A Digital Framework for Interlocking SL-Blocks Assembly with Robots

Binqun Liu*[1] Yuxi Liu*[1] Mehrzad Esmaeili Charkhab*[1]

Abstract— **Interlocking structures assembled by SL-Blocks offer a reversible system that allows the elements to be constantly disassembled and reassembled into various new building configurations. The building structure is kinematically immobilized by the constraints of the geometrical boundaries and friction coefficients without any adhesives. The novel concept of using SL-Blocks in architecture advances the transition to circular reuse. However, the assembly of such contact-rich Blocks also introduces new challenges to the construction process. Considering robot technologies that allow for assembly automation can respond to a lack of skilled labour and increase the efficiency of building structures with SL-Blocks. In this research, we focus on developing a digital framework for closing the loop between the design and robotic assembly of SL-Block's structures. On the one hand, we will showcase our recursive algorithm that can automate assembly sequence generation. On the other hand, we illustrate a pre-programmed setup developed for robotic assembly execution in simulation and physical environments.**

I. INTRODUCTION

In recent years, the integration of robotics into the construction life-cycle has gained in popularity due to a societal paradigm shift in sustainability, safety and carbon efficiency. Robotic automation increases productivity and efficiency, responding to a lack of skilled labour in the construction sector. The potential of industrial robots, with regards to their flexibility and adaptability, was first explored with the development of the irregular brick walls, where robotic arm is programmed to orient and place bricks into differentiated physical prototypes[1]. The concept was further extended through the implementation of a mobile robotic brick construction process in 2014, which enabled the robots to be employed directly on site[2]. In order to extend the application of brick-stacking to cantilevering structures, Stefana et al. introduced a novel fabrication process, in which two robots collaborated to place and support the elements to build a stable brick vault[3]. Robotic arm with multiple degrees of freedom allowing a wider range of fabrication speed, but require more sophisticated controls and path planning. The goal of Moving more materials more quickly is often caught in a conflict with the complexity of the construction tasks.

Robotic discrete assembly, instead, can simultaneously address both. In the process of discrete assembly, the individual building element is seen as digital material[4] with relative positions, providing kinematic constraints for physical connection and constructing large structures through its geometric property of male-female joints. This male-female connection is often seen in the morphological design of interlocking pieces for assembling timber structures. The relation can be further analogized with a binary 0-1. In that sense, digital materials establish a computational logic for the robotic assembly of discrete elements in the physical world[5]. Indeed, the above-mentioned studies are mainly attempting to prove an argument, that is, Discrete and "Digital" is supposed to be a solution to negotiating the conflict between increasing complexity and speed during the robotic construction process.

In this research context, we aim to bridge the gap between discrete design and robotic assembly of SL-Blocks, expanding the scope of the digital material to multi-level discretization beyond its one-level with binary thinking in connecting interlocking elements.

In this report, we introduce our first step-up for robotic automation of assembling monolithic building components with SL-Blocks. Advanced assembly procedures are here established by (1) the analysis of the manual assembly of SL-Blocks, (2) the development of recursive algorithms for reordering string sequences into assembly/disassembly sequences, (3) the definition of robotic setup for simulation and execution, and (4)Experiments and Evaluation of the robotic setup.

II. RELATED WORK

A. SL-Block

Module is widely used in the field of building construction and civil engineering. The advantages of modular construction are a shortened construction time, more cost-effective production of building elements, and potential facilitation of dismantling of materials. Architects and engineers are exploring self-interlocking structures with efficient modules. SL-Block[6], first proposed by Shih, is a single type of element, which can be aggregated to form hierarchical self-interlocking structures without using mortar or adhesive.

Formed by S- and L-shaped Tetracubes, SL-Block is a specially designed Polycube. Two SL-Blocks can be grouped as a conjugate pair. The rules of combining repetitive units are defined by six types of geometric transformation(represented as h, a, d, s, t, y), rather than relying on male-female joints (Fig.1). Due to that, the binary-programming logic is insufficient to handle the complexity. Based on the consecutive geometric transformations, SL-Blocks can be aggregated into a wide variety of new 3D Geometries called SL-Strands, the generation of which is a sequential process represented as string sequences(Fig.2). These SL-Strands can be further assembled into larger tectonic and reconfigurable

*All authors contributed equally

[1]Digital Design Unit, Department of Architecture, Technical University of Darmstadt, Germany. Correspondence to: BingQun Liu bingqun.liu@stud.tu-darmstadt.de

systems(Fig.3), responding to the challenges of industrial automation. Linking SL-Blocks with robotic assembly hints at a potential opportunity of developing a new paradigm for the construction industry, where multi-level discrete elements are defined from and for automation.

