mirror operation. Cube and post designs are symmetric in such a way that x and y mirror operations can be accomplished by simple rotation of the block. Construction of a mirrored structure follows the same order of the original structure with opposite directions along

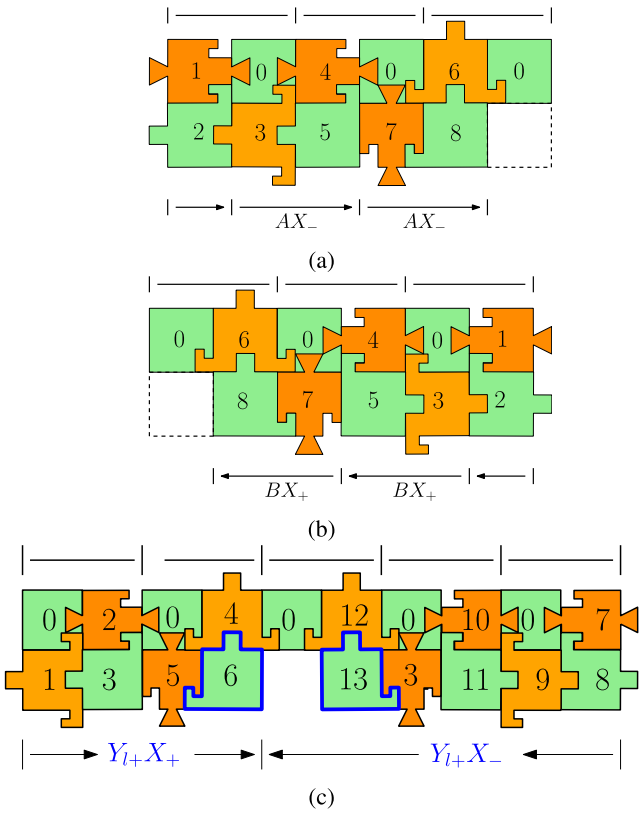


Fig. 11. Construction of two x-mirrored segments. Arrows indicate construction direction. Numbers indicate assembly order. (a) $Y_{r+}X_+$ segment. (b) $Y_{l+}X_-$ segment. (c) Pair of segments with keys in the middle.

the same axis; for example, we may build a $Y_{r+}X_-$ segment by x-mirroring a $Y_{l+}X_+$ segment.

Two other types of segments we will need for layer construction are $Y_{r+}X_+$ [see Fig. 11(a)] and its x-mirror $Y_{l+}X_-$ [see Fig. 11(b)]. To build a $Y_{r+}X_+$ segment with n squares, where $n \geq 2$, we first assemble two blocks ($C3D$ and $P1W$) in the left two positions. Then, assemble a $Y_{l+}X_+$ segment of $n-1$ squares. When all preexisting posts are prevented from moving along the z-axis, the segment is interlocked.

For many input geometries, it may turn out that neither end of a segment is adjacent to the next segment, causing the key to be exposed. In this case, we may replace a single segment with two segments grown from the ends, effectively allowing placement of a pair of keys at an arbitrary position in the middle, as shown in Fig. 11(c). These keys may then be immobilized by later segments.

VI. Layers

Now that we know how to build different kinds of segments, we can connect a set of segments on the same plane to create complicated interlocking 3-D structures, by careful assignment of subvoxels from the original model into layers, segments, squares, and blocks.

A layer is a set of squares with the same z-coordinate. A set of connected squares with the same z-coordinate is a layer component. We assume that all layer posts are provided by the prior layer. This is a fundamental limitation of our approach—it does not allow overhanging structures to be generated without building additional supports.

This section first introduces the ordering of segments in a component. Once ordered, segments are ready to be assigned square types and assembled. Then, we discuss some special cases caused by the nature of our block design and square structure, and techniques to ensure interlock.

A. Layer key(s)

As the first step of building any interlocking structure, we determine the key(s) of the layer. A layer is immobilized if the key(s) is fixed with respect to its neighbors. Since every even layer has an upper layer with the exact same shape, based on the division of voxels into subvoxels, postblocks that connect the upper layer will be immobilized as long as the upper layer is interlocked, preventing the horizontal motion of any posts. Therefore, we only consider the odd layers in this section.

For any odd layer component without adjacent upper layer blocks, we select a postblock at the x negative end of a boundary segment as the key, where a boundary segment is a segment with adjacent neighbors on only one side. If the odd layer component has an adjacent upper layer, the key can be any postblock covered by an upper layer square.

Under this rule, every layer component constrains the key to its lower component. Any layer components that do not have an immediate upper layer introduce a new key that will not be covered. The number of key pieces of the whole structure is thus the number of layer components without an immediate upper layer. This introduces an interesting effect of the orientation of the object to be constructed. For example, the chair in Fig. 1(a) has a single key, but if the chair were built upside-down, then there would be four keys: one in each leg.

B. Segment Construction Order

Once a layer's key square and all starting posts of squares are known, the second step of assembling a layer is to determine the order and type of each segment.

In the preprocessing step, every voxel is broken into two squares, making every layer of voxels two layers in the assembly. The bottom layer has an even z-coordinate value, whereas the upper layer has an odd z-coordinate. Every segment in an even layer is constructed along the y-axis directions. We simply assemble every segment as $X_{l-}Y_+$ or 90° clockwise rotation of a $Y_{r+}X_-$ segment, from left to right. An even layer component is not necessarily interlocked because there can be many segment keys unconstrained and able to move in the x positive direction. However, all square keys are posts in the upper layer, and as long as the upper layer is interlocked or all posts are prevented from moving in the x positive direction, the two-layer structure is interlocked.

Each square in an odd layer component is initially assumed to have a post in the bottom-right position. This, however, could change after the segment types have been assigned. We first build a set of post lists where each list contains posts with the same y-coordinate, and two adjacent posts are two

units away. Each list will be built into a segment. Two posts are considered adjacent if their x- or y-coordinates have a difference of 2. Two lists are considered adjacent they have adjacent posts. Lists are ordered by their shortest distances to the final list that contains the post of the key square, where the distance between two adjacent lists is 1.

Given a list l and the next-built adjacent list l_n, the type of the segment S_l associated with l is determined as described next.

