Brick Yourself within 3 Minutes

Guyue Zhou[1]✉, Liyi Luo[1,2], Hao Xu[3]✉, Xinliang Zhang[1], Haole Guo[1] and Hao Zhao[4]

***Abstract*— This paper presents an intelligent machine which can automatically convert the captured portrait into a physical gadget made up of LEGO bricks. On the contrary to synthesising a 2D image or a virtual 3D object, generating physical 3D assembly object needs to take physical properties and assembly process into consideration, leading to more challenges. To generate brick models for arbitrary portraits, we formulate the transformation between the attribute space (extracted from 2D images) and the brick model space as a constraint integer programming problem which can be solved with a heuristic search method. Furthermore, as the bricks are physically scattered, we propose an algorithm to generate corresponding assembly instructions for customized figure-featured-bricks to facilitate users' assembly. Meanwhile, we deploy the proposed algorithms on an automatic machine which integrates a camera, a printer, a laptop, and a brick operation unit. Finally, the generated brick models and assembly instructions are evaluated by a large number of users. It is worth noting that the whole system works as an intelligent vending machine, producing a 150-brick-model within 3 minutes.**

I. Introduction

With the rapid development of 3D printing technology, mass customization, which was one of the forefront innovations in manufacturing, have been gaining popularity even for daily life. Compared with customizing a single physical object, to manufacture product-level assemblies is obviously more challenging, because the transition from user requirements to product design is non-trivial and assembling multiple parts into the final object involves a complex procedure. Nowadays, consumer products can hardly be customized because of the relatively high cost of manufacturing non-standard parts. An alternative way to achieve product-level customization is to assemble standard parts in diversified ways. For instance, toy building bricks are seen everywhere in our daily lives and hence are perfect candidates of standard parts for product customization.

In particular, figure-featured-bricks, e.g., LEGO Brick-Headz series, are increasingly popular as toys, gifts, decorations and even artworks. Unfortunately, the lack of neither the design expertise nor the flexible supply chain makes it still difficult for laymen to customize figure-featured-bricks. The situation becomes worse when standard bricks are not self-serving, meaning that we need to produce different customized assembly instructions for each of different designs. To solve the aforementioned problems, we demonstrate an intelligent robotic system that enables laymen to customize their own figure-featured-bricks by simply taking a selfie to be sent to the proposed vending machine.

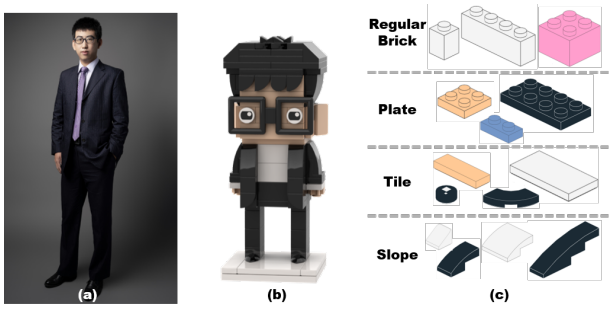

Fig. 1. We develop a fully automatic system that converts a captured portrait (a) into a brick model that resembles the portrait and can be physically assembled (b). The generated model is made up with the most common LEGO bricks (c), such as regular bricks, plates, tiles, slopes, etc.

To bridge the gap between casual user requirements and their expected professional design, we present an easy-to-use system that handles all stages of the figure-featured-brick customization, from single-portrait input, design generation, human-in-the-loop interaction, to final fabrication. In our system, experts define templates and basic assembly rules that have physical guarantees. A series of deep neural networks (DNN) and heuristic search methods are employed to generate brick model proposals based on a single photo from causal end users. End users can fine-tune these design proposals in an interactive interface and the confirmed brick design can be physically delivered on site. While our machine is picking selected physical bricks, our system automatically generate the corresponding assembly instructions to facilitate user's assembly. We accomplish the complete system by contributing the following aspects.

- An end-to-end framework that supports automatic figure-featured-bricks customization: from design to fabrication.
- A novel algorithm that generates the brick model for a given portrait, and furthermore generates the assembly instruction for a given brick model.
- A fine-tune scheme that iteratively improves the automatic design performance on the basis of user feedback.
- A series of experiments that verify the design through fabrication of customized figure-featured-bricks from an automatic machine.