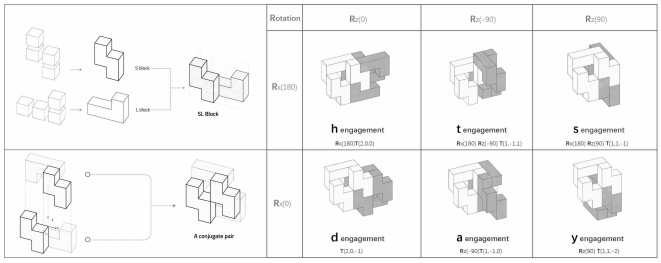

Fig. 1. SL Block, its conjugate pair, and six SL-Engagements representing the connection rules of SL-Blocks. (Image by Yuxi Liu)

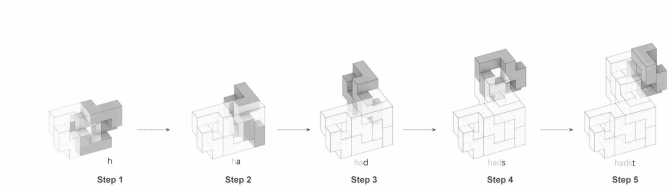

Fig. 2. The connection of conjugate pairs of SL-Block, whose process can be encoded as a string sequence. (Image by Yuxi Liu)

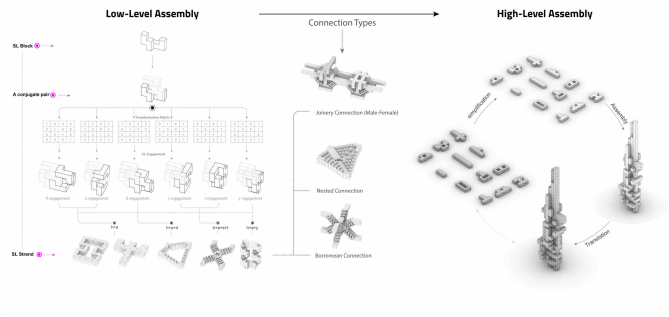

Fig. 3. A hierarchical classification of SL-Block. Within Low-Level Assembly, SL-Blocks' connection is based on six transformation rules. While in High-Level Assembly, the combination of SL-Strands is a kind of binary logic. (Image by Yuxi Liu)

B. Challenges of Manually Assenbling SL-Blocks

In the case of assembling "h" SL-Engagement(Fig.4), blocks are arranged in a linear direction. The bottom blocks are constantly rotated 180 degrees along the x-axis and assembled one by one. However, when implementing the assembly of the top blocks, two blocks need to be considered as a group and inserted together into the bottom structures.

Compared with "h" SL-Engagement, assembling "a" SL-Engagement is not block by block(Fig.5), and the assembly direction is a constant rotation of 90 degrees. A pair of two blocks should be first combined together and connected with other pairs. In this case, as eight SL-Blocks will be

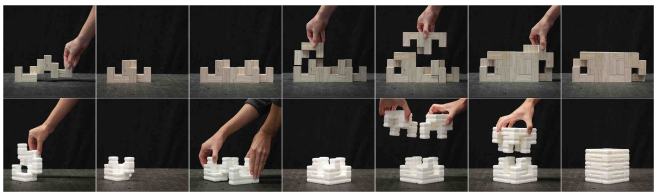

Fig. 4. Assembly sequences of the engagement "hhhh" and "aaaa"

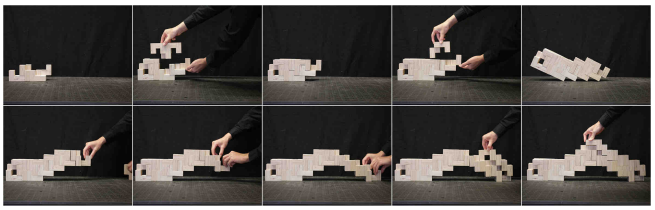

Fig. 5. Assembly sequences of the engagement "hhhdddhddd"

connected into a closed structure, the maximum number of SL-Engagements is limited to four.