1) If l_n is at $y-$ side of l and the left end post of l is adjacent to l_n, S_l is $Y_{r+}X_-$.
2) If l_n is at $y-$ side of l and the right end post of l is adjacent to l_n, S_l is $Y_{r+}X_+$.
3) If l_n is at $y-$ side of l and neither ends of l is adjacent to l_n, S_l is broken into a $Y_{r+}X_-$ and a $Y_{r+}X_+$ segment.
4) If l_n is at y positive side of l and the left end post of l is adjacent to l_n, S_l is $Y_{r-}X_-$.
5) If l_n is at y positive side of l and the right end post of l is adjacent to l_n, S_l is $Y_{r-}X_+$.
6) If l_n is at y positive side of l and neither ends of l is adjacent to l_n, S_l is broken into a $Y_{r-}X_-$ and a $Y_{r-}X_+$ segment.

The segment associated with the last built list has been specified a key (line 8 of Algorithm 1). Its type is thus determined.

C. Special Cases

At this point, the type of each segment and the order of construction in each layer have been selected. Many interlocking layer structures can be assembled by directly following the construction of each segment, as specified in Section V. However, depending on the successor segments, some small modifications might be applied to insure the interlocking of adjacent segments.

Consider a segment with key(s) in the y negative side, for example, $Y_{l+}X_+$. Its successor can be: 1) a segment whose key will be constrained by further segments in the y negative side; 2) a segment with the key being constrained in y positive side; or 3) a segment whose key will be constrained by the upper layer. We now list all possible cases that need modifications.

In Case (1), A $Y_{l+}X_+$ segment followed by another $Y_{l+}X_+$ segment. We use an S_b square at the later built segment to prevent the segment's key from moving [see Fig. 10(b)]. Otherwise, a $Y_{l+}X_+$ segment always uses an S_a end square.

Case (2) contains four subcases where the current segment has one or both ends adjacent to its successor whose key is in the x positive or negative side. Fig. 12(a) shows one subcase. The first $Y_{l+}X_+$ segment is still assembled as usual. We leave some positions adjacent to the first segment unfilled and assembled the rest part. Fig. 12(b) is a similar subcase where both ends of the segment are adjacent to the successor. We divide the lower segment into two segments: one containing no posts adjacent to the upper segment will be built first, and the other containing the rest posts will be built after the upper segment. In the other subcases, the successor has a key in the x negative direction, and we change the upper segment to $Y_{l+}X-$ and create an x-mirror of the previous case.

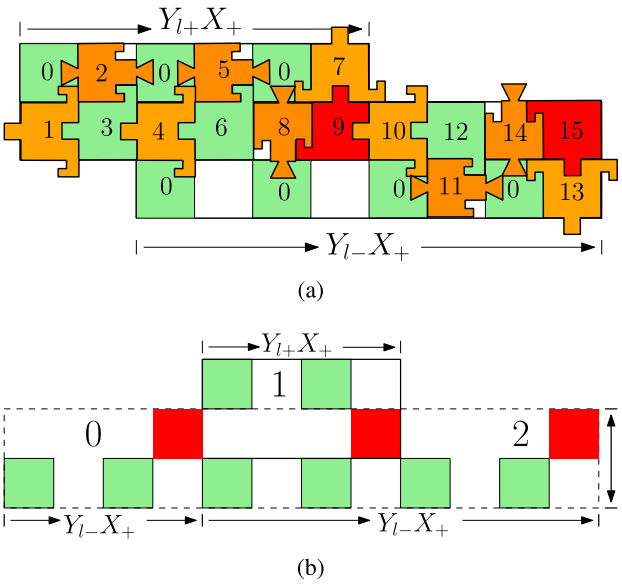


Fig. 12. Two special cases of building adjacent segments. Green blocks are posts, and red blocks are keys of each segment. Numbers indicate the assembly order. (a) $Y_{l-}X_+$ segment built after a $Y_{l+}X_+$ segment. Some positions are left empty. (b) Longer lower segment. The lower segment is broken into two segments.

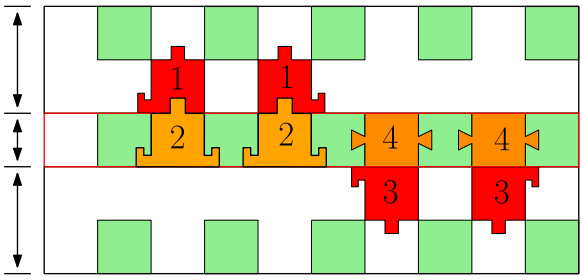


Fig. 13. Special cases where two segments with posts in different sides are finished before the segment in the middle. Red blocks are keys of two segments ($Y_{r+}X$ and $Y_{r-}X$ types). Numbers indicates the assembly order.

Case (3) is shown in Fig. 13 where a $Y_{r+}X$ segment and a $Y_{r-}X$ segment are assembled before the segment in the middle. We require the upper and lower segments' keys to be in different x positions. To ensure interlocking, we first finish the upper segment and then assemble two C5N blocks in the middle segment. After the lower segment is assembled, we put in C3D block(s) in the middle to constrain the motion of the lower segment key(s). The last assembled blocks (keys) in the middle will be constrained by its upper layer. If the upper layer is not wide enough to cover the keys, we must expand the upper layer (Line 13 in Algorithm 1).

VII. Automatic Assembly and Parallel Construction

Section III-B introduced the high-level layout algorithm, which assembles an interlocking structure layer-by-layer from bottom to top. Each layer will be an interlocking substructure that also connects with its lower layer using the posts and constrains the key(s) of the lower layer. Now that we have a better understanding of how the substructures are labeled and constructed, we can transform the high-level algorithm to a more detailed level and execute the assembly accordingly.

TABLE I
PREDECESSORS OF EACH TYPE OF BLOCK

Block type	Predecessors	Block type	Predecessors
C1D, C3D	$(x-1, y)$, $(x+1, y)$	C8W	$(x, y+1)$, $(x, y-1)$, $(x-1, y)$
C2D, C4D	$(x, y-1)$, $(x, y+1)$	P1W, P1N	$(x-1, y)$, $(x, y+1)$
C5N	$(x-1, y)$, $(x+1, y)$, $(x, y+1)$	P2N, P2E	$(x, y+1)$, $(x+1, y)$
C6E	$(x, y+1)$, $(x, y-1)$, $(x+1, y)$	P3S, P3E	$(x+1, y)$, $(x, y-1)$
C7S	$(x-1, y)$, $(x+1, y)$, $(x, y-1)$	P4W, P4S	$(x-1, y-1)$, (x, y)

Our construction starts from the bottom layer to the top. For each layer, we first check whether all required posts exist. If not, we lay out these posts before starting the assembly (Line 4). Even layers are constructed using $X_{l-}Y_{+}$ segments (Lines 5 and 6). Odd layers need to find the keys first (Line 8). Based on the key to each layer component, we order segments (Line 9) and then determine segment keys and segment types (Lines 10 and 11). Before assembling, we check whether any special cases exist, as mentioned in Section VI-C (Line 12). Since $Y_{r+}X$ and $Y_{r-}X$ segments require at least two adjacent squares, we need to modify the current layer if the condition is not satisfied. The special case as in Fig. 13 can also require the upper layer to expand and cover lower layer keys (Line 13). We then finally assemble blocks based on block types and special cases. Fig. 1 and Fig. 14 are two structure assembled using this algorithm in simulation.

