Automated re-prefabrication system for buildings using robotics

Cuong Kasperzyk [a], Min-Koo Kim [b*], Ioannis Brilakis[a]

[a]*Department of Engineering, University of Cambridge, Trumpington Street, Cambridge, United Kingdom*

[b]*Department of Building and Real Estate, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*

Abstract

Prefabrication has the advantages of simplicity, speed and economy but has been inflexible to changes in design which is a primary reason behind its limited market share in the construction industry. To tackle this drawback, this study presents a Robotic Prefabrication System (RPS) which employs a new concept called "re-fabrication": the automatic disassembly of a prefabricated structure and its reconstruction according to a new design. The RPS consists of a software module and a hardware module. First, the software employs the 3D model of a prefabricated structure as input, and returns motor control command output to the hardware. There are two underlying algorithms developed in the software module. First, a novel algorithm automatically compares the old and new models and identifies the components which the two models do not have in common in order to enable disassembly of the original structure and its refabrication into the new design. In addition, an additional novel algorithm computes the optimal refabrication sequence to transform one model into another according to the differences identified. Meanwhile, the hardware module takes the motor control commands as input and executes the appropriate assembly/disassembly operations, and returns the desired refabricated structure in real-time. Validation tests on two lab-scaled prefabricated structures

* Corresponding author. Tel.: +852 2766 5819 Fax: +852 2764 5131
E-mail address: minkoo.kim@polyu.edu.hk

demonstrate that the system successfully generated the desired refabrication sequences and performed all assembly operations with acceptable placement precision.

Key words: Robotic Prefabrication System (RPS), Robotics, Prefabrication, Disassembly, Refabrication

1. Introduction

In theory, most common construction components can be decomposed to a combination of parts and connectors, such as bricks and cement, wooden slabs and mating joints, or girders and bolts. It follows that most construction activities can be broken down into a series of assembly operations to form larger and larger assemblies from individual parts. Over the last few decades, individual elements, also called prefabricated components, have become popular in the construction industry. Prefabrication is a construction practice which manufactures the majority of building´s sub-assemblies ranging from wall panels to complete rooms in a controlled factory environment, before transporting the sub-assemblies to the construction site for assembly [1]. Modular buildings and modular homes, which are recently getting more popular in the construction industry, are a representative example of adopting the concept of prefabrication [2]. Compared to site-cast (or in-situ) construction, precast concrete elements offer faster production, lower cost, and more efficient assembly of elements [3]. For example, it has been reported that replacing in-situ concrete casting panels with prefabricated elements has resulted in a 70% reduction in construction time and a 43% reduction in labour cost [4]. Moreover, the use of precast concrete elements leads to a cleaner and safer construction environment [4-5].

Despite these benefits, off-site construction methods are estimated to comprise only around 10% of the construction market of UK [6]. There are numerous technical, financial and regulatory barriers that contribute to such a slow adoption of prefabrication [7]. While the relative prominence of most of these barriers is still open to debate, there seems to be a general consensus within the industry as stated that *"The main disadvantages of prefabrication are inflexibility to changes in design."* [5]. This study focuses on tackling the main disadvantage of prefabrication: the inflexibility of prefabrication to changes in design.

Current construction industry practice aims to increase flexibility by mass customization to overcome the shortcoming [8]. This involves the mass production of certain core designs which can later be customized using a catalogue of modules: a plain timber panel, for example, can be switched for a panel with thermal insulation layers and window frame components pre-fitted. This approach requires automation as a prerequisite since any change to the repetition of parts slows down production until the entire process is fully automated, including assembly and not just the making of the parts [2]. The need for an automated and mass-customisable construction process thus motivates developments in the field of 'robotic prefabrication'. It was argued that the level of automation in making prefabricated building components using robots in the precast concrete industry is high and this has mainly stemmed from the flexible production system which could execute various tasks such as setting moulds and placing reinforcement bars [9].

Even though mass customization using robotic fabrication has improved flexibility during the design process, design changes such as those arising from inspection failures or changes in customer requirements can no longer be incorporated once the design has been physically built. Flexibility can thus be further improved if it becomes possible to *automatically disassemble a prefabricated structure and reconstruct it according to a new design* - a concept which shall be referred to from here onwards as "refabrication".

Not only will a solution to this problem associated with automation and refabrication help accentuate the benefits of prefabrication over bespoke construction and increase its market share, but also it will boost productivity levels. It was reported that approximately 40% of construction projects experience more than 10% change [10]. It was also estimated that productivity will drop below the estimated level for projects with more than 20% change, and conversely productivity will increase when change is effectively dealt with and kept below 5% [10]. Based on the statistical productivity estimation in the previous study, development of a solution with the capability of automated refabrication can increase the productivity as changes in design can be addressed in a timely and effective manner. Moreover, this solution will provide positive environmental impact: When subjected to customers´ order changes or inspection failures such as a joint failing under load or a component exceeding tolerance limit, a modification of the original structure is much less wasteful than a complete demolition. In this sense, an automated disassembly and refabrication solution in the prefabrication industry can significantly contribute to the development of sustainable construction which attempts to reuse the components and other resources needed for construction [11].

This study presents a new concept and demonstrates the idea to increase the flexibility of prefabrication through the early development of a refabrication system using robotics. A Robotic Prefabrication System (RPS) that employs a new concept "refabrication" is presented here. The RPS consists of a software module and a hardware module which are detailed in Section 3.

The rest of this paper is organized as follows. Section 2 reviews current state-of-practice and state-of-research into robot-aided construction. The proposed system and its modules are then presented in Section 3. Validation tests are conducted and the results are reported and analysed in Section 4. Finally, conclusions are drawn and recommendations for future work are discussed in Section 5.

2. Related work

It is often argued that the construction industry has the features of a loosely coupled system which favours productivity in projects while innovation suffers [12]. A number of researchers have also argued that the construction industry has failed to adopt techniques that have improved performance in other industries such as just-in-time [13] and 'industrialization' of manufacturing processes [14]. In this regard, the construction industry particularly in the prefabrication sector needs to revolutionize by embracing such advanced automation techniques and systems. This section presents related studies and attempts that has been made so far regarding robotic based automation in the construction industry to identify the needs and gaps in knowledge in the current prefabrication domain.

2.1. Robot-aided automated construction in the building industry

Over the past few decades, automation systems using robot technologies has been less favourably developed and applied in the construction and building industry compared to the industrial and the manufacturing industry because of the dynamic and uncertain environments of the industry [8, 15]. In an attempt to automate repetitive construction processes and increase the productivity in construction, several robotic systems such as slab finishing robot system and concrete formwork cleaning robot system, were developed in the 1980s [16-17]. Skibnieswski also conducted the feasibility study on selected construction industry processes in order to examine the possibility of using robots in the future construction industry [16]. During the 1990s, Japanese companies and universities led the R&D activities in the field of robot-aided automated construction and the focus was the development of new robotic systems and the automation of existing machinery [9]. These robots developed for house buildings tried to automate certain construction processes such as layering bricks, constructing building walls and facades [18-21]. However, the 'bubble economy' crisis in Japan had reduced investment in the research area, and only few construction robots had succeeded in the market. As the result of the risk of high initial cost and the unsatisfactory return on investment, construction industry had continued to be conservative in "tomorrow's construction robots" [8].

Regarding the recent development of construction robots for buildings, there are some commercial systems available in the market such as SAM [22], Contour Crafting [23] and Oversize 3D printing systems [24]. SAM is a semi-automated mason robotic bricklayer and has a function of laying about 800 to 1,200 bricks a day while a human mason can lay about 300 to 500 bricks a day. This robot, however, still requires a human construction worker to tidy up the mortar and place bricks in difficult area such as corners. Another innovative development named Contour Crafting is a layered fabrication system designed for automating the construction of whole structures. This system, however, has not reached the stage of constructing a complete housing or building with a satisfactory accuracy. D-shape is a large 3D printer that uses a layer-by-layer printing process to create stone-like objects. It is reported that the printer still needs to be further developed in order to make larger and more complex buildings [24].

In addition to the commercial systems mentioned above, several academic studies have been conducted. Choi et al. [25] developed a construction robot using pneumatic actuator and servo motor to support construction workers in mounting window glasses or fixing panels. A cable-robot system called 'SPIDERobot' was also developed to perform assembly operations in on-site architectural construction [26]. Chu et al. [27] presented the development of a robotic beam assembly system consisting of a robotic bolting device that performs the main function for the beam assembly work and a robotic transport mechanism that transports the robotic bolting device to target bolting positions around a building under construction. However, it seems that the recent studies have focused on development of robot systems with the purpose of automating the construction or maintenance tasks, which has limitations in overcoming the inflexibility problem mainly occurred in the design and manufacturing phase of a project.