By assembling the SL Blocks with the engagement "hhhd-ddhddd"(Fig.5), due to the overhanging structure, it requires support during the assembly process. Otherwise, the structure will lose its stable equilibrium.

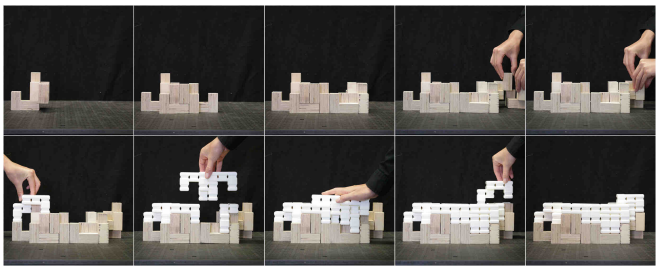

Fig. 6. Assembly sequences of the engagement "hsshhaht"

In this more complicated combination with the engagement "hsshhaht"(Fig.6). Due to errors caused by the production of blocks as well as the friction between different blocks caused by material and unoptimized dimensions, it is sometimes hard to insert blocks into the group.

All the cases above face the problem of irregular assembly sequences. When it comes to complex assembly situations, the sequence of assembly needs to be known in advance, which is time-consuming. In some cases, two blocks need to be combined in advance to complete the group assembly sequence. Therefore, assembly sequence is the first issue that should be focused on in the current study.

C. Robotic Assembly in Dry-jointed Construction

Besides the combinatorial design of building elements, discrete automation also requires robotic systems that can automate task and motion planning(TAMP), determining the sequence of assembly actions and the execution trajectories. The ideal speculations of the simulation of assembling ideal geometries and material properties require optimizations based on the feedback from the actual, physical situation.

Regarding the combination of two aspects, Tessmann and Rossi(2019) developed an approach to continuously check the aggregation of modules during the design process for construction issues like collision or instability[7]. Zhang et al. presented an algorithm automating the generation of large structures from 3D-jigsaw-like interlocking blocks. Moreover, they developed a dual-robot system that can enable parallel assembly on the same structure made of 48 blocks[8]. Roberto et al. proposed a framework for the design and assembly automation of layered timber structures, establishing the bi-directional communication channel between the simulation and the actual process.[9] Bastian et al. proposed an approach to generate a set of modules. The zick zack modules, which are designed with self-calibrating properties, to be suitable for robotic assembly, are digitally fabricated into cantilevered structures[10]. It shows the perspective, that building structures can be disassembled and the building elements can be reused.

III. Digital Design Framework for Robot-based Assembly

A. Assembly/Disassembly sequences of SL-Blocks

Different combinations of SL-Block's engagements can generate an infinite number of aggregation variations. Due to the complexity of the structure, it is necessary to have assembly instructions consisting of sequences and directions. SL-Block's engagements can't provide the assembly information because, in many cases, following the SL-Block's engagements order leads to a dead end (i.e., blocks without any degree of freedom, Fig. 7). The situation is even more complex in the cantilever structures (Fig. 5) that need to have a support structure during the assembly. This chapter will explain our approach to solving this problem.

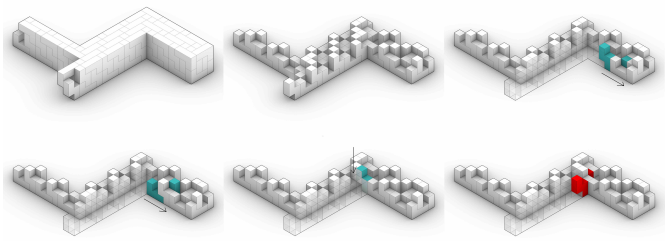

Fig. 7. Voxel based representation of SL-Blocks

A recursive algorithm is used to search through aggregation and find the possible removable blocks in each iteration until no blocks are remaining. In each iteration, removable blocks are added to a disassembly sequence list. This process allows for disassembling the remaining blocks in the subsequent iteration by opening the space and changing the degrees of freedom. Reversing the disassembly sequence list provides the assembly sequences.

To apply this algorithm, we need to answer one crucial question. How can we find the degrees of freedom of each block? For each block, there are several blockees in each direction. There are two approaches to finding blockees—first, the geometrical method, e.g., finding the intersection between geometries using 3D modeling libraries. Second, converting block's geometries into calculable data, e.g., numbers. Since the first approach increases the calculation time, we follow the second approach.