A. Parallel Construction

Laying out blocks one-by-one is time-consuming when a structure has a large number of blocks. This section provides an algorithm that generates a parallel construction order to accelerate the process. We first consider preliminary blocks of assembling each new block and build a graph between blocks. By querying the graph for blocks whose preliminaries are satisfied, we can have multiple agents to lay out the blocks.

Consider a block b to be assembled in a layer. Any adjacent block(s) to be assembled later should not be prevented by the male joint(s) of b, meaning that the joints of a block connect to only the preexisting blocks. Along the assembly direction of b, the male joints of b should not be able to touch any blocks. The blocks that must be assembled before a new block to prevent collision are called predecessors of the new block. Every block has a predecessor below it if an adjacent block exists in the lower layer. Consider a block at position (x, y) in any layer. Table I shows a list of predecessors of different types of blocks in the same layer.

Besides predecessors listed above, inside each square, cube blocks with mortise joints connecting blocks in the same layer (C5N, C6E, C7S, or C8W blocks) must be assembled before others (C1D, C2D, C3D, or C4D blocks).

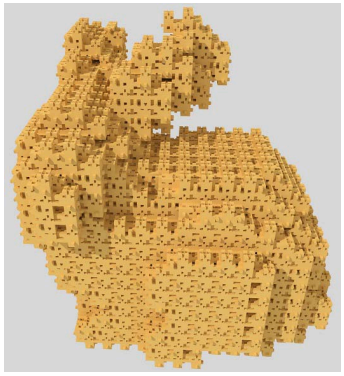


Fig. 14. Stanford Bunny assembled in simulation.

With the predecessors of each block, we then construct a directed graph $G = \{V, E\}$, where V is the set of blocks, and directed edge $e_{i,j} \in E$ indicates block i being a predecessor of block j. The construction follows the order of removing nodes with an in-degree of 0. Each construction agent/thread will take a block whose predecessors have been placed and remove the node from the graph when the block assembly is finished.

A simple observation with the parallel construction is, after the construction of one square s, all the adjacent squares to be assembled after s in the sequential order is ready to assemble. We, therefore, have the following theorem.

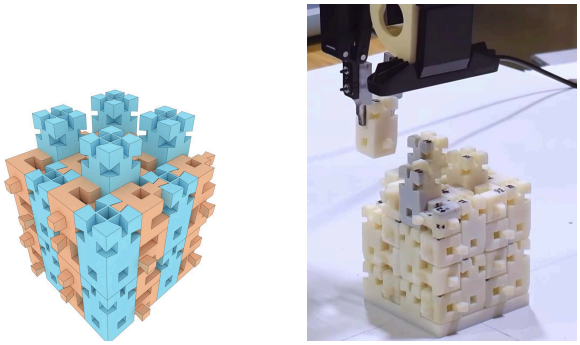
Theorem 1: Parallel construction of a solid cube of N squares takes $O(\sqrt[3]{N})$ time.

Proof: First, consider constructing a solid layer of $n \times n$ squares. For simplicity, we scale the width of each square to one. After assembling the square at the corner $(0, 0)$, two adjacent squares in the x and y positive directions will be assembled at the next time step, then three, four, and so on. It takes k steps to construct $k(k+1)/2$ squares. When $k = n$, over $n^2/2$ squares are constructed, thus constructing a layer takes at most $2n$ steps. In a cube, since finishing every square allows all adjacent squares in the x, y, and z positive directions to assemble. When the last square of the bottom layer is done, it takes one more step to finish the upper layer. Thus, $2n-1$ more steps will finish all upper layers. Therefore, a solid cube of $2n \times n \times n$ squares takes $O(n) = O(\sqrt[3]{N})$ time to assemble. ■

VIII. RESULTS

We algorithmically designed plans to assemble several models, including a Stanford bunny and a chair model and animated the results in software. Figs. 1 and 14 show these examples. The Stanford bunny model has 7337 blocks, while the chair model has 472 blocks. The assemblies of both models are done in sequential order. The rendered animation of chair assembly can be found in Video [44].

We also have 3-D printed 406 blocks and assembled them into a similar chair based on the rendered animation. The assembled chair is a simplified version of the chair in the simulation; two layers were omitted to save material and assembly time. Four legs of the chair are relatively loose compare to other parts because each pair of layers in the legs are connected by only one post and a mortise and tenon joint.



(a) (b)

Fig. 15. 48-block cube-like interlocking structure is assembled using a two-arm robot system. Blue blocks are posts and orange blocks are cubes. (a) Rendered structure in simulation. (b) Assembly in the real world.

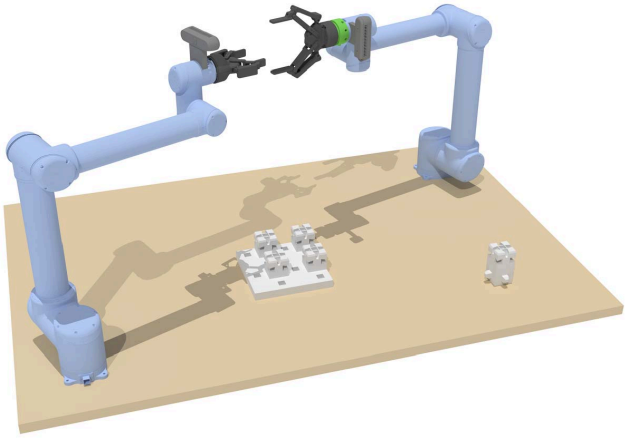


Fig. 16. Our experiment environment setup. Robot arms are blue, the force–torque sensor is green, the depth cameras are in dark gray, and the grippers are black. The blocks and structure base are the white part.

Some very simple structures were assembled using a 4-DoF robot arm as an early-stage exploration. These can be found in Video [45] (one layer) and [46] (two layers).