2.2. Robotic prefabrication in the building industry

Robotic systems have been mainly employed in the prefabrication construction industry for the production of modular and prefabricated housing components such as ceilings, walls and roofs. Bock [17] detailed a robotic precast concrete panel factory that utilizes a multi-functional formwork unit which allows flexible production of concrete floors, walls and roof panels. In this factory, a precast manufacturing system, which integrates CAD with Computer-Aided Manufacturing (CAM), controlled concrete distributor to spread the right amount of concrete by taking into account the geometric position of window or door openings according to CAD layout.

Three primary projects which illustrate the advances and the state of the art of the robotic prefabrication in the building industry are: (1) ROCCO [18], (2) FutureHome [19-20], and (3) ManuBuild [21].

ROCCO [18] features two different robotic systems: one for erection of walls in residential buildings with a reach of 4.5m and a payload of 400kg, and one for industrial buildings with a reach of 8.5m and a payload of 500kg. It includes a software system that assists engineers in wall partitioning, layout planning and logistics planning. The system is also capable of automatically generating manufacturing commands and robot assembly tasks to produce prefabricated elements.

FutureHome [19-20] aims to build fully-manufactured houses instead of only prefabricated parts. The hardware now features both an off-site production plant and on-site assembly plant, with a robotized gantry crane to perform on-site assembly tasks. The software system, AUTOMOD3, generates assembly sequences and motion paths for robots to automatically carry out the construction process. It also provides a simulation tool to allow the assembly process to be visualized and inspected before execution.

ManuBuild [21] facilitates the adoption of mass customization in the construction industry. This project targets a breakthrough from a "craft and resource-based construction" industry into an "open and knowledge-based manufacturing" industry, leading to not only make buildings as open systems equipped with flexible and scalable components but also offer customers increased choice and design flexibility.

Recently, the group of Gramazio Kohler Research at ETH Zurich have developed numerous automated robot systems including a mobile robotic brickwork system [28] and an aerial robotic construction system [29]. These studies are recognized as a meaningful contribution to the additive non-standard fabrication for the assembly of building components.

2.3. Robotic disassembly and reconstruction

Nevertheless there have been academic and practical studies aiming to develop robotics-based automated assembly systems as investigated in Sections 2.1 and 2.2, it has been found that there is still no study available dealing with automated disassembly and reconstruction of prefabricated structures in the construction industry. In order to tackle this limitation in the current prefabrication industry, a new system that provides the capability of automated disassembly and refabrication was proposed in this study. This study adopts the most common assembly planning strategy 'assembly-by-disassembly'. This is because (1) when only geometric constraints are considered, an invertible disassembly sequence always leads to a feasible assembly sequence; and (2) a structure in its assembled state has many more constraints than in its disassembled state, which results in a smaller search space for the planner [30-32]. For this reason, knowledge from the field of automated product assembly, which has been widely researched since the late 1980s, is directly relevant to the disassembly and refabrication of prefabricated structures.

The core algorithmic parts of the automated product assembly include geometrical reasoning in assembly planning [33], stability analysis of assemblies [34] and assembly sequencing using a path planning approach [31]. Recently, Rakshit and Akella [35] combined stability and geometric constraints analysis to produce an algorithm capable of simulating the entire assembly sequence by taking into account physical forces and part motion. This algorithm is outlined below:

Assumptions:

- The sequence is two-handed and monotone
- Each part is moved by a gripper at constant velocity with perfect position control
- Part movement is modelled as quasi-static motion with finite translations
- Collision of gripper with assembly is not considered

Geometric analysis:

- Firstly, an enumeration of all possible sequences is generated using AND/OR graph [30]
- Secondly, geometrically feasible sequences are filtered out using Non-Directional Blocking Graph [36]

Stability analysis:

- For frictionless cases, calculation of the relative movement in terms of the relative acceleration between the parts in the assembly [37] is conducted
- For cases with friction, Baraff´s method [37] becomes ineffective and a different set of complementary constraints must be used [38]

2.4. Gaps in knowledge and scope of this study

Even though the state-of-the-art algorithm developed by Rakshit and Akella [35] can be used to generate a stable disassembly sequence for the majority of common structural assemblies, this is only part of what is needed to realize the concept of “refabrication” which also requires the refabrication sequence based on a new design. Therefore, the objective of this study is to develop a RPS of prefabrication that provides the automatic disassembly and reconstruction of a prefabricated structure. The concept is demonstrated using an automated robotic system operating on a small-scale structure to provide the first stepping stone for future researchers working towards the final goal: refabrication of arbitrary full-scaled structures.

Refabrication is an extension of the general assembly planning problem, which includes many sub-problems such as connector design and manipulation, feeder and tool selection, assembly sequencing, and robot path planning. However, since this study focuses on proving the proof-of-concept of the RPS as a first stepping stone, a full treatment of all aspects above is beyond the scope of this work. The simplifications made in this study are:

- The robot arm can only move in 2D (a vertical plane with respect to the ground)
- The path planner[1] produces collision-free but non-optimal paths
- The assembly sequencer[2] only takes into account:
    + "Stacking" operations (pure translations and no rotation)
    + Geometric constraints (ignore stability constraints)
    + Two-handed monotone assemblies.
- Only two types of connectors were considered:
    + Null connectors: where two parts are kept in contact purely by gravity (e.g. Jenga blocks)
    + Permanent connectors: where two parts are connected through a joint which is impractical to undo after the assembly operation is completed (e.g. cemented bricks).

[1] A path planner calculates paths in space that a robot arm can take to execute a specific assembly sequence. These paths are often subject to a certain set of constraints, such as collision-free or optimal-time.
[2] An assembly sequencer produces a set of assembly operations and constraints on their ordering. Each operation specifies a motion that combines two or more subassemblies to form a larger assembly. Any ordering of operations that obey the sequence constraints is called an assembly sequence.

3. Development of Robotic Prefabrication System

3.1. System design

3.1.1. Top level

The RPS is designed with the capability of automatically building a 3D structure given its digital model, as well as of deconstructing obsolete parts and updating the original structure given a new design. This capability can be divided into two main functions, which are ‘assemble’ and ‘refabricate’ as illustrated in Figure 1. When the RPS implement the ‘assemble’ function, the digital 3D model of a structure and raw material are fed into the RPS as inputs and a 3D structure is assembled according to the original design. Meanwhile, when a new 3D model comes into the RPS due to a change in design, the RPS implements the tasks of disassembly and reconstruction according to the new design and finally results in a new 3D structure.

Insert Figure 1 approximately here

3.1.2. Second level

The RPS includes both a software module and a hardware module to meet top-level functional requirements. The software module takes digital inputs and gives motor control commands to the hardware module, while the hardware module takes the motor control commands, manipulates the physical inputs, and returns physical outputs. In addition, to carry and update information about motor states, a feedback loop from the hardware module to the software module is included.

3.1.1 Third level

For the software module, there are four software sub-modules needed to carry out the second-level function described above. Figure 2 illustrates the workflow of the software module. The functions of each software sub-module are described as follows:

- *Model analyser*: Analyses an input 3D model and returns the its geometric data, such as the size, shape and position of its individual parts.
- *Models comparator*: Takes in geometric data from two different 3D models, identifies all those individual parts which the two models do not have in common, and returns the geometric data of these parts.
- *Assembly sequencer*: Takes in geometric data of the entire model or set of specific parts, depending on which top-level function the RPS needs to execute, returns the appropriate (dis)assembly sequences.
- *Hardware controller*: Takes in (dis)assembly sequences and generates a set of motor control commands such that the hardware will carry out the appropriate (dis)assembly operations and create the desired 3D structure. The controller also needs to take in the feedback signal containing the motor states from the hardware module, in order to synchronize the execution of motor control commands.

Insert Figure 2 approximately here

More details of the design of the software sub-modules are presented in Section 3.3.

Since the task of the hardware module is common to many existing assembly systems in industry, different types of systems were investigated to pick out one as a suitable template. However, due to the limited variety of components available for construction of the hardware system as well as the large number of motors required, it became clear that assembly design typically employed in industry was impractical to pursue in this study. Therefore, a basic hardware module was designed specifically for this study to fulfil our objectives. Figure 3 shows the hardware module designed in this study. The hardware module comprises four sub-modules**:**

- *Gripper*: Securely holds a raw material block during its transportation from stock side to assembly side, and vice versa.
- *Lift drive*: Enables vertical translation of the gripper
- *Forward drive*: Enables longitudinal translation of the gripper
- *Support structure*: Provides an elevated runway for the forward drive on top, as well as stock space and assembly space at the bottom

Insert Figure 3 approximately here

More details of the design of the hardware sub-modules are presented in Section 3.4.