Each SL-Block consists of eight cubes that can be translated into voxels and the center points of each voxel (Fig. 8). These voxel-based geometries interact with each other in a voxelized environment. Therefore, we can save all data regarding aggregation geometries and the assembly environment with X, Y, and Z coordinate lists.

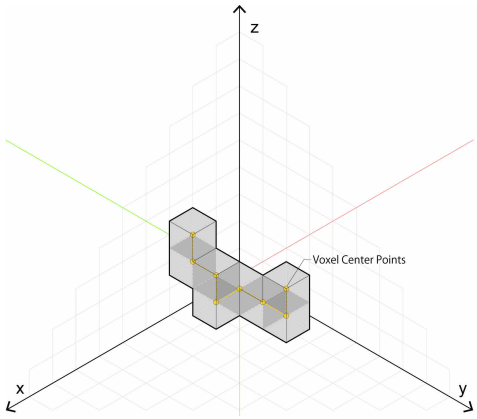

Fig. 8. Voxel-based representation of SL-Blocks (SL-Block = [[1, -1, 2], [1, -1, 1], [1,0,1], [1,0,0], [0,0,0], [0,1,0], [0,2,0], [0,2,1]])

*1) **Object-Oriented Programming**:* To calculate the blockees around each block, a list of six possible directions for assembling SL-Block is needed:

- DirectionsNames = [z, -z, x, -x, y, -y].
- Directions = [(0, 0, 1), (0, 0, -1), (1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0)].

A for loop is iterating in each direction and moving the SL-Blocks voxel points to check if there are any other obstacles (voxel points of other blocks or assembly environment). Adding the obstacles into a list creates Blockees lists as below (Fig. 9 and Fig. 13):

- BlockeesList = [
  [index number of Blockees in Z direction or assembly environment],
  [index number of Blockees in -Z direction or assembly environment],
  [index number of Blockees in X direction or assembly environment],
  [index number of Blockees in -X direction or assembly environment],
  [index number of Blockees in Y direction or assembly environment],
  [index number of Blockees in -Y direction or assembly environment]
  ]

We offer to calculate and save all the necessary data initially for later use to decrease the calculation time. So, the mentioned process can be done in another way. Voxel

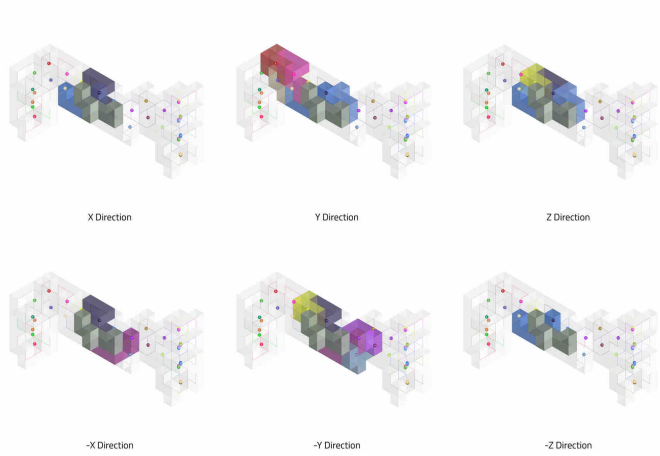

Fig. 9. Representation of Blockees in each direction

points of each SL-Block are moved in different directions to create directional checking point lists (Fig. 10). Then we iterate each point of each list to check for any obstacles in that direction and save the final list.

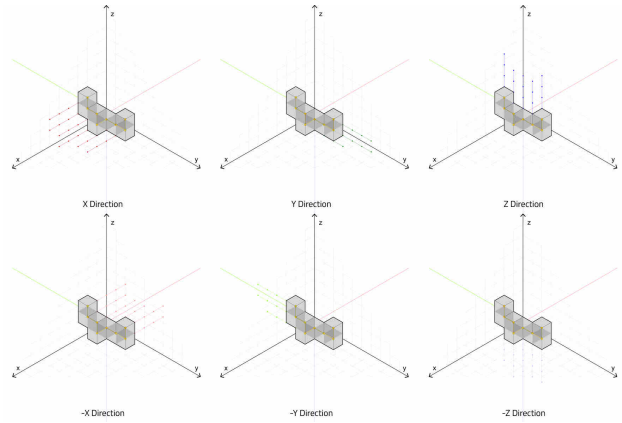

Fig. 10. Directional checking point lists

We changed the SL-Blocks geometry to SL-Blocks python objects consisting of the following data (Fig. 11): Index, Geometry, Central Line, Color, Voxel points, Directional checking points, Directional checking points, Blockees list and Geometrical Center Point.