IX. Robotic Assembly Experiment

To demonstrate the feasibility of the assembly approach, we developed a system to allow two 6-DoF robot arms to perform a simple example of interlocking structure assembly using our blocks. With this system, we assembled a four-layer cube-like structure of 48 blocks. The structure does not have overhangs and can be constructed without support material. Due to the lack of more robots and the limitation of the workspace dimensions, our construction was not done in a parallel manner, as mentioned in Section VII-A.

To correctly assemble a block into its target position, the system must solve the following problems: 1) recognize the position and orientation of the construction base; 2) distinguish between cube and postblocks; 3) pick up blocks that initially sit in a specified area; 4) precisely estimate the position and orientation of the held block precisely; 5) reorient the block if it cannot be picked up in the desired orientation; and 6) follow a path to assemble the block. In this section, we will introduce the robot system, how each problem mentioned above is solved, and experimental results.

A. Experiment Setup and Assembly Process

The environment setup is rendered in Fig. 16. The system includes the following hardware.

1) *One Flat Table:* The table was used as the platform for assembly. A construction base is attached to the center of the table. The base is our origin. Blocks are initially placed randomly close to the bottom-right corner of the table.
2) *Two 6-DoF Robot Arms:* We used two Universal Robots' UR-10 robots, each with six degrees of freedom. The arms cooperate to regrasp the block sequentially to reorient blocks and remove configuration error. Details of regrasping are presented in Section IX-C.
3) *Two Depth Cameras:* One for each robot arm. The arm not currently holding the block estimates block pose using a Create Labs' BlasterX Senz3D camera attached near the end-effector. The cameras integrate Intel's RealSense technology for 3-D point cloud reconstruction based on structured light. Depth images are more stable than RGB images, which can be influenced by small light condition changes.
4) *Two Two-Finger Grippers:* The arms are equipped with Robotiq 2F-85 and 2F-140 model grippers. We designed and 3-D printed finger tips for grasping small areas on the blocks.
5) *One Force–Torque Sensor:* One Robotiq FT-300 force–torque sensor was installed on one robot arm that does all block insertion. The force/torque sensor gives some information about the contacts one block might experience when inserted. We utilize this information to prevent jamming caused by friction.

Robot arms are mounted on two diagonal corners of the table to maximize space utilization. The right robot, R_A, is responsible for picking up blocks and placing them into desired locations and is equipped with the force–torque sensor. The other arm, R_B, is used during the regrasping and pose estimation process.

We placed a base of four posts at the center of the table as the starting point for structure assembly. Waiting blocks are typically placed at the bottom-right corner of the table, so R_A can move to check that area and pick up a block if presented.

Our assembly process is described in Algorithm 2. The input is a sequence of block types and a desired final configuration. We call a tuple of block type and configurations a command. Commands are generated based on the assembly rules described in previous sections. For each command, the right-side robot R_A will first pick up a block based on the input type. The block is then moved above the center of the table, so the camera mounted on R_B can see the block and estimate its current configuration. If the block is not currently grasped in the desired orientation, we must find a sequence of regrasping actions and execute them to reorient the block correctly. Finally, we place the block into the desired location.

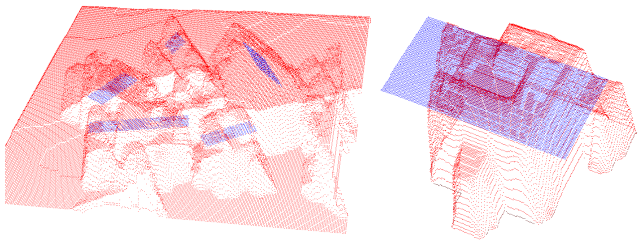
Algorithm 2 High-Level Robotic Assembly Process

function ASSEMBLE($C \leftarrow [(b_1, c_1), \ldots, (b_n, c_n)]$)
  for Block b_i and orientaiont o_i in C **do**
    Move R_A to bottom-right corner of the table.
    Pick up a block (same type as b_i). (Sec IX-B)
    Move R_A and R_B above the table center so two arms are facing each other.
    $o_e \leftarrow R_B.EstimateBlockOrient()$. (Sec.IX-B)
    if $o_e \neq c_i.orientaion$ **then**
      $Acts \leftarrow$ generate re-grasping actions for each robot arm. (Sec.IX-C Algorithm 3)
      $ExecuteRegrasp(Acts, R_A, R_B)$. (Sec.IX-C)
    end if
    Plan path for block assembly.
    R_A executes the path and transform the block into a desired configuration.
  end for
end function

B. Block Bin Picking and Pose Estimation

Blocks are placed in a pile on the table for the robot to pick up. For successful placement of a block in the assembly, the robot must first locate the desired block in the pile, grasp and remove it from the pile, determine the pose of the grasped block, regrasp the block so it is held by the fingers of the gripper in a manner where it can be placed in the assembly without the fingers getting in the way, and then finally placed in the assembly at the desired location. The fingers are equipped with rubber pads to provide a solid grasp of the block; however, during the initial contact phase of regrasping, the block can shift necessitating a precise pose estimation of the block at each step to ensure successful regrasp operations and placing the block in its final pose in the assembly.

For bin picking, we use a neural network to translate a height map of the pile of blocks to labeled grasp locations for the gripper fingers, where the label is the block type and face (designated as a block/face id) associated with the grasp location. Training data for the neural net are generated by a simulator using Bullet (reference) to simulate different piles of blocks. For each block in the pile, we test for a collision-free placement of the fingers to grasp the block, using candidate grasping areas preauthored for each block. Valid grasp regions are represented as a rectangle oriented in 3-D in a heightmap, where the extent of the rectangle represents the finger opening for the grasp and collision-free range for the fingers perpendicular to the opening. Note that we assume a two-finger parallel gripper setup. Fig. 17(a) shows an example of the generated heightmap of a pile of blocks with the associated labeled grasp regions (in blue) superimposed in the same heightmap. The grasp location heightmap is stored in an RGBA image format, using the alpha channel for height, and the red and blue color channels for encoding the discretized yaw of the fingers and the associated block/face id for the grasp. For each pile of blocks, we simulate the heightmap generated by the depth camera by rendering the pile with the same field of view as the SR-300 camera and then extracting the depth map and reprojecting it into a heightmap. To facilitate the transfer of the learned model for use with real hardware, we also add Perlin noise to the heightmap for domain randomization. The simulated heightmap of the blocks is stored in an RGBA image, using the alpha channel for the height information. We generate roughly 800 000 pairs of various block piles and their associated grasp heightmap as 128×128 pixel images.