3.2. Choice of materials

Two constituent component options were considered for the choice of materials for this study:

- Fully-customized components: Structural components could be designed using CAD software packages, then either machined or manually created in a workshop. This enables great flexibility in design, but it is relatively time and cost demanding during the manufacturing and construction of the components.
- Standardized components: Structural components could be built directly out of LEGO Duplo block and LEGO Mindstrom [39] components. Actuators are also available as servo motors from the LEGO Mindstorms set. This gives limited flexibility in design, but requires relatively little time during the construction of the sub-modules.

The use of standardized components to construct the entire hardware module can act as supporting evidence for the philosophy advocated in this study that many structures can be efficiently constructed through the assembly of modular components. For this reason, it was decided that the hardware module would be constructed entirely out of LEGO Duplo and LEGO Mindstorm components. This reasoning also applied to the choice of building block used as input raw material to the hardware module; LEGO Duplo and Jenga blocks were therefore chosen. Since Jenga blocks are held in contact by gravity alone, and Duplo blocks are held in contact by fairly sturdy male-female connectors, they here represent null and permanent connectors, respectively.

3.3. Software module

3.3.1. Model analyser

The model analyser is built to analyse an input 3D model and return the model' geometric data. Since the 3D structures which our system operates on are cuboid, the geometric data extracted are: (1) The coordinates of each component's centroid, (2) The size of each component's bounding box (width, length and height), and (3) The ID of each component (which must contain the string "Lego" or "Jenga" so that the type of connector it possesses can be later inferred).

Having identified the output requirements above, an algorithm was developed to take in a file of 3D model and return an array containing three variables {id, boundBoxSize, centrePoint} (see ***Algorithm*** *1*).

Let <C> be an array containing individual components found in the input 3D model and <G> be an array representing the geometric data of the components:

Algorithm *1: Model analyser*

Input: A 3D model file, e.g. 'model.ifc'

```
1:    do <C> = ReadModel( );
2:    if <C> == ∅ then
3:          return NO COMPONENT FOUND
4:    end if
5:    for each C in <C> do
6:          G.id = C.GetName( );
7:          G.boundBoxSize = C.GetSize( );
8:          G.centrePoint = C.GetCentroid( );
9:          <G>.Add (G);
```

```
end for each
return <G>
```

3.3.2. Model comparator

The role of the model comparator is to take in geometric data of two different 3D models, identify all individual parts which the two models do not share in common, and return the geometric data of such parts. A function was created to do the tasks. It takes in two arrays containing the geometric data of two different 3D models, loops through each member of the first array, and checks if it also exists in the second array. Finally, it returns an array containing all members of the first array that do not exist in the second array. However, this function itself only returns true if the pair of array members being compared have the exact same variables (i.e. fully identical). This means that given two 3D models which are identical in every aspect except their position in space (i.e. partially identical), the models comparator will conclude that these two models have zero common parts.

In order to tackle this issue, a new function that helps align the coordinates of the two input models was created. It is, however, a non-trivial problem to align two arbitrarily different models such that the alignment should lead to as few refabrication operations as possible. It is also impractical to attempt every possible alignment of large models since the number of checks is proportional to N^2, where N is the number of partially identical parts and determined from a brute-force alignment approach. Hence, assuming that it is focused on prefabrication of building walls and the majority of design changes are either wall extensions while keeping the door/window positioning or repositioning of door/window while keeping the wall dimensions, the number of alignments attempted can be limited to three: (1) Alignment of lower left corner, (2) Alignment of lower right corner, and (3) Alignment of door feature. Three sub-functions were thus developed: ‘AlignLeft()’, ‘AlignRight()’ and ‘AlignDoor()’.

Let <G1> and <G2> be arrays representing the geometric data of components from two different 3D models; <G2L>, <G2R> and <G2D> be arrays representing the geometric data of components from the second model after left, right and door alignment respectively; and <L>, <R> and <D> be arrays representing not-in-common components between <G1> and <G2L>, <G2R>, <G2D> respectively:

***Algorithm 2**: Model Comparator*

```
Input: <G1>, <G2>
    do <G2L> = AlignLeft (G2);
    do <G2R> = AlignRight (G2);
    do <G2D> = AlignDoor (G2);
    foreach G1 in <G1> do
        if <G2L>.Contains(G1) == false then
            <L>.Add(G1);
        end if
        if <G2R>.Contains(G1) == false then
            <R>.Add(G1);
        end if
        if <G2D>.Contains(G1) == false then
            <D>.Add(G1);
        end if
    end foreach
    return <L>, <R>, <D>
```

Note that since the model comparator does not have the capability to evaluate the number of refabrication operations required as a result of model alignment, it must pass on the geometric data of not-in-common parts for all three alignment scenarios to the next sub-module, the ‘Assembly sequencer’.

3.3.3. Assembly sequencer

The assembly sequencer can execute two functions, “Assemble” and “Refabricate”. If the system is executing the “Assemble” function, the assembly sequencer takes in the geometric data of the components previously extracted from the 3D model, and returns the appropriate sequence of assemblage. Since it is already assumed that all raw material blocks are cuboids, an effective stack-assembly sequencing algorithm is as follows:

- Search through all members of the input array containing geometric data
- Sort the members in ascending order according to the distance between ground and each member's bottom bound line
- Then proceed to sort the members in descending order according to the distance between the stock side and each member's right-hand-side bound line

This algorithm will thus return an array whose members are indexed in such a way that the building blocks will be assembled from the bottom layer up and from the far end of the assembly side towards the stock side. This is illustrated in Figure 4.

Insert Figure 4 approximately here

Let <G> be an array representing a set of all components of a 3D model and its geometric data; <S> be an array representing the same components now indexed according to the desired assembly sequence; and <bottomBoundLine> and <rightBoundLine> be arrays containing the position of the bottom and right bounds of the components' geometry:

Algorithm 3: *Assembly Sequencer (executing the "Assemble" function)*

```
Input: <G>
1:    foreach G in <G> do
2:          bottomBoundLine = G.centrePoint.Y – G.boundBoxSize.Y ÷ 2
3:          rightBoundLine = G.centrePoint.X + G.boundBoxSize.X ÷ 2
4:          <bottomBoundLine>.Add(bottomBoundLine)
5:          <rightBoundLine>.Add(rightBoundLine)
4:    end foreach
5:    do <S> = <G>.OrderBy(<bottomBoundLine>).ThenByDescend(<rightBoundLine>)
6:    return <S>
```

If the system is executing the "Refabricate" function, the purpose of the assembly sequencer is to take in the geometric data of not-in-common parts for all three alignment scenarios outlined above, evaluate which alignment is the most optimal, then return the appropriate “refabrication sequence”. Here, the most optimal alignment is defined here as *the alignment which results in the minimum number of (dis)assembly operations required to refabricate the existing structure* and this in turn begs the question on *how can one calculate the number of (dis)assembly operations required*? This question can be answered using the Non Directional Blocking Graph (NDBG) technique developed by Wilson [33]. This technique involves three main steps: Step 1 - the construction of directional blocking graphs (DBGs), where each one indicates which parts within the assembly would collide given an instantaneous displacement in a particular direction; Step 2 - the partitioning of space into regions which share the same DBG; Step 3 - the combination of all DBGs to form the NDBG.

However, since this technique can be applied to assemblies of arbitrary polygons and accounts for arbitrary linear motion in 3D space, it is too generalized for the purposes of this study. Consequently, a stripped-down version of the NDBG technique was used to develop the assembly sequencer. Figure 5 provides an example illustrating how the NDBG technique can be simplified when all (dis)assembly operations are restricted to 2D stacking operations:

- The left hand side is an example stack assembly consisting of four subassemblies, P1, P2, P3 and P4.
- The top right side shows the DBG whose nodes represent the subassemblies and where each outgoing arrow indicates an expected collision when given an instantaneous displacement in the vertically upwards direction. Since vertically upwards is the only direction allowed for disassembly operations, the NDBG is the same as the DBG and steps 2 and 3 of the NDBG algorithm can be skipped.
- The bottom right side represents the DBG as a matrix. The matrix rows and columns represent all possible origin and destination nodes of DBG, while the elements 0 and 1 of the matrix represent the absence or presence of all possible DBGs.

Insert Figure 5 approximately here

In order to calculate the number of disassembly operations required for any chosen sub-assembly, the values of the matrix elements must first be determined and the optimal disassembly sequence be deduced. The values of matrix elements are determined using a function called 'CalcDBG()' which takes an array with N members containing geometric information of the assembly, and returns an N by N matrix which represents the DBG of the assembly.

The algorithm implemented is outlined below:

1. Create an N by N matrix with all elements set to zero
2. For each subassembly (denoted as A), check the bounding box of any other subassembly (denoted as B) and see if both of the following conditions are satisfied:
    - The top line of the bounding box of A is at the same height as the bottom line of the bounding box of B.
    - The bounding box of B lies in the "collision zone", which is defined as the 3D space covered by the bounding box of A when extended in the vertical direction.