II. Related Work

Design for manufacturability is an important problem in engineering design [1] [2] [3] [4]. More recently, advances in

[1]Institute for AI Industry Research (AIR), Tsinghua University, China {zhouguyue, zhangxinliang, guohaole}@air.tsinghua.edu.cn
[2]McGill University, Canada lyl@gmail.com
[3]Qianzhi Technology, China hao.xu.chn@gmail.com
[4]Intel Labs and Peking University, China hao.zhao@intel.com

manufacturing have garnered a lot of interesting fabrication-oriented design systems that bring down the design barrier for casual users. A wide range of domain specific design tools have been proposed in previous work, including systems for push toys [5], clothes [6], linkage-based characters [7], model airplanes [8], robots [9] [10] and carpentry [11]. Specific to the LEGO design and fabrication problem, related work will be discussed in sequence of system architecture, brick model and assembly instruction respectively.

A. System Design

Although there does not exist any exact system which can convert the captured portrait into a physical gadget made up of LEGO bricks, some inspirations can still be obtained from similar works. For example, some work has been done to generate virtual character from a single image, using a skinned vertex-based model [12], a silhouette-based model [13], or a Hierarchical Mesh Deformation (HMD) framework [14]. Meanwhile, the mechanical system design [15] [16] of chip mounters demonstrates a high-speed execution of automatic pick and place tasks for tiny parts.

B. LEGO Design

With a given target shape, Gower et al. [17] firstly formulated the automatic construction of LEGO models using regular bricks (refer to Fig.1) as a combinatorial optimization framework to maximize a goodness measure for LEGO structures. Some follow-up works solve this problem using different methods, include evolutionary algorithms [18], beam search methods [19], graph-based algorithms [20], multi-phase approaches [21], and genetic algorithms [22].

Considering more LEGO parts other than regular bricks, complicated structure brings mechanics, assembly and artistic problems. For example, LEGO assemblies with oriented thin plates was computed in [23] and silhouette fitting [24] was used to improve LEGO constructions. Moreover, stability analysis was solved via a max-flow network [25] based on a two-phase approach. The order of brick colors during assemblies was taken care with a comprehensive method [26] for buildable LEGO structures in larger scales. The visual quality was optimized in Pixel2Brick [27] which constructed computed brick sculptures from pixel arts.

C. Instruction Design

The two primary tasks in designing assembly instructions are planning and presentation [28]. The assembly planning, which is a classic problem in robotics [29] [30], aims to choose a sequence of assembly operations that will be friendly for users to understand and follow. The assembly presentation, which is essentially a visualization problem [31] [32], aims to automatically produce diagrams or other digital contents that are easy for humans to understand. With the development of new technology, traditional paper-made instructions evolve to e-versions embedded in Computer Aided Design (CAD) [33], Virtual Reality (VR) [34], and Augmented Reality (AR) [35] systems. Specific to the field of LEGO instruction design, there exist quite a few interesting attempts: an assembly guide can be generated in real time with a RGBD camera input [36], a component-driven instruction can be designed based on a component-based LEGO sculpture model [37], and an assembly instruction for multiple robots can be planned to finish collaborative LEGO construction [38].

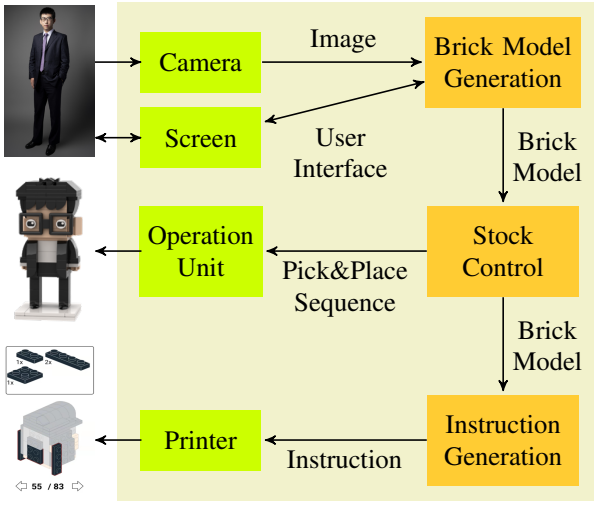

Fig. 2. The overview of system design: external input/output (left) and internal modules (right) including hardware (green blocks) and software (orange blocks).

III. System Architecture

Figure 2 illustrates our proposed system framework, which converts the captured portrait into a physical gadget made up of LEGO bricks, with the following assumptions.