(a) (b)

Fig. 17. Generated bin-picking and pose estimation data for training the neural network. The height map data are downsampled from a raw image, including Perlin noise to simulate the real sensor output. (a) Height map of a pile of blocks generated virtually. The blue rectangles represent the faces the gripper can approach to pick up blocks. (b) Generated height map of a block. Blue rectangle represents the grasping region. The image is for pose estimation.

The neural network architecture we use to learn the mapping is a ResNet-Unet configuration. The encoder layers are arranged as ResNet-101 with a mirroring of the structure in the decoder. To make this fully convolutional, we replace the max-pooling layers with convolution layers. Skip connections are added between the corresponding blocks of the ResNet structure. We use a multiclass cross-entropy loss per pixel. We train for three epochs.

To extract a specified block from the pile, the robot moves its attached camera over the designated area where the pile of blocks is located, and a heightmap is generated and sent to the model as an RGBA image. The output grasp heightmap from the network is processed using OpenCV to isolate the grasp regions. For each grasp, we count the block/face id in each pixel of the region and use the highest count to represent the block/face id associated with the grasp. Likewise, we extract the finger yaw by using the highest count yaw value in the region. The roll and pitch of the fingers are extracted from the oriented rectangle represented by the region in the grasp heightmap. The location of the midpoint between the fingers is the centroid of the oriented rectangle. The finger opening is the corresponding rectangle width. Of the available inferred grasps, we choose the one with the desired block type and using the embedded grasp information command the robot to approach, grasp, and pick up the specified block.

For regrasping and placement of the block, we assume that we have a rough estimate of the pose of the opposite face of the grasped block and can move the camera attached to the nongrasping robot to view the opposite face in a safe manner. Indeed, we know the face associated with the grasp of the block from the pile, and given a rough estimate of where it is grasped, we can infer a rough estimate of the pose of the opposite face.

We leverage the same ResNet-Unet configuration we used for bin picking to get pose estimates of the grasped blocks. Given the reliability of the architecture to translate a heightmap of the scene to a representative labeled heightmap, we cast the pose estimation problem to a translation problem of taking a heightmap of a block face to a reference rectangle fixed to that face. The reference rectangle normal is parallel to the face normal and is placed over the centroid of the block at the height of the topmost part of the face in the direction of its normal. By training a neural network to infer the heightmap of the reference rectangle fixed to the face of the input heightmap, we can use PCA to obtain the pose of the rectangle and then map that back to get the pose of the block. Training data for the neural net are generated by a simulator. For each face of the block, a heightmap is generated (with some noise) looking toward that face at various offsets of its centroid from the view center. The height map of the reference rectangle fixed to that face is associated with each height map. In addition, the label of the block/face id is embedded in the heightmap using the red color channel. Since we assume that the camera can be placed roughly facing the opposite face of the grasped block, the offsets for the generated data are within a modest ± 3 cm and ± 0.3 radians of the centered view. Fig. 17 shows an example of the generated heightmap of a face with the associated labeled reference rectangle (in blue) superimposed in the same heightmap. We generate roughly 800 000 pairs of 128×128 heightmap images and train for three epochs.

To get an accurate pose estimate, the camera attached to the nongrasping robot is moved to roughly view the opposite face of the grasped block. A heightmap is generated of this face by taking the depth map from the camera, clipping at 20 cm from the camera to remove background noise, and then reprojected into a heightmap. The heightmap is sent to the ResNet-Unet model and the inferred pose of the block (obtained from the PCA of the reference rectangle) along with its block/face id is extracted from the output heightmap. If the inferred face id matches the expected value, the camera is moved to center the face in the view. This process is repeated several times until the adjustment is smaller than an empirically defined threshold. At this point, we have an accurate estimate that the block is centered in the view and can generate an accurate pose estimate of the block to drive the regrasping and placement actions. Ideally, a single query of the model from a heightmap of the face would be enough to get an accurate pose estimate; however, we found large offsets from the center yield coarse inferred pose estimates of the reference rectangle and hence the need for this visual servoing strategy.

C. Regrasping

When robot R_A picks up a block, there is no guarantee that the block is held at the desired location. However, the gripper does not allow rotation of a block while holding that block. To solve this problem, we used the second robot R_B to temporarily hold the block, so R_A can regrasp the block in a different location (see Fig. 18). We may have to iterate this process many times to achieve the goal orientation.

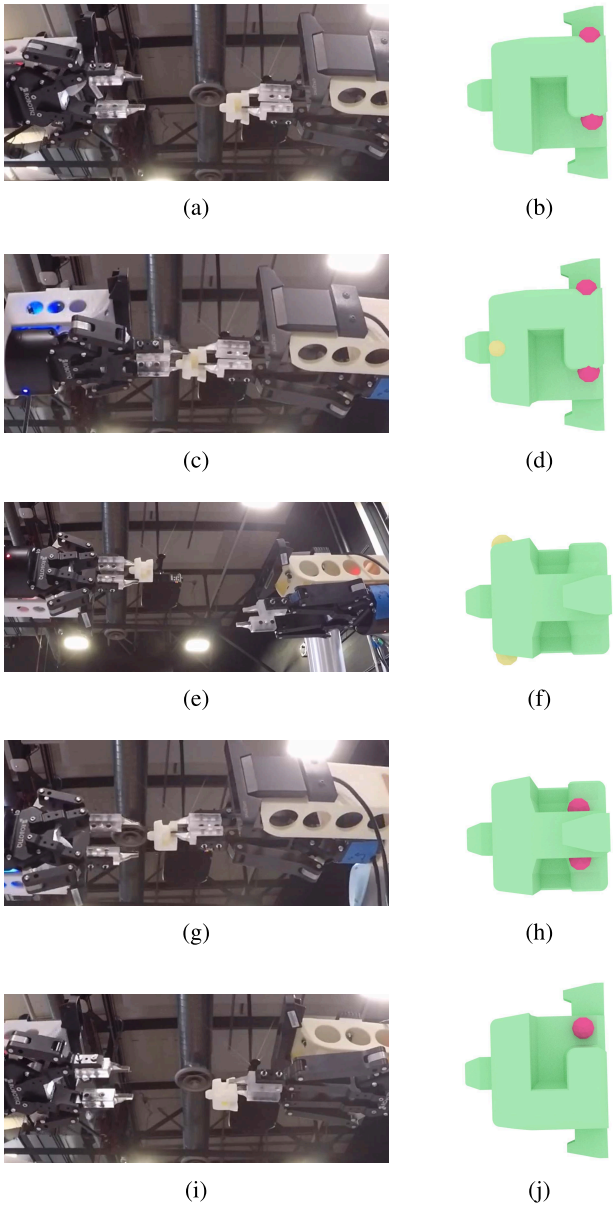


Fig. 18. Regrasping process that allows the right robot R_A to hold the block in different locations. The left robot is named R_B. Red dots represent grasping locations of R_A, and yellow dots are grasping points by R_B. (a) Right robot R_B estimating the block pose. (b) Block held by R_A at red points. (c) R_A transferring the block to R_B. R_A will release. (d) Block grasped by two grippers. (e) R_B rotated. R_A estimating the block pose. (f) Block pose when rotated. (g) R_B transferred the block back to R_A and then released. (h) Block held by R_A and R_B released. (i) Robot R_A rotating back. The block is grasped at the desired location. (j) Block grasped by R_A in the right location.