    If yes, change the appropriate matrix element to one.
3. Terminate when step 2 has been performed for all subassemblies.

Let <G> be an array representing all components and their geometric data from a 3D model; and dbg[] be a matrix which represents the DBG of the same model:

Algorithm 4: *Function CalcDBG( )*

Input: <G>
do N = <G>.GetLength()
do dbg[] = NewZeroMatrix(N, N)
for i = 1, 2...N **do**
    for j = 1, 2...N **do**

```
            if G_j.BottomLine == G_i.TopLine then
                if Collision (G_i, G_j) == true then
                    dbg [i,j] = 1
                end if
            end if
        end for
    end for
    return dbg
```

Note that the above function is based on the original DBG technique, which assumes that all subassemblies are free-flying and held together via null connectors. However, since our system operates on assemblies with the presence of permanent connectors, another function called 'CalcTruncatedDBG()' was generated. This function takes the N by N matrix produced by the CalcDBG() function and returns a M by M matrix, where M = (N - the number of permanent connectors), using the algorithm below:

1. Find all matrix rows which contain "1" element
2. For each row found in step 1, check the following cases of its "1" elements:
    - If the two subassemblies involved are not held together by a permanent connector, skip to the next "1" element.
    - Otherwise, perform the following operations on the rows and columns which represent two subassemblies involved (here denoted as A and B):

      + Combine column of B and column of A using Boolean OR

      + Combine row of B and row of A using Boolean OR

      + Set all elements on the matrix diagonal to zero
3. Terminate when step 2 has been performed on all rows

Let <G> be an array representing all components and their geometric data from a 3D model and let dbgTrunc[] be a matrix which represents the truncated DBG of this 3D model:

Algorithm 5: *Function CalcTruncatedDBG( )*

```
Input: <G>
do N = <G>.GetLength()
do dbgTrunc[,] = CalcDBG(<G>)
for i = 1, 2...N do
    for j = 1, 2...N do
        if dbgTrunc[i,j] = 1 then
            if PermCon (Gi, Gj) == true then
                combineOR(dbgTrunc[i,*], dbgTrunc[j,*])
                combineOR(dbgTrunc[*,i], dbgTrunc[*,j])
                dbgTrunc[i,i] = dbgTrunc[j,j] = 0
            end if
        end if
    end for
end for
return dbgTrunc
```

An illustrative example of a transformation from the DBG matrix shown in Figure 5 to a new truncated DBG matrix is provided in Figure 6.

Insert Figure 6 approximately here

Once all matrix elements are calculated, the optimal disassembly sequence for any chosen subassembly are determined using a function called GetDisassemblyTree(). This function takes in three pieces of information, (1) an array with N members containing the geometric information of the assembly, (2) the M by M matrix produced by the CalcTruncatedDBG() function and (3) the geometric information of the subassembly that needs to be removed, and then returns an array with L members where L = the number of subassemblies that need to be removed as a consequence. Members of the output array contain the geometric information of the to-be-removed subassemblies and the ordering of these members represents the sequence in which they need to be removed. The implemented algorithm is as follows:

1. Jump to the matrix row corresponding to the subassembly that needs to be removed from the overall assembly, here denoted as subassembly A.
2. Search the current row for “1” elements:
    - If one or more “1” elements are found, go to step 3.

- Otherwise, add the subassembly to disassembly tree and check:

  + If the added subassembly is not A, go to step 4.

  + Otherwise, terminate the algorithm.

3. Jump to the row whose index is equal to the column index of one of the “1” elements found in step 2, and repeat step 2.
4. Jump to the row visited immediately before the current row, and repeat step 2.

Let <G> be an array representing all components and their geometric data from a 3D model; dbgTrunc[] be a matrix which represents the truncated DBG of the 3D model; G* be a representation of the component to be removed from the 3D model; and <T> be an array representing the disassembly tree:

Algorithm 6: *Function GetDisassemblyTree( )*

```
Input: <G>, G*, dbgTrunc[,]
 1:  do N = <G>.GetLength()
 2:  do index = CorrespondingRow (G*, dbgTrunc[,])
 3:  i = index
 4:  if GetCountOnes(dbgTrunc[index,*]) > 0  then
 5:      for j = 1, 2…N do
 6:          if dbgTrunc[i,j] = 1 then
 7:              i = j
 8:              JUMP TO LINE 4
 9:          end if
10:      end for
11:  end if
12:  if GetCountOnes(dbgTrunc[index,*]) == 0 then
13:      <T>.Add(G_i)
14:      if i ≠ index then
15:          i = GetPreviousI(i)
16:          JUMP TO LINE 4
17:      end if
18:      if i = index then
19:          return <T>
20:      end if
21:  end if
```

Using this algorithm on the example in Figure 5, a request to remove P3 will return the disassembly tree in the form of an array {P1, P2+4, P3}.

Now the optimal disassembly sequence can be calculated for any chosen subassembly and the three arrays returned by the ‘Model Comparator’ sub-module can finally be evaluated. A function ‘GetDisassemblyForest()’ was generated to compute the optimal alignment and return the optimal disassembly sequence, which is outlined below:

1. For each of the three input arrays:
   - Calculate the disassembly tree for each array member using GetDisassemblyTree()
   - Concatenate all disassembly trees and remove duplicated members to obtain a new disassembly sequence, here denoted as a *“disassembly forest”*
   - Count the members of the disassembly forest
2. Compare the three counts obtained from step 1 and choose the alignment type which resulted in the lowest count.
3. Return the disassembly forest associated with the alignment type chosen in step 2

Let <L>, <R> and <D> be arrays representing not-in-common components returned by the ‘Model Comparator’ sub-module; <fL>, <fR> and <fD> be arrays representing the disassembly forests resulting from the three alignment scenarios:

Algorithm 7: *Function GetDisassemblyForest( )*

```
Input: <L>, <R> and <D>
foreach L in <L> do
        <fL>.Add( GetDisassemblyTree(L))
        <fL>.RemoveDuplicates()
end foreach
do countLeft = <fL>.GetLength()
foreach R in <R> do
        <fR>.Add( GetDisassemblyTree(R))
```

```
        <fR>.RemoveDuplicates()
end foreach
do countRight = <fR>.GetLength()
foreach D in <D> do
        <fD>.Add( GetDisassemblyTree(D))
        <fD>.RemoveDuplicates()
end foreach
do countDoor = <fD>.GetLength()
do minCount = GetMinimum(countLeft, countRight, countDoor)
if minCount = countLeft then
        return <fL>
end if
if minCount = countRight then
        return <fR>
end if
if minCount = countDoor then
        return <fD>
end if
```

The original model with all the not-in-common subassemblies removed can now be compared with the new model and the assembly sequence required to transform the former to the latter can be computed. This is done using the ‘GetReassemblyForest()’ function which utilises the following algorithm:

1. Subtract the array representing the disassembly forest from the array representing the original model
2. Subtract the array obtained in step 1 from the array representing the new model
3. Compute an assembly sequence for the subassemblies contained in the output array of step 2 using the AssemblySequencer() function, here denoted as *“reassembly forest”*

Let <GOrg> and <GNew> be arrays representing components from the orginal and the new 3D model respectively; <F> be an array representing the optimal disassembly forest returned by the GetDisassemblyForest() function; and <R> be an array representing the optimal reassembly forest:

Algorithm 8: *Function GetReassemblyForest( )*

```
Input: <GOrg>, <GNew>, <F>
do <R> = AssemblySequencer( <GNew> - <GOrg> + <F>)
return <R>
```

Finally, the “disassembly forest” and “reassembly forest” obtained above are concatenated to produce the desired *refabrication sequence* at the output.

3.3.4. Hardware controllers

The purpose of the hardware controller is to take in assembly/refabrication sequences and generate a set of motor control commands such that the hardware will carry out the appropriate assembly/refabrication operations and create the desired 3D structure. The controller also needs to take in the feedback signal from the hardware module which contains motors’ states in order to synchronize the execution of motor control commands. Note that the hardware controller incorporated an open-source library called “MindSqualls” [40], which acts as the interface between the C# .NET environment and the microcontroller of the Lego Mindstorm kit. This sub-module also incorporated an open-source program called “Motor Control” developed by [41], which implements algorithms that lead to more precise motor movements compared to those produced by the native LEGO Mindstorm firmware.