- The captured portrait covers the frontal full-body.
- The homogeneous bricks (with same shape and same color) are preset in a dedicated storage bin – the minimum unit for machine's loading and unloading.
- The bricks are interspersed and are required to be manually assembled following the assembly instruction.
- The assembly instruction is e-version and can be accessed by scanning a printed QR code.

In the remaining of this section, we introduce hardware and stock control modules. We will discuss our method of brick model generation and instruction generation in Section IV and Section V, respectively.

A. Hardware

As shown in Fig.3(a), the hardware system is composed of a RGB camera for capturing portraits, a wireless printer for printing QR codes (link to the e-version assembly instructions), a laptop for user interaction, and an operation unit for brick storage and manipulation. The total dimension is 124cm $\times$ 84cm $\times$ 130cm (laptop excluded). The operation unit, which is the core module of the hardware system, is made up of a X-Z table for feeding, a Y-Z table for pushing, a friction wheel, a multi-layer storage bins, a pneumatic device with ducts, a servo-driven fence and other mechanisms.

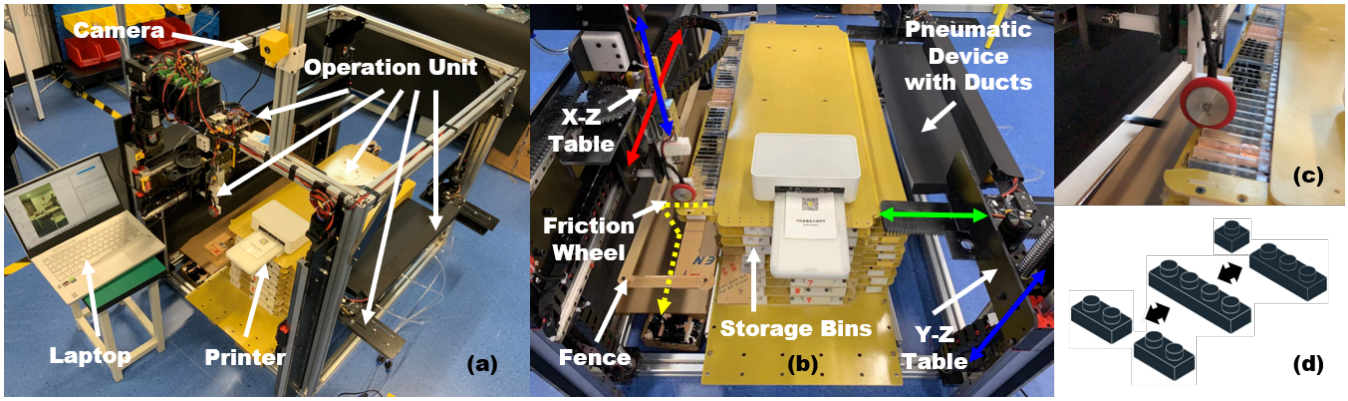

Fig. 3. Instances of hardware: (a) hardware system overview, (b) breakdowns inside the operation unit and a brick's trajectory (yellow dashed) from the storage bin to the output box, (c), the instant of the friction wheel touching a black brick, and (d) alternative brick combinations for a 1×4 plate.

Before designing the storage bins, we build a brick library including 998 colored bricks which are frequently used in figure-featured-brick design. To adapt bricks of different heights and widths, we design 8 types of storage bin. Our operation unit contains 9 layers of 385 storage bins, which are able to store 6719 bricks at full load and can be easily extended to cover the entire brick library with more layers inside the current machine. Furthermore, each layer, which layouts 29 to 50 storage bins, can be pushed by the Y-Z table or pulled by a drag spring. In this case, the friction wheel mounted on the X-Z table can access to every storage bin.

When a specific brick is required, our system will first push the corresponding layer and move the friction wheel right on the corresponding storage bin. Then, we adjust the fence, which is a part of a parallelogram structure driven by a servo. When the fence is close to pushed storage bins, pneumatic device can blow bricks toward the friction wheel side through ducts. When the fence is away from pushed storage bins as shown in Fig.3(c), the brick is driven by the rotating friction wheel to the fence, and is next landed on a slope to the output box following the trajectory in Fig.3(b). In practice, we know the complete sequence of required bricks in advance, so the operation unit will execute layer by layer continuously.

B. Stock Control

For each type of brick b, its stock volume $S(b)$ can be known from on-board statics and its expected consumption volume per gadget $C(b)$ can be known from big-data statics. The stock control module's task is to maximize the expected gadget volume that our machine can produce.