Grasping locations for each of the six faces of a block are specified manually. Initially, we move robots to home configurations C_{ra} and C_{rb}, respectively, where the grippers are 40 cm above the center of the table facing each other, and the cameras are up. The configuration C_{rb} is shown as the left robot configuration in Fig. 16. Every time a regrasping action is executed, the robot holding the block is moved back to the home configuration. Our goal is to have the robot R_A hold the block in the desired orientation, while the robot is in the home configuration.

To make the process as fast as possible, we constructed this problem as a tree search and then used breadth-first search to find the shortest sequence of regrasping actions. The algorithm is shown in Algorithm 3. In this algorithm, we assume that the block is initially held by R_A and the initial and goal block orientations are in the frame of the gripper attached to R_A. We first check whether the block is held in the desired orientation; if not, a set of valid grasping locations are generated for the other robot. The other robot will imaging regrasping the block according to each valid grasping location and then transform to its home configuration. If a valid regrasp will hold the block in the desired orientation, the algorithm will stop; otherwise, children valid regrasps will be further explored. This process is continued until R_A is holding the block correctly.

There are several configurations a robot cannot do making some grasping locations invalid. For example, our environment requires that no gripper is facing up, as the gripper may collide with the assembling structure in these configurations. Also, due to the nature of how we mount the cameras, we do not want the grippers to be perpendicular to each other with cameras on the same side because cameras may collide in these configurations.

Algorithm 3 generates a sequence of regrasping actions since it is planned virtually. We then have the robots to execute the actions. The execution process is straightforward but includes an extra part of block pose estimation before grasping the block. This is because the block is not strictly static with respect to the gripper when and after the previous grasping. Thus, a pose estimation will help to reduce the error.

Algorithm 3 Regrasping Actions Generation

function PLANREGRASP(init orient o_s, goal orient o_g)
  Move R_A to home configuration C_{ra}.
  Move R_B to home configuration C_{rb}.
  $orients \leftarrow$ a stack of block orientations and corresponding holding robot
  $orients.push((c_s, R_A))$.
  while $orients$ is not empty **do**
    $o, r \leftarrow orients.pop()$
    if $r = R_A$ and $o = o_g$ **then**
      return $TraceBack(o, r)$
    end if
    if $r = R_A$ **then**
      valid_grasps $\leftarrow$ valid grasps for R_B
      for $grasp \in valid\_grasps$ **do**
        $o_{next} \leftarrow$Transform block according to $grasp$, assuming R_B will be moved to C_{rb}.
        $orients.push((o_{next}, R_B))$.
      end for
    else
      valid_grasps $\leftarrow$ valid grasps for R_A
      for $grasp \in valid\_grasps$ **do**
        $o_{next} \leftarrow$Transform block according to $grasp$, assuming R_A will be moved to C_{ra}.
        $orients.push((o_{next}, R_A))$.
      end for
    end if
  end while
end function

D. Experiment Results

Using the robot system, we assembled a cube-like interlocking structure of 48 blocks. The rendered structure is shown in Fig. 15(a) where blue blocks are posts and orange blocks are cubes. This experiment includes all the operations described above. Fig. 19 shows a screenshot of the robots executing assembly commands.

A regrasping operation can be found at Video [47]. In this video, four regrasping actions were performed (excluding the initial pickup) to reorient a cube held by the right robot R_A. This operation took over 5 min for one block.

A complete video of assembling the cube structure can be found in Video [48]. To prevent making the experiment video too long, we avoided some regrasping by hand-feeding the robot blocks in correct grasping locations. In total, the experiment took less than 2 h. Because of the limitation of my camera (30-min max video recording), the assembly of each layer is executed and recorded separately in two days.

E. Challenges

We faced many challenges during the experiment. In this experiment, we made two very strong assumptions. First, we assumed that a block can be moved perfectly precisely such that there is no contact during insertion. Second, we assume that block joints are fabricated precisely so that the gap between a pair is quite small. These assumptions are unfortunately not satisfied in the real world.

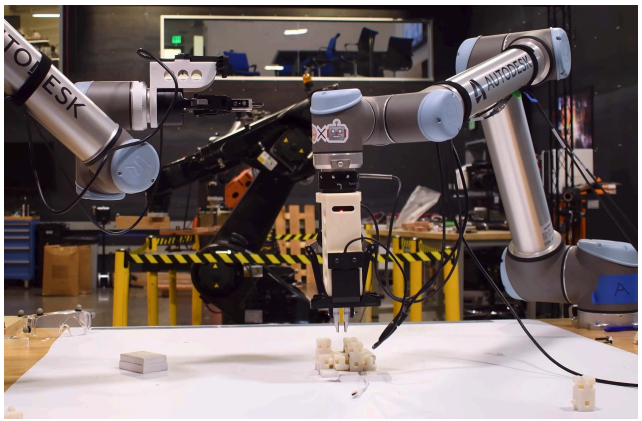


Fig. 19. Real platform setup. The robots are assembling the first layer of an interlocking cube.

Our robots were fairly precise in terms of repeatability. However, in order to coordinate two robots to work in a shared workspace, a calibration process was performed to align two robots' local frames. This calibration is not perfect and could introduce errors causing the block movement not exactly following a designed trajectory. The pose estimation can also introduce errors due to the nature of neural networks. When grasping a block, the gripper is not always holding near the center of mass, which might cause a tall block to rotate

slightly. These errors combined could cause a block to touch other blocks during the insertion process.

The blocks were 3-D printed with a rather precise printer. However, the support material covered outside each block is problematic. It is impossible to get rid of completely, and environmental conditions, such as time, temperature, and humidity, appeared to cause the support material to expand or shrink slightly, making insertion very difficult. To avoid this issue, we designed the male joint slightly smaller than the female joint such that not all contact faces of a joint give resistant friction. This design, however, makes the blocks slightly flexible causing the jamming problem later.