3.4. Hardware module

Regarding the design of gripper sub-module, there are two main types of grasping profiles, namely (1) Encompassing grasp: Where the gripper provides an enclosure to secure the object; and (2) Frictional grasp: Where the contact surfaces generate friction to secure the object. Since the purpose of our gripper is to securely hold a Lego Duplo/Jenga block during its transportation from stock side to assembly side, an encompassing grasp would not be suitable as it would prevent direct contact between the block and the structure, making it difficult to achieve a precise stack operation. Hence, as for the grasp selection, a frictional grasp with flat plates was chosen as the gripper mechanism to generate the required contact surface area.

Meanwhile, since servo motors are the only type of actuator available in a Lego Mindstorm kit, a mechanism which converts rotational motions into a "grasping motion" is needed. In this study, a rotational grasping design was chosen for the gripper and an attempt was made to "upgrade" the gripper by equipping it with the capability to undo the connection between two Lego Duplo blocks, using the exact same rotational grasping motion. In addition, rectangular patches of Egrips material [42] were glued to the surface of the flat plate to improve the coefficient of friction. An additional fixture was also added to the gripper to provide an attachment point for the lift drive. The realization of the final design of the gripper is shown in Figure 7(a).

Regarding the design of forward drive sub-module, the number of wheel types available were limited to two: cylindrical wheels or caterpillar tracks. It is desirable to have as much contact with the ground as possible to spread out the load. Since the forward drive module also has to carry both the lift drive and the gripper module, the caterpillar tracks coupled with one servo motor were thus chosen for our forward drive. Its realization is shown in Figure 7(b).

Given that the purpose of our lift drive is to enable vertical translation of the gripper, and that the only type of actuator available in a Lego Mindstorm kit is servo motors, one needs to design a mechanism which converts rotational motion into a linear motion. There are two main types of design for lift drive: (1) The crank-slider design and (2) The scissors design. In this study, it was found that the maximum vertical translation of the crank-slider design was insufficient by taking into account the height of the 3D structure that needs to be built. For this reason, the scissors design was chosen for our lift drive. The realization of the lift drive design is shown in Figure 7(c).

Insert Figure 7 approximately here

Finally, the support structure comprises an elevated runway with runway guards, as shown in the right side of Figure 3, with the stock and assembly space.

4. Validation

In this section, two tests were designed and conducted to demonstrate the feasibility of the proposed RPS. In the tests, the hardware and software were connected wirelessly using Bluetooth to make the RPS automated. The test models created were an N-by-N stack of single Jenga blocks and two wall designs as shown in Figure 8.

Insert Figure 8 approximately here

The first test consisted of two different structures to (1) assemble WallDesign1 (Figure 8(a)), and then to (2) refabricate WallDesign1 into WallDesign2 (Figure 8(b)). The first half of the test was completed successfully, with the occasional difficulty in connecting the top Lego bricks firmly to the Lego door due to placement inaccuracy. For the second half of the test, a correct refabrication sequence was computed, with two key emphases:

1. The system recognized that the three Lego blocks from WallDesign1 must be treated as a single entity during the disassembly process, since the Lego's connectors are assumed to be permanent connectors.
2. The system also recognized that even though WallDesign1 and WallDesign2 have both the three Lego blocks and the Jenga blocks on the bottom right corner as common parts, it must disassemble the Lego blocks so that the Jenga blocks on the bottom left corner can be disassembled, and can only leave the bottom right corner as is.

The RPS computed correctly the disassembly sequence {1, 2, 3} of WallDesign1 and the refabrication sequence {5, 2, 6} into WallDesign2 by taking into account the presence of the permanent connectors and the common part numbered 4.

However, despite the correct refabrication sequence being computed, manual intervention had to made to complete the second disassembly operation, in which the group of three Lego blocks must be lifted over the group of 4 Jenga blocks on the bottom left corner, as the robot's maximum lift height was insufficient.

In the second test, the command to refabricate could come at any time during the WallDesign1 assembly process. The system performed this test as successfully as the first test, by keeping track of the assembly process and generating the correct refabrication sequence regardless of what state the existing structure was in. A snapshot of the entire system after having completed the final test is shown in Figure 9.

Insert Figure 9 approximately here

5. Conclusion and future work

With the aim of increasing the design flexibility of current prefabrication methods, a system called RPS consisting of hardware and software modules was developed to demonstrate a new concept called "refabrication": the automatic disassembly of a prefabricated structure and its reconstruction according to a new design.

Two key algorithms within the software module were developed in this study for implementing the RPS. An algorithm was developed to automatically compare the old and new 3D models and identify all components which the two models do not have in common. Upon testing, this algorithm identified the correct differences between two non-trivial 3D models. In addition, an algorithm was developed to automatically compute the optimal refabrication sequence that would transform one model into another when given the differences between the two design models. This desired function was broken down into two sub-functions. First, the number of (dis)assembly operations required for the removal of any one single subassembly must be calculated. In order to achieve this, the algorithm incorporated a stripped-down version of the NDBG technique. Second, the smallest number of (dis)assembly operations required for the removal of all not-in-common subassemblies must be calculated. This was achieved by comparing three different alignment scenarios for the two models, calculating the total number of (dis)assembly operations required in each scenario, and finally picking the scenario with the smallest number of operations required. Upon testing, this algorithm also calculated the correct (dis)assembly sequence for the two 3D models mentioned with two notable successes: (1) The connectors between Lego blocks were assumed to be permanent connectors, and the system successfully recognized that connected blocks must therefore be treated as a single entity during the disassembly process; (2) The system also recognized that certain components which are common to both models must still be removed if such components are blocking the disassembly path of not-in-common components.

A hardware system was developed to demonstrate the working of the developed algorithms in real-time. This system performs all assembly operations with successful placement precision although some disassembly operations needed manual intervention due to insufficient maximum lift height. The scope of this study was, however, restricted to the refabrication of assemblies which employ only stacking operations (1D) and subassemblies of cuboid shapes. The results from this study could therefore be scaled-up and applied to a more realistic problem set by incorporating the full version of the NDBG technique, which accounts for 2D assembly operations and subassemblies of arbitrary polygons. Future investigations are warranted to extend the applicability of the proposed system as follows:

- Incorporating stability analysis algorithm to the assembly sequencer
- Adding a motion planner to ensure assembly operations are executed in optimal time
- Taking into account arbitrary placement of connectors as additional constraints on the assembly sequencing process

- Incorporating computer vision techniques to achieve better placement precision for the gripper arm

Acknowledgements

The first author would like to acknowledge the support of Dr. Andrew Gee and Mr. Konstantinos for providing permission to use the Lego Mindstorm kit and generous amount of Lego Duplo.

Reference

[1] G. Sparkman, A. Gibb, R. Neale, Standardization and preassembly: Adding value to construction projects, Volume 176 of Report from Construction Industry Research and Information Association, London, 1999, ISBN: 978-0-860-17498-1.

[2] J.M. Schoenborn, A case study approach to identifying the constraints and barriers to design innovation for modular construction, Master of Science Thesis in Architecture, Faculty of the Virginia Polytechnic institute and State University, 2012, Retrieved from http://hdl.handle.net/10919/32397 (accessed at 14 August 2017)

[3] R. Sacks, C.M. Eastman, G. Lee, Process model perspectives on management and engineering procedures in the precast/prestressed concrete industry, Journal of Construction Engineering and Management ASCE 130 (2) (2004) 206–215. DOI:https://doi.org/10.1061/(ASCE)0733-9364(2004)130:2(206)

[4] L. Jaillon, C.S. Poon, Y.H. Chiang, Quantifying the waste reduction potential of using prefabrication in building construction in Hong Kong, Waste Management 29 (1) (2009) 309–320. DOI:https://doi.org/10.1016/j.wasman.2008.02.015

[5] V.W.Y.Tam, C.M. Tam, S.X. Zeng, W.C.Y Ng, Towards adoption of prefabrication in construction, Building and Environment. 42 (2007) 3642-3654. DOI:https://doi.org/10.1016/j.buildenv.2006.10.003

[6] E. Wright, Fast build nation: Richard Ogden on offsite construction. 2010. Retrieved from http://www.building.co.uk/fast-build-nation-richard-ogden-on-offsite-construction/3160607.article, (accessed at 14 August 2017)

[7] D.A. Steinhardt, K. Manley, W. Miller, Reshaping Housing – The role of prefabricated systems. In Proceedings of World Sustainable Building Conference (SB14), Barcelona, (2014), pp. 30–36. Retrieved from <http://eprints.qut.edu.au/78400/> 14 August 2017.

[8] C. Balaguer, M. Abderrahim (Subcontractors), Trends in Robotics and Automation in Construction, in: C. Balaguer and M. Abderrahim (Editors) ,Robotics and Automation in Construction, In-Tech, Croatia, 2008, pp. 1–20 ISBN: 978-953-7619-13-8, DOI: 10.5772/5865.