*2) **Recursive algorithm for finding assembly/disassembly sequences**:* Inspired by Song's recursive algorithm to design 3D interlocking puzzles [11] and Blocking Graphs by Wang [12], we developed an algorithm to recursively determine the assembly sequences and directions of an existing interlocking SL-Block structure. After generating a list of SL-Blocks objects, we can start determining disassembly sequences and directions by running the recursive algorithm. There is a main for loop in each recursive level to iterate on each SL-Blocks object, and another for loop in each object iterates on directional checking point lists. Finally, the algorithm compares the checking point's X, Y, and Z coordinates to find a match in other blocks or assembly environments. When the coordinates of blocks' voxel center points match with

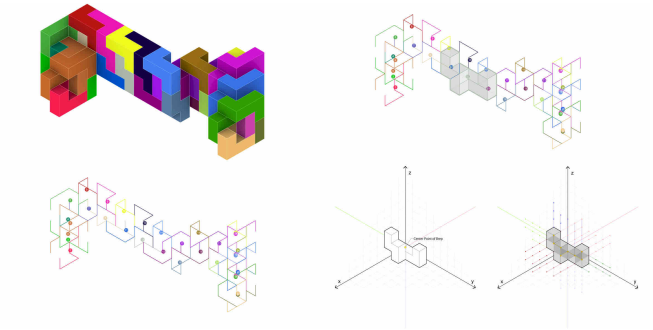

Fig. 11. Representation of SL-Block object's variables

the checking points, it means the block is an obstacle in the corresponding direction for the block which we want to move. Otherwise, the block is free in the direction. We can disassemble the block in the checking direction if there is no match (Fig. 12, left image).

In that case, the free block's object and moving direction are added to the disassembly list and removed from the main aggregation list. In the end, the list of SL-Blocks objects needs an update to remove the index number of the free block from all of the blockees lists, i.e., through this process, the aggregation space opens for the upcoming recursion.

- SL-Block. Index = 0
- SL-Block . Bloockes List = [
  Z direction = [] (Free),
  -Z direction = [3, 2, 1],
  X direction = [8, 7, 2, 1],
  -X direction = [2],
  Y direction = [1, 4, 2, 5],
  -Y direction = [1]
  ]

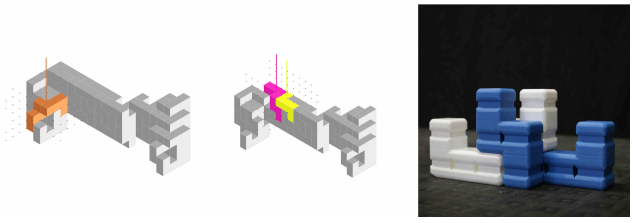

Fig. 12. Determining the disassembly directions

If there is no single block with a free direction to move (assemble/ disassemble), it is necessary to move several blocks together. Based on SL-Blocks' geometry and engagement rules, two pairs of blocks with a specific configuration can move together (Fig. 12, right image). We can find pairs of blocks that can move together by comparing the index of block and indexes of blockees in different directions. For example, in Fig. 12, the middle image represents a pair of blocks that can move together in the Z direction.

There are two conditions to detect movable pairs of blocks: if both blocks have on blockee in the same direction. And if the index number of one block is equal to the index number of the blockee of the other block, vice versa (Fig. 13). A pair

of movable blocks are moved to the disassembly list. And the list of SL-Blocks objects is updated as explained before.

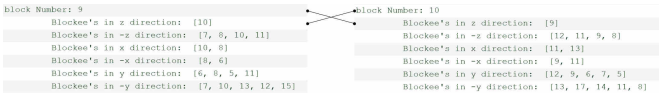

Fig. 13. Removable pairs of Blocks

The algorithm is recursively repeating until there are no blocks in the aggregation list. By reversing the disassembly list, the assembly list is provided (Fig. 14).