Considering the real-time computing requirement, we simplify the task by assuming consumption expectations are independent among different types of bricks. A two-step greedy strategy can be applied on a generated brick model. Firstly, we should select all flexible brick combinations, which can be equivalently replaced by other bricks. Secondly, for each flexible brick combination $\mathcal{C}$, we should enumerate every possible alternative $\mathcal{B} \in \mathcal{A}(\mathcal{C})$ and select the one maximizing the stock function $\mathcal{F}(\mathcal{B}) = \min_{b\in\mathcal{B}} S(b)/C(b)$ where b represents the brick element in B.

To quickly find the alternative for a certain brick combination, we can prepare a look-up table among different bricks in advance. Besides typical alternative relationships between single big part and several small parts as shown in Fig.3(d), alternative relationships of non-appearance bricks can come from different colors. For a specific brick model, do notice that a brick's alternative should be double-checked in terms of structure and assembly using the methods described in Section IV and Section V.

IV. Brick Model Generation

Fig.4 shows our 3-step pipeline of the brick model generation module. Firstly, the EfficientNetV2 [39] is used to convert the input portrait to a 92×1 vector in the attribute space. Secondly, an uncolored brick model is transformed from the attribute vector by solving a integer programming under collision and assemblability constraints. Thirdly, the Attribute-Mask R-CNN [40] is employed to assign colors for the brick model. Here follows the detailed descriptions.

A. Map Portraits to Attributes

Define the attribute vector $V_a = \{V_h; V_d\}$ where V_h is a 40×1 head vector to describe faces, hairs and emotions, V_d is a 52×1 dress vector to describe the dressings, and each element of V_a should be a float number in $[0, 1]$ to represent the confidence value for the particular attribute. To achieve a image-to-attribute mapping, we use the EfficientNetV2 as the backbone and fine-tune the efficientnet-b7 pre-trained model. V_h can be obtained with a network trained with the Large-scale CelebFaces Attributes (CelelbA) [41] Dataset. V_d can be decomposed into the 2-dimensional category component and the 50-dimensional attribute component. Two networks are trained using the DeepFashion: Category and Attribute Prediction Dataset [42] with all 50 categories and the most common 50 clothing attributes to generate V_d.

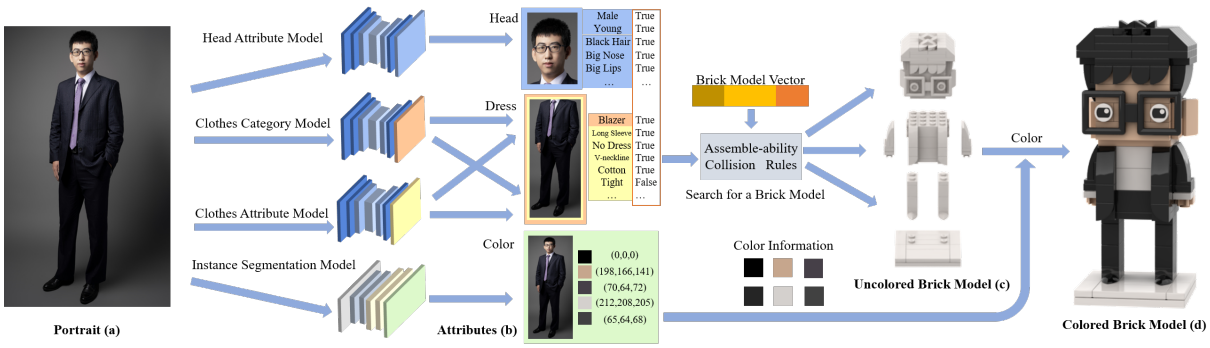

Fig. 4. Overview of our brick model generation pipeline. Given the input portrait (a), we use several deep neural networks to extract the attributes (b) to describe the human appearance shown in the image. Built on these attributes, we generate the corresponding uncolored brick model (c) by iteratively searching for brick component with the coordinate descent algorithm. Finally, we assign color to every brick to get the final brick model (d).