[9] T. Bock (Subcontractor), Construction Automation and Robotics, in: C. Balaguer and M. Abderrahim (Editors) ,Robotics and Automation in Construction, In-Tech, Croatia, 2008, pp. 21-42 ISBN: 978-953-7619-13-8, DOI: 10.5772/5861.

[10] W. Ibbs, Construction Change: Likelihood, Severity, and Impact on Productivity. J. Legal Affairs and Dispute Resolution in Engineering and Construction, 4(3) (2012), 67-73 DOI:https://doi.org/10.1061/(ASCE)LA.1943-4170.0000089.

[11] C. Kibert, Sustainable Construction: Green Building Design and Delivery, Wiley, Hoboken, New Jersey, 2016 ISBN: 978-1-119-05517-4.

[12] A. Dubois and L. E. Gadde, The construction industry as a loosely coupled system: implications for productivity and innovation. Construction Management & Economics, 20(7) (2002) 621-631. DOI:https://doi.org/10.1080/01446190210163543.

[13] S. P. Low and S. H. Mok, The application of JIT philosophy to construction: a case study in site layout, Construction Management and Economics, 17 (1999), 657-668. DOI:https://doi.org/10.1080/014461999371268.

[14] D. Gann, Construction as a manufacturing process? Similarities and differences between industrialized housing and car production in Japan, Construction Management and Economics, 14 (1996), 437-450. DOI:https://doi.org/10.1080/014461996373304.

[15] K. Jung, B. Chu, D. Hong, Robot-based construction automation: An application to steel beam assembly (Part II), Automation in Construction, 32 (2013) 62-79. DOI:https://doi.org/10.1016/j.autcon.2012.12.011.

[16] M. J. Skibniewski, Robotics in civil engineering, Computational Mechanics Publications, 1988. ISBN: 978-0-905-45177-0

[17] T. Bock, Construction robotics, Autonomous Robot 22 (2007) 201-209. DOI:https://doi.org/10.1007/s10514-006-9008-5.

[18] E. Gambao, C. Balaguer, F. Gebhart, Robot assembly system for computer-integrated construction, Automation in Construction, 9(5) (2000) 479–487. DOI:https://doi.org/10.1016/S0926-5805(00)00059-5.

[19] C. Balaguer, EU FutureHome project results. In Proceedings of the 20th International Symposium on Robotics and Automation in Construction (ISARC'03), Eindhoven, Netherland, (2003), pp 49-54. Retrieved from <http://www.iaarc.org/publications/proceedings_of_the_20th_isarc/eu_futurehome_project_results.html> (accessed at 14 August 2017)

[20] R. Wing, B. Atkin, FutureHome – A Prototype for Factory Housing, In Proceedings of the 19th International Symposium on Robotics and Automation in Construction (ISARC'02), Washington, U.S.A, (2002), pp 173-179. Retrieved from http://www.iaarc.org/publications/proceedings_of_the_19th_isarc/futurehomea_prototype_for_factory_housing.html> (accessed at 14 August 2017)

[21] T. Bock, The Integrated Project ManuBuild of the EU, In Proceedings of the 23rd International Symposium on Robotics and Automation in Construction (ISARC'06), Tokyo, Japan, (2006), pp

361-364. Retrieved from <http://www.iaarc.org/publications/proceedings_of_the_23rd_isarc/the_integrated_project_manubuild_of_the_eu.html> (accessed at 14 August 2017)

[22] J. Sklar, Robots lay three times as many bricks as construction workers, MIT Technology Review, 2017. Retrieved from <https://www.technologyreview.com/s/540916/robots-lay-three-times-as-many-bricks-as-construction-workers/> (accessed at 14 August 2017).

[23] B. Khoshnevis, Automated construction by contour crafting – related robotics and information technologies, Automation in Construction, 13 (2004), 5-19. DOI:https://doi.org/10.1016/j.autcon.2003.08.012.

[24] Enrico Dini, D_Shape, Retrieved from http://www.d-shape.com. (accessed at 14 August 2017).

[25] H.S. Choi, C.S. Han, K.Y. Lee, S.H. Lee, Development of hybrid robot for construction works with pneumatic actuator. Automation in Construction, 14(4) (2005) 452–459. DOI:https://doi.org/10.1016/j.autcon.2004.09.008.

[26] J.P. Sousa, C.G. Palop, E. Moreira, A.M. Pinto, J. Lima, P. Costa, G. Veiga, A. P. Moreira, The SPIDERobot: A Cable-Robot System for On-site Construction in Architecture, Robotic Fabrication in Architecture, Art and Design (2016) pp 230-239. DOI:https://doi.org/10.1007/978-3-319-26378-6_17

[27] B. Chu, K. Jung, M.T. Lim, D. Hong, Robot-based construction automation: An application to steel beam assembly (Part I), Automation in Construction, 32 (2013) 46-61. DOI:https://doi.org/10.1016/j.autcon.2012.12.016

[28] K. Dörfler, T. Sandy, M. Giftthaler, F. Gramazio, M. Kohler, J. Buchli, Mobile robotic brickwork, Robotic fabrication in Architecture, art and design (2016) 204-217. DOI:https://doi.org/10.1007/978-3-319-26378-6_15

[29] A. Mirjan, F. Gramazio, M. Kohler, Building with flying robots, in FABRICATE: negotiating design and making (2014) 266-271. ISBN: 9783856763312

[30] L. S. Homem de Mello, A. C. Sanderson, A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences, IEEE Transactions on Robotics and Automation, 7(2) (1991) 228-240. DOI: https://doi.org/10.1109/70.75905

[31] D.T. Le, J. Cortes, T. Simeon, A path planning approach to (dis)assembly sequencing. In IEEE International Conference on Automation Science and Engineering, Bangalore, India, (2009) pp 286–291. DOI: https://doi.org/10.1109/COASE.2009.5234177

[32] S. Sujay, R. Ian, M. A. Nancy, Disassembly sequencing using a motion planning approach, Proceedings ICRA, IEEE International Conference on Robotics and Automation, In Proceedings of IEEE International Conference of Robotics Automation Seoul, Korea, May (2001), pp. 1475-1480. DOI: https://doi.org/10.1109/ROBOT.2001.932818

[33] R. H. Wilson, On Geometric Assembly Planning. PhD thesis, Stanford Univ., Stanford Technical Report STAN-CS-92-1416 (1992). Retrieved from <http://dl.acm.org/citation.cfm?id=143786> (accessed at 14 August 2017).

[34] H. Mosemann, F. Rohrdanz, F.M. Wahl, Stability analysis of assemblies considering friction. IEEE Transactions on Robotics and Automation, 13(6) (1997) 805–813. DOI: https://doi.org/10.1109/70.650159

[35] S. Rakshit, S. Akella, The Influence of Motion Path and Assembly Sequence on the Stability of Assemblies, IEEE Transactions on Automation Science and Engineering , 12(2) (2015) 615-627. DOI: https://doi.org/10.1109/TASE.2014.2345569

[36] R.H. Wilson, J.C. Latombe, Geometric Reasoning About Mechanical Assembly, Artificial Intelligence 71(2) (1994) 371-396. DOI: https://doi.org/10.1016/0004-3702(94)90048-5

[37] D. Baraff, R. Mattikalli, P. Khosla, Minimal Fixturing of Frictionless Assemblies: Complexity and Algorithms, Algorithmica 19 (1997) 4-39. DOI: https://doi.org/10.1007/PL00014419

[38] D.E. Stewart, J.C. Trinkle, An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction. International Journal of Numerical Methods Engineering, 39

(1996) 2673-2691. DOI: https:// 10.1002/(SICI)1097-0207(19960815)39:15<2673::AID-NME972>3.0.CO;2-I

[39] Mindstorm, Lego Inc., Retrieved from <http://www.lego.com/en-gb/mindstorms> (accessed at 14 August 2017).

[40] Mindsqualls, Lego Inc., Retrieved from <http://www.mindsqualls.net/Introduction.aspx> (accessed at 14 August 2017).

[41] RWTH – Mindstorms NXT Toolbox: Motor Control, RWTH Aachen University, Retrieved from <http://www.mindstorms.rwth-aachen.de/trac/wiki/MotorControl> (accessed at 14 August 2017).

[42] Erips, Egrips Inc., Retrieved from <https://egrips.com/> (accessed at 14 August 2017).