The biggest resulting issue of the unsatisfied assumptions was the friction. Friction can be from many different contact surfaces or points whose number and location are unpredictable. The friction led to some slight rotation of blocks which in some trials caused jamming.

ACKNOWLEDGMENT

The authors are grateful to Haopeng Zhang and Geoffrey Hsuan-Chieh Huang, who helped to build 3-D models of blocks, built robot grippers, and recorded videos. They also thank Jeremy Betz for useful insights on the geometry of joints. They would also like to thank Emily Whiting, as well as members of the Dartmouth Robotics Lab, for useful feedback and insights throughout. They also thank Adam Arnold, Heather Kerrick, Hui Li, and Mike Haley of Autodesk for their insights of 3-D printing, interlocking structures, as well as their help in design of the physical robot assembly system, and to Autodesk for the use of the lab space at Pier 9, San Francisco, CA, USA.

REFERENCES

[1] Y. Zhang and D. Balkcom, "Interlocking block assembly," in *Proc. 13th Int. Workshop Algorithmic Found. Robot.*, 2018, pp. 709–726.

[2] S. Lensgraf *et al.*, "PuzzleFlex: Kinematic motion of chains with loose joints," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 6730–6737.

[3] G. Strano, L. Hao, R. M. Everson, and K. E. Evans, "A new approach to the design and optimisation of support structures in additive manufacturing," *Int. J. Adv. Manuf. Technol.*, vol. 66, nos. 9–12, pp. 1247–1254, Jun. 2013.

[4] J. Vanek, J. A. G. Galicia, and B. Benes, "Clever support: Efficient support structure generation for digital fabrication," *Comput. Graph. Forum*, vol. 33, pp. 117–125, Aug. 2014.

[5] X. Zhang, X. Le, A. Panotopoulou, E. Whiting, and C. C. Wang, "Perceptual models of preference in 3D printing direction," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 215:1–215:12, 2015.

[6] K. Zwerger and V. Olgiati, *Wood and Wood Joints: Building Traditions of Europe, Japan and China*. Basel, Switzerland: Birkhäuser, 2012. [Online]. Available: https://books.google.com/books?id=yPoEBDZSUyoC

[7] J. P. Wilson, D. C. Woodruff, J. D. Gardner, H. M. Flora, J. R. Horner, and C. L. Organ, "Vertebral adaptations to large body size in theropod dinosaurs," *PLoS ONE*, vol. 11, no. 7, Jul. 2016, Art. no. e0158962.

[8] Y. Zhang and D. Balkcom, "Interlocking structure assembly with voxels," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 2173–2180.

[9] P. Song, C.-W. Fu, and D. Cohen-Or, "Recursive interlocking puzzles," *ACM Trans. Graph. (SIGGRAPH Asia)*, vol. 31, no. 6, pp. 128:1–128:10, Dec. 2012.

[10] P. Song *et al.*, "Reconfigurable interlocking furniture," *ACM Trans. Graph.*, vol. 36, no. 6, p. 174, 2017.

[11] P. Song *et al.*, "CofiFab: Coarse-to-fine fabrication of large 3D objects," *ACM Trans. Graph.*, vol. 35, no. 4, p. 45, 2016.

[12] C.-W. Fu, P. Song, X. Yan, L. W. Yang, P. K. Jayaraman, and D. Cohen-Or, "Computational interlocking furniture assembly," *ACM Trans. Graph.*, vol. 34, no. 4, p. 91, 2015.

[13] Z. Wang, P. Song, and M. Pauly, "DESIA: A general framework for designing interlocking assemblies," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 1–14, Jan. 2019.

[14] K. Tang, P. Song, X. Wang, B. Deng, C.-W. Fu, and L. Liu, "Computational design of steady 3D dissection puzzles," *Comput. Graph. Forum*, vol. 38, no. 2, pp. 291–303, 2019.

[15] J. Yao, D. M. Kaufman, Y. Gingold, and M. Agrawala, "Interactive design and stability analysis of decorative joinery for furniture," *ACM Trans. Graph.*, vol. 36, no. 2, pp. 20:1–20:16, Mar. 2017. [Online]. Available: http://doi.acm.org/10.1145/3054740

[16] W. Kong and B. B. Kimia, "On solving 2D and 3D puzzles using curve matching," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 2, Dec. 2001, pp. 2–583.

[17] Z. Wang, P. Song, F. Isvoranu, and M. Pauly, "Design and structural optimization of topological interlocking assemblies," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 1–13, Nov. 2019.

[18] J. Andres, T. Bock, F. Gebhart, and W. Steck, "First results of the development of the masonry robot system ROCCO: A fault tolerant assembly tool," in *Automation and Robotics in Construction XI*. Amsterdam, The Netherlands: Elsevier, 1994, pp. 87–93.

[19] C. Balaguer, E. Gambao, A. Barrientos, E. A. Puente, and R. Aracil, "Site assembly in construction industry by means of a large range advanced robot," in *Proc. 13th Int. Symp. Autom. Robot. Construct.*, Jun. 1996, pp. 65–72.

[20] V. Helm, S. Ercan, F. Gramazio, and M. Kohler, "Mobile robotic fabrication on construction sites: DimRob," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 4335–4341.

[21] M. Giftthaler *et al.*, "Mobile robotic fabrication at 1:1 scale: The *in situ* fabricator," *Construct. Robot.*, vol. 1, nos. 1–4, pp. 3–14, Dec. 2017.

[22] S. Hauser, M. Mutlu, P.-A. Léziart, H. Khodr, A. Bernardino, and A. J. Ijspeert, "Roombots extended: Challenges in the next generation of self-reconfigurable modular robots and their application in adaptive and assistive furniture," *Robot. Auto. Syst.*, vol. 127, May 2020, Art. no. 103467.

[23] S. Hauser, M. Mutlu, and A. J. Ijspeert, "Kubits: Solid-state self-reconfiguration with programmable magnets," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6443–6450, Oct. 2020.

[24] J. Willmann, F. Augugliaro, T. Cadalbert, R. D'Andrea, F. Gramazio, and M. Kohler, "Aerial robotic construction towards a new field of architectural research," *Int. J. Architectural Comput.*, vol. 10, no. 3, pp. 439–459, Sep. 2012.

[25] F. Augugliaro *et al.*, "The flight assembled architecture installation: Cooperative construction with flying machines," *IEEE Control Syst.*, vol. 34, no. 4, pp. 46–64, Aug. 2014.