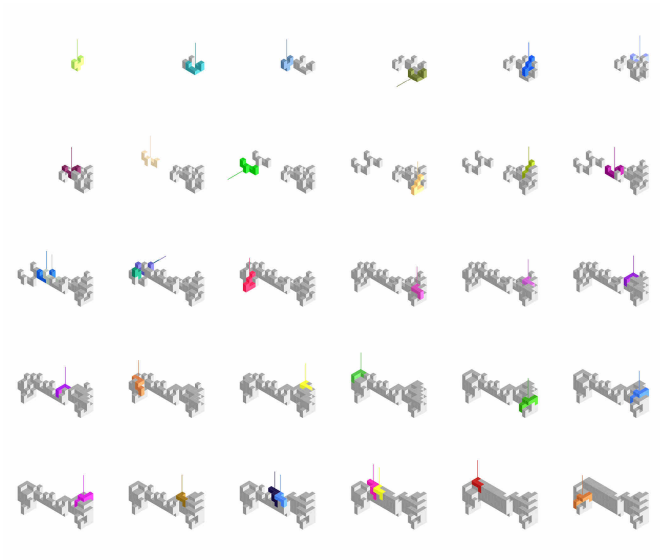

Fig. 14. Assembly Sequences of the engagement "hasthhhhttttssss"

B. Grasshopper Setup using Plugin-Robots

The Robots plugin in the Grasshopper is used to program and visualize the UR10 robotic arm. Plane is used in grasshopper to generate motion trajectory for robotic arm. A plane consists of coordinates of position and direction(Fig.15, left image), which guides the robotic arm to an exact position and orient the gripper to a specific direction. By inputting a series of planes in grasshopper, the corresponding trajectory is automatically generated by plugin-Robots(Fig.15, right image).

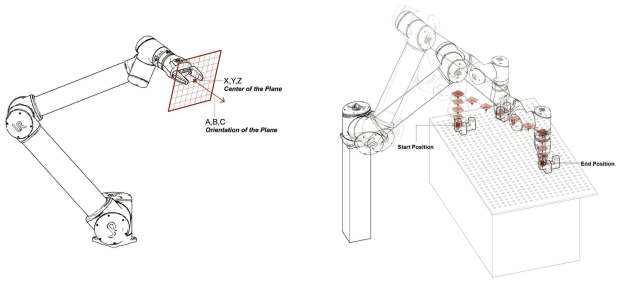

Fig. 15. Explanation of Plane and Trajectory

First, by defining the Starting point, the gripper is set to a proper position, which is ready for starting the assembly process. The second plane is set to the Gripper point, where the gripper is guided to the gripping position of SL-Block. After the gripper is closed to grip the block, it will be guided to the third plane named as Start of transporting point. Subsequently, the block is transported to the End of transporting point. A conditional statement is implemented after that, if the direction of assembly is in the Z direction or not. When it is in the Z direction(Fig.16), the Start of the placing point and the End of the placing point are the same point, the block can be released simply there. If the direction of assembly is not in the Z direction(Fig.17), the block will be taken to the Start of the placing point and then to the End of the placing point and be released at Release point. It means the block in this situation will be placed into the aggregation from the side but not in the Z direction. And the gripper will be guided to Start of the reset position and ready for the next iteration.