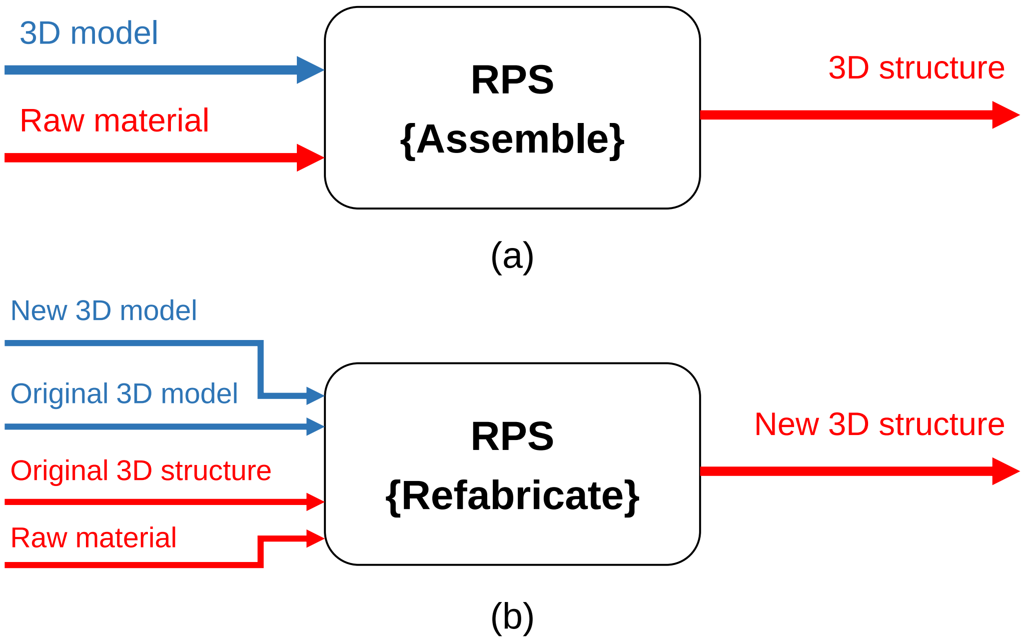


Figure 1. The two functions of RPS: (a) Assemble and (b) Refabricate

*Note: The blue and red arrows represent digital and physical quantities, respectively.

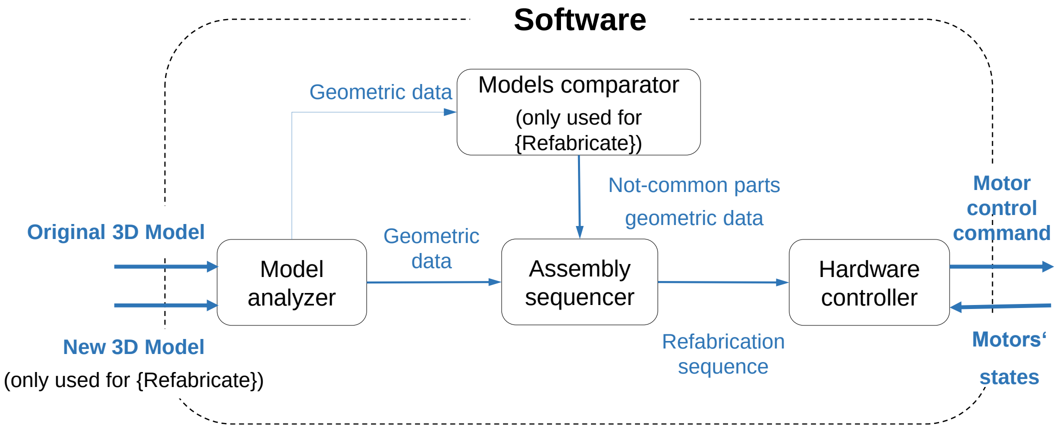


Figure 2. Design of the RPS software module

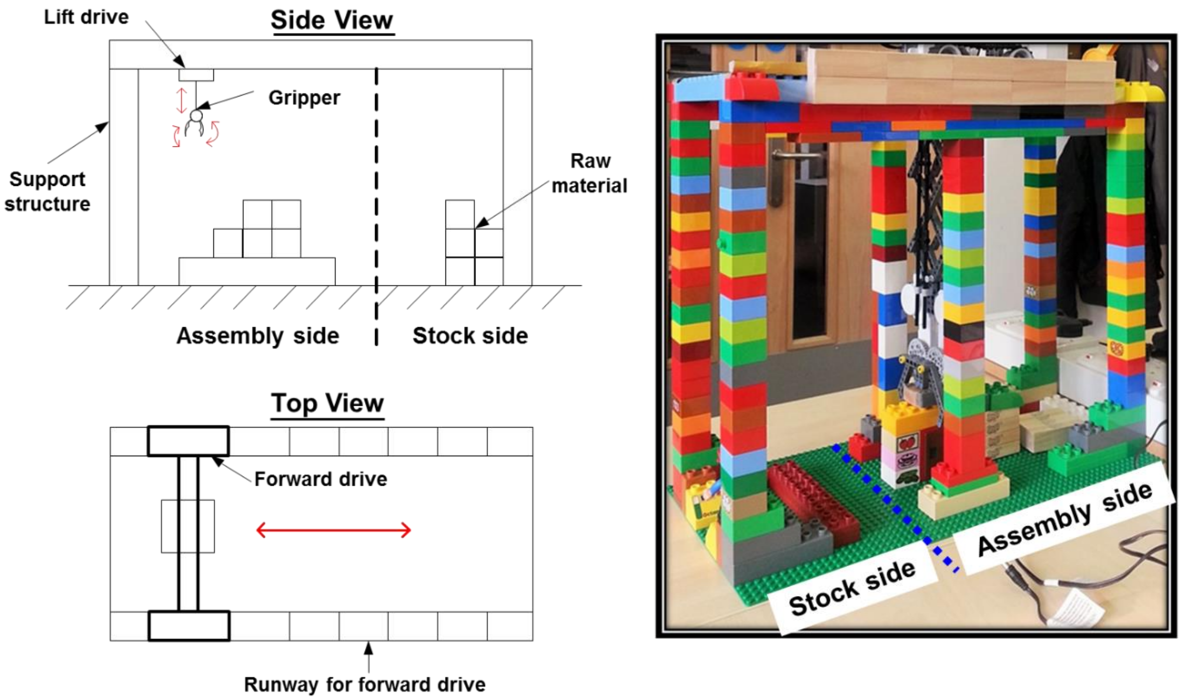


Figure 3. Design of the RPS hardware module

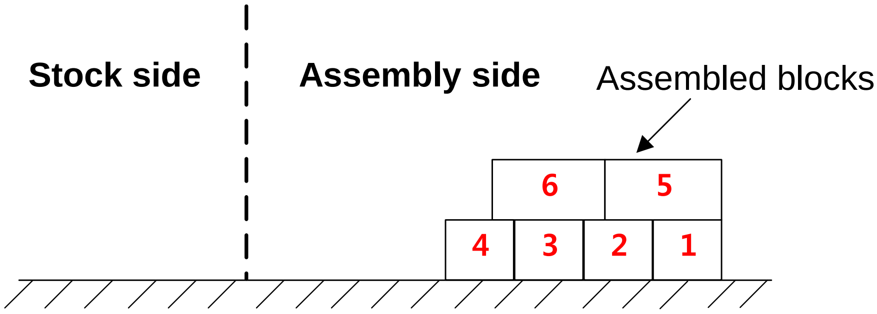
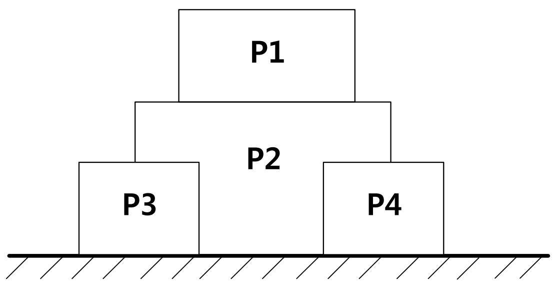
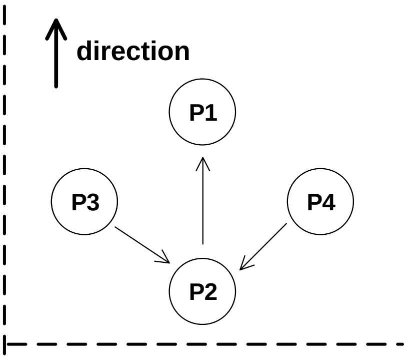


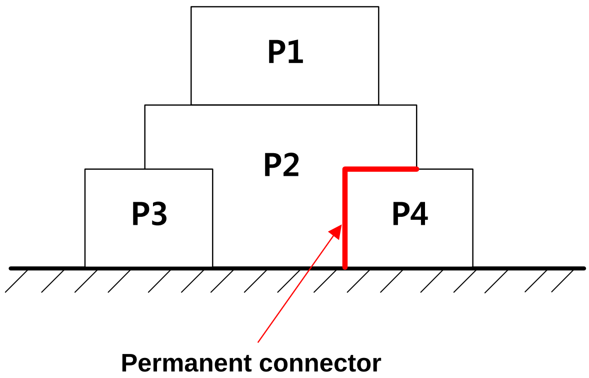
Figure 4. An example assembly sequence calculated by the assembly sequencer