[26] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of cubic structures with quadrotor teams," in *Robotics: Science and Systems VII*. Jun. 2011.

[27] F. Augugliaro, A. Mirjan, F. Gramazio, M. Kohler, and R. D'Andrea, "Building tensile structures with flying machines," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 3487–3492.

[28] S. J. Keating, J. C. Leland, L. Cai, and N. Oxman, "Toward site-specific and self-sufficient robotic fabrication on architectural scales," *Sci. Robot.*, vol. 2, no. 5, Apr. 2017, Art. no. eaam8986.

[29] D. Rus and M. Vona, "Crystalline robots: Self-reconfiguration with compressible unit modules," *Auto. Robots*, vol. 10, no. 1, pp. 107–124, 2001.

[30] P. White, V. Zykov, J. C. Bongard, and H. Lipson, "Three dimensional stochastic reconfiguration of modular robots," in *Robotics: Science and Systems*. Cambridge, 2005, pp. 161–168.

[31] J. W. Romanishin, K. Gilpin, and D. Rus, "M-blocks: Momentum-driven, magnetic modular robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 4288–4295.

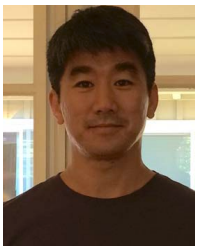
[32] H. Wei, Y. Chen, J. Tan, and T. Wang, "Sambot: A self-assembly modular robot system," *IEEE/ASME Trans. Mechatronics*, vol. 16, no. 4, pp. 745–757, Aug. 2011.

[33] J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell, "An integrated system for perception-driven autonomy with modular robots," 2017, *arXiv:1709.05435*. [Online]. Available: http://arxiv.org/abs/1709.05435

[34] T. Tosun, G. Jing, H. Kress-Gazit, and M. Yim, "Computer-aided compositional design and verification for modular robots," in *Robotics Research*. Springer, 2018, pp. 237–252.
[35] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin, "Reconfiguring massive particle swarms with limited, global control," in *Proc. Int. Symp. Algorithms Exp. Sensor Syst., Wireless Netw. Distrib. Robot.* Springer, 2013, pp. 51–66.
[36] A. Becker, G. Habibi, J. Werfel, M. Rubenstein, and J. McLurkin, "Massive uniform manipulation: Controlling large populations of simple robots with a common input signal," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 520–527.
[37] A. T. Becker *et al.*, "Tilt assembly: Algorithms for micro-factories that build objects with uniform external forces," *Algorithmica*, vol. 82, no. 2, pp. 165–187, 2020.
[38] Y. Zhang, X. Chen, H. Qi, and D. Balkcom, "Rearranging agents in a small space using global controls," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 3576–3582.
[39] S. Manzoor, S. Sheckman, J. Lonsford, H. Kim, M. J. Kim, and A. T. Becker, "Parallel self-assembly of polyominoes under uniform control inputs," *IEEE Robot. Autom. Lett.*, vol. 2, no. 4, pp. 2040–2047, Oct. 2017.
[40] A. Schmidt, S. Manzoor, L. Huang, A. T. Becker, and S. P. Fekete, "Efficient parallel self-assembly under uniform control inputs," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3521–3528, Oct. 2018.
[41] M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, no. 6198, pp. 795–799, Aug. 2014.
[42] E. Schweikardt and M. D. Gross, "RoBlocks: A robotic construction kit for mathematics and science education," in *Proc. 8th Int. Conf. Multimodal Interfaces (ICMI)*, 2006, pp. 72–75.
[43] R. Thompson, E. Ghalebi, T. DeVries, and G. W. Taylor, "Building LEGO using deep generative models of graphs," 2020, *arXiv:2012.11543*. [Online]. Available: http://arxiv.org/abs/2012.11543
[44] Y. Zhang. *Robotic Assembly of Interlocking Blocks—WAFR 2018*. Youtube. Accessed: Jan. 4, 2021. [Online]. Available: https://youtu.be/lV2xIA_Q8SI
[45] Y. Zhang. *One-Layer Structure Robotic Assembly Experiment With Two Kinds of Blocks. (2.5x Speed)*. Youtube. Accessed: Jan. 4, 2021. [Online]. Available: https://youtu.be/1_lbVyPcLOI
[46] Y. Zhang. *Two-Layer Structure Robotic Assembly Experiment With Two Kinds of Blocks. (2.5x Speed)*. Youtube. Accessed: Jan. 4, 2021. [Online]. Available: https://youtu.be/ZjFFZzrl69s
[47] Y. Zhang. *Block Regrasping Using Two Ur-10 Robot Arms. (1080p)*. Youtube. Accessed: Jan. 4, 2021. [Online]. Available: https://youtu.be/DyMBOK-Pz1I
[48] Y. Zhang. *Interlocking Cube Assembly Using Two Ur-10 Robots (4K 60FPS)*. Youtube. Accessed: Jan. 4, 2021. [Online]. Available: https://youtu.be/OaB7LMgl6rY

Yinan Zhang received the B.E. degree in software engineering from Southeast University, Nanjing, China, in 2013, and the Ph.D. degree in computer science from Dartmouth College, Hanover, NH, USA, in 2019, advised by Prof. D. Balkcom.

His research focused on motion planning, interlocking structure assembly, and manipulation.



Yotto Koga received the Ph.D. degree from the School of Engineering, Stanford University, Stanford, CA, USA, in 1994, advised by Prof. Jean-Claude Latombe.

He is currently a Software Architect with the Machine Intelligence Group, Autodesk Research, San Francisco, CA, USA, working on big data and machine learning problems. Previously, he was an Architect of Firefly, a web, mobile, and desktop 2-D and 3-D viewing platform for Autodesk products, and an Architect of Design Graph, a knowledge graph for design. His research focused on robot motion planning.

Devin Balkcom received the Ph.D. degree in robotics from Carnegie Mellon University, Pittsburgh, PA 15213, USA, in 2004, advised by Matt Mason.

He is currently a Professor with the Department of Computer Science, Dartmouth College, Hanover, NH, USA. He is interested in efficient robot designs and motion. His research highlights include exact time-optimal motion for mobile robots, robot origami, knot-tying, laundry folding, and construction of interlocking structures.

Dr. Balkcom was awarded the NSF CAREER Grant in 2006, the Dartmouth McLane Family Fellowship in 2010, and the Dartmouth John M. Manley Huntington Award in 2010.