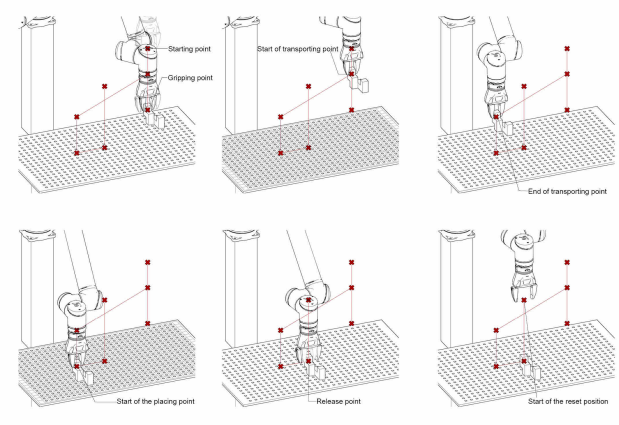

Fig. 16. Direction of assembly is in Z direction

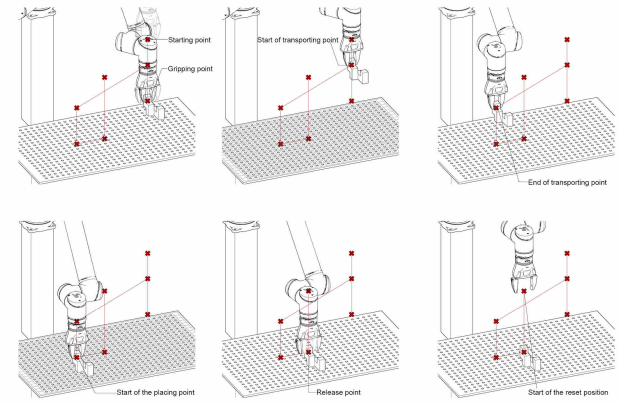

Fig. 17. Direction of assembly is not in Z direction

After defining the trajectory, we need to program the gripper to open and close in the proper position and set the movement speed of the robotic arm in each movement. Following is the final program of the robotic arm:

- Starting point - Gripper Open - Speed 100
- Gripping point - Gripper Open

- Gripping point - Gripper Close - Speed 50
- Start of transporting point - Speed 200
- End of transporting point - Speed 30

- If the direction of assembly is in the Z direction:
- Start of the placing point

- If the direction of assembly is not in the Z direction:
- Release point - Speed 50
- End of the placing point - Speed 100

- else:
- Release point - Speed 100
- End Position - Speed 200

There are four positions and combinations of blocks in the feeding section(Fig.18), and based on the assembly position and direction, the robotic arm will pick the proper block.

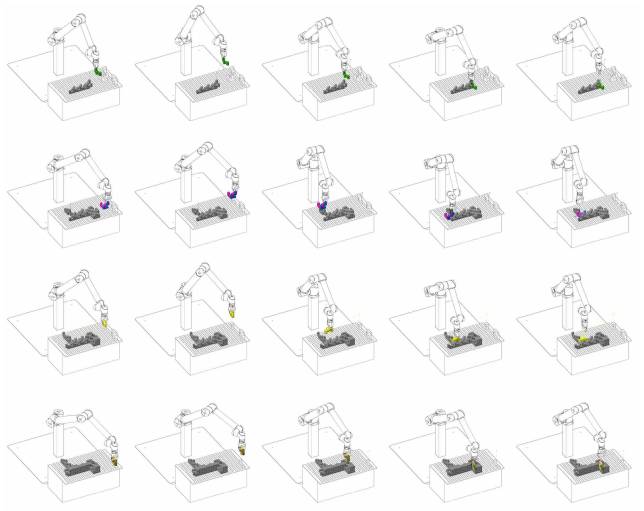

Fig. 18. Four examples of assembly sequences for different feeding positions and block directions

IV. Experiments and Evaluation

According to the assembly sequence generated by the recursive algorithm, we prepared the planes in grasshopper in advance. By applying the plane to Plugin-Robots, the trajectories are generated to execute the robotic assembly in the actual environment. At the start of assembly, we pre-placed the blocks manually on the table in a proper position(Fig.19). Afterwards, we programmed in grasshopper to execute assembly step by step.

During the assembly process, collision between blocks as well as gripper and blocks is a big issue. On the one hand, friction between blocks will cause displacement of blocks, which can lead to a collapse of the assembly sequence. And the robotic system will be interrupted due to the collision. In order to avoid these unexpected situations, it is crucial to fix blocks to ensure the stability of the assembly system. Therefore, we developed single cubes, which can be inserted and fixed with the operating table, to fix the position of SL-Blocks(Fig.19 and Fig.20 right image). On the other hand, the size of the SL-Block is another important

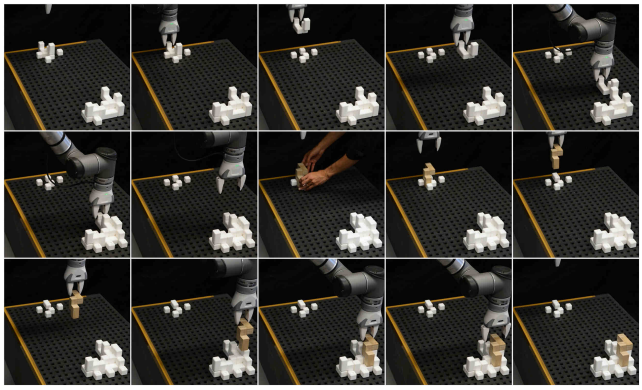

Fig. 19. Robot assembly process with shape-optimized SL-Blocks under the assistance of human and support cubes

parameter affecting the assembly. Blocks with absolutely precise dimensions (30 mm*30 mm*30 mm/cube) are subject to collisions due to friction. Consequently, we optimized the shape of SL-Block(Fig.20), such as chamfering the edges and adjusting the size to add tolerance between neighbor blocks, to reduce the friction, which ensures that SL-Block can be assembled smoothly without collision.