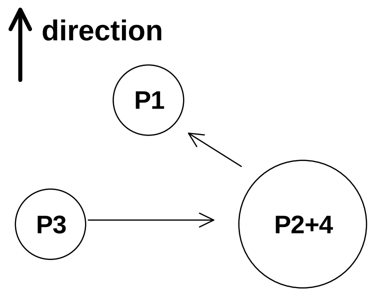




	P1	P2	P3	P4
P1	0	0	0	0
P2	1	0	0	0
P3	0	1	0	0
P4	0	1	0	0

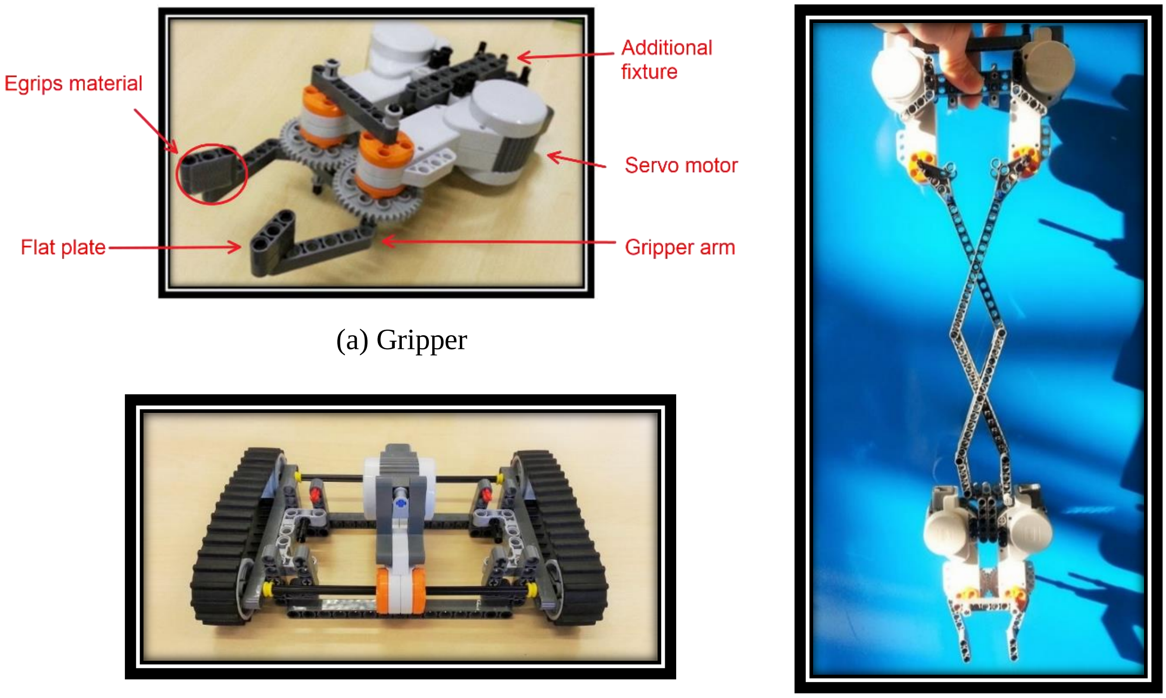
Figure 5. Example of a DBG & its matrix representation





	P1	P2 + P4	P3
P1	0	0	0
P2 + P4	1	0	0
P3	0	1	0

Figure 6. Example of a truncated DBG & its matrix representation



(a) Gripper

(c) Forward drive

(b) Lift drive (coupled with the "gripper" sub-module)

Figure 7. Hardware submodules: (a) gripper, (b) lift drive, and (c) forward drive.

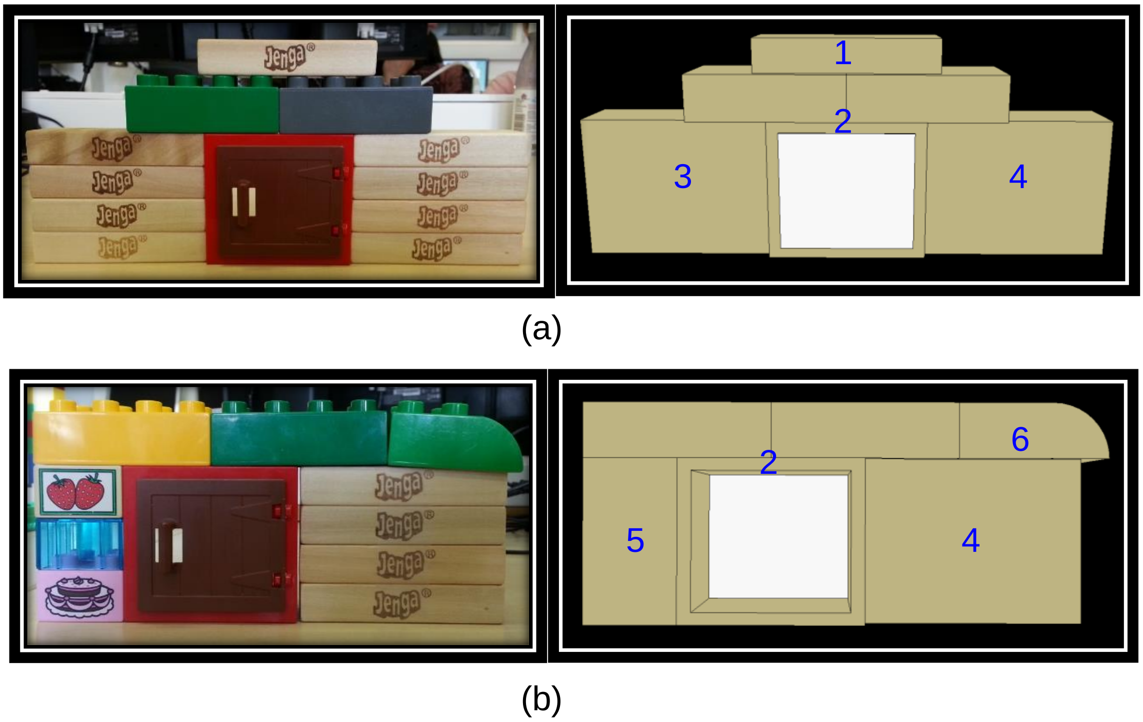


Figure 8. Real-life and digital versions of test model: (a) WallDesign1 and (b) WallDesign2

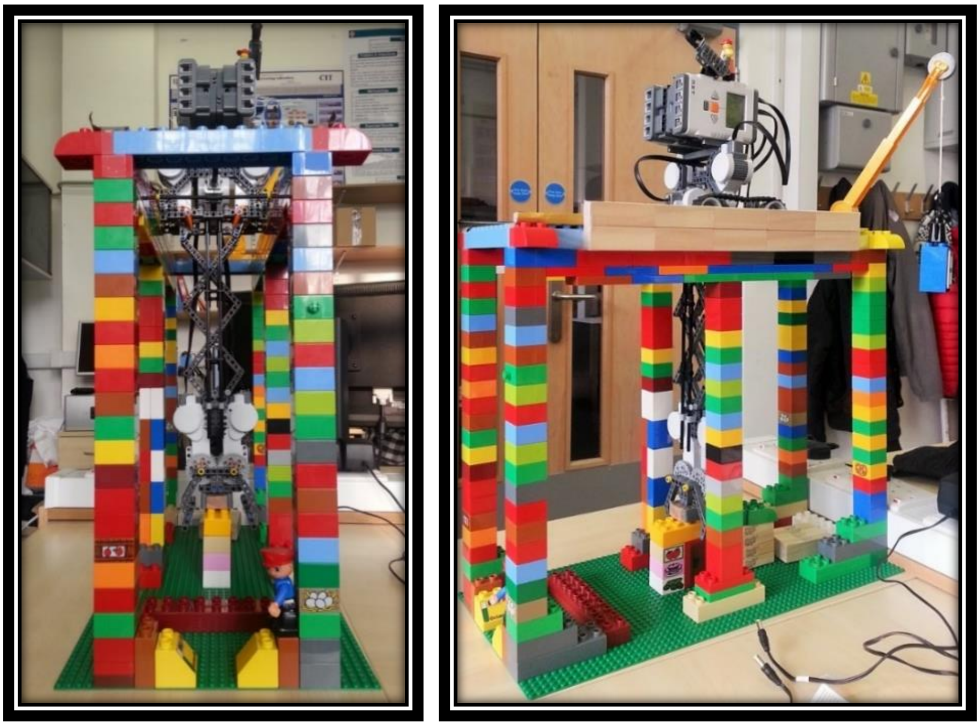


Figure 9. Snap-shot of the entire system after completing the final test