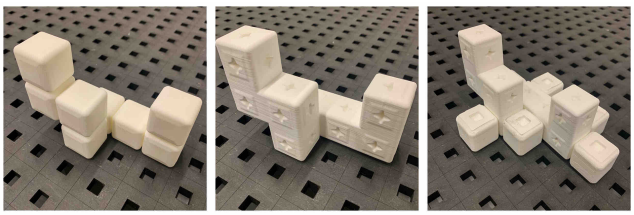

Fig. 20. Shape Optimization of SL-Block

V. Conclusion and Outlook

In this work, we developed the recursive algorithm to automate the generation of assembly sequences of SL-Blocks. Robotic simulation and physical assembly are programmed with the help of Grasshopper Plugin-Robots. However, robotic assembly in the actual environment will introduce much more issues than the simulation process. (e.g. the tolerance problem, the interruption of the robotic system arising by collision, the check of stability when assembling cantilevering structures)

In the next step, we should continue to consider the optimization of the geometrical form of SL-Blocks, as well as materiality to avoid collisions during assembly. However, collisions or accidents during assembly are inevitable in the real world. Our current experiment with pre-programmed robots is not autonomous in solving problems, requiring a combination between robot assembly sequences and machine learning, and object tracking in future research.

As such, we plan to integrate the grasshopper with ROS, while the grasshopper is pre-programmed and not able to get an approach to machine learning. We will use COMPAS, which is an interface for communicating grasshopper and ROS. Additionally, several parts require further development

in the assembly/ disassembly sequence determining algorithm. First, generating supports with a robotic trajectory for automatically rearranging them based on the progress in assembling main Blocks. Second, consider the position of the robotic picking point on SL-Blocks and choose the assembly/disassembly direction to avoid collisions between the gripper and the existing structure.

REFERENCES

[1] Bonwetsch T, Kobel D, Gramazio F, Kohler M (2006) The informed wall: applying additive digital fabrication techniques on architecture. In: ACADIA 2006 international conference, pp 489–495.

[2] Dörfer K, Sandy T, Giftthaler M, Gramazio F, Kohler M, Buchli J (2016) Mobile robotic brickwork: automation of a discrete robotic fabrication process using an autonomous mobile robot. Robotic fabrication in architecture, at and design 2016. Springer, Berlin.

[3] Parascho, S., Han, I.X., Walker, S. et al. Robotic vault: a cooperative robotic assembly method for brick vault construction. Constr Robot 4, 117–126 (2020).

[4] Felbrich B, Prado M, Saffarian S, et al. Multi-machine fabrication: an integrative design process utilising an autonomous UAV and industrial robots for the fabrication of long-span composite structures[J]. 2017.

[5] Retsin G. Discrete assembly and digital materials in architecture[J]. 2016.

[6] Shih S G. The art and mathematics of self-interlocking SL blocks[C]//Proceedings of Bridges 2018: Mathematics, Art, Music, Architecture, Education, Culture. 2018: 107-114.

[7] Tessmann O, Rossi A. Geometry as interface: parametric and combinatorial topological interlocking assemblies[J]. Journal of Applied Mechanics, 2019, 86(11).

[8] Zhang Y, Koga Y, Balkcom D. Interlocking Block Assembly With Robots[J]. IEEE Transactions on Automation Science and Engineering, 2021, 18(3): 902-916.

[9] Naboni R, Kunic A, Kramberger A, et al. Design, simulation and robotic assembly of reversible timber structures[J]. Construction Robotics, 2021, 5(1): 13-22.

[10] Bastian W, Sequential Modular Assembly: Robotic Assembly of Cantilevering Structures through Differentiated Load Modules, CAADRIA, 2020

[11] Song P, Fu C W, Cohen-Or D. Recursive interlocking puzzles[J]. ACM Transactions on Graphics (TOG), 2012, 31(6): 1-10.

[12] Wang Z, Song P, Pauly M. DESIA: A general framework for designing interlocking assemblies[J]. ACM Transactions on Graphics (TOG), 2018, 37(6): 1-14.