B. Transform Attributes into Brick Models

Define the brick vector V_b which is a 20×1 vector to describe brick model options for beards, coiffures, face shapes, glasses, waists, sleeves, trousers, shoes and other figure features. Each element in V_b is a positive integer which stands for an index of preset brick model options. Meanwhile, every brick model option is prepared by professional designers and is manually labeled with the same attributes according to the attribute vector. In this case, the essential problem, searching for the best matching brick model under collision and assemblablility constraints, turns out to be a constraint integer programming [43].

$$V_b^* = \arg\min_{V_b}(|V_a, \mathcal{L}(V_b)|_2 | V_b \in C_B \cap A_B) \qquad (1)$$

where $\mathcal{L}$ is the mapping function from the brick vector to the attribute vector with known labels, C_B and A_B stand for the brick spaces satisfying collision and assemblability rules respectively.

To minimize the Euclidean distance in the attribute space in 1, we apply a coordinate descent algorithm [44]. Based on the expertise from professional designers, we follow a heuristic search strategy starting from the head brick model, following with the upper body brick model, and fitting the rest brick models lastly.

C. Color Brick Models from Portraits

Define the color vector $V_c(b)$, which is a $k \times 1$ vector and k varies from different brick model b, to represent color information. Each element of V_c stands for an index of discrete brick colors. A two-stage method is used for color calculation. Firstly, we use the Attribute-Mask R-CNN as the backbone and the ResNet-101 FPN as the pre-trained model achieving pixel-level parsing for the portrait. Secondly, we calculate major colors of k regions of interest. The major colors come from the color-consistent connected areas with related semantics.

V. Instruction Generation

After generating the brick model, we present our system to automatically generate the assembly instruction, which will be used to guide users to assembly the model in an easy and friendly way. The overall pipeline of our assembly instruction generation framework is inspired from the method in [28], where we adopt and integrate their method to fit in our system.

Below, we firstly introduce our method of modeling the problem as a discrete optimization and then present our search-based method to efficiently solve the optimization problem.

A. Modeling an Assembly Instruction

The major objective of a user-friendly assembly instruction is to compute an assembly sequence of all brick pieces in the model as well as grouping them into steps. Based on this principle, we model the assembly instruction generation as an optimization, where we aim to find a proper sequence of all brick pieces in the generated model, as guided by our objective functions. The objective function mainly stands for two physical meanings in general: 1) the brick model can be physically assembled according to our generated assembly sequence, and 2) we strive for a better assembly experience for users. Below, we elaborate the requirements from the two aspects.

a) Modeling Assemblability: Assemblability is a fundamental requirement for an assembly instruction, which means the users can complete the final brick model according to the assembly sequence as indicated by the assembly instruction, without the need to remove assembled bricks or add additional support. In this work, we model the fundamental requirements for assembly instruction as hard constraints, which include two terms. First, the bricks shown in each step should be able to be snapped with existing bricks without removing existing bricks. Second, the bricks shown in each step should be able to connect with existing bricks; or in other words, do not "float" in the air. We compute the removal direction of the bricks based on the blocking relationship as presented in [28], and compute the connection between bricks based on the proximity of the connection point of adjacent bricks.

b) Modeling Assembly Experience: Built on the hard constraints discussed above, we strive to make the instruction

easy to be understood by general users. Here, we consider the following objectives:

- Firstly, we maximize the visibility of all bricks shown in each step. Because an assembly instruction can only present 3D objects in a 2D space, we need to carefully select the bricks to be assembled in one step, as well as to select the viewing angle of the camera. In this work, we measure the visibility of the bricks shown in one step by minimizing the blocked area of the newly-added bricks shown on the screen.
- Secondly, we maximize the semantic meaning of the bricks shown in each step, for example, instead of adding brick to both head and arms in one step, the users prefer to assembly arms after completing the head. We preserve the semantic meaning by leveraging the semantic grouping information produced in the model generation procedure, where we minimize the total number of steps in which the bricks cover more than one semantic group.
- Thirdly, we improve the consistency between every two consecutive steps, which facilitate the users to quickly locate the position to place new bricks. We quantify the step consistency by computing the distance between every two consecutive steps and aim the minimize the total sum of all consecutive steps.

B. Searching for an Assembly Instruction

In our problem modeling as presented in Section V-A, we are facing a combinatorial optimization problem with hard constraints and soft objectives, where the decision variables are the assembly sequence and the viewing angle of the camera in each step. The solution space of the optimization grows exponentially with the problem size, yielding an immense discrete search space.

In this section, We present our method to solve such a challenging combinatorial problem, where we present a search procedure that computes the brick steps one by one, as guided by our objective introduced in Section V-B. The general pipeline of our method can be viewed as a disassembling procedure, where we iteratively find removable bricks from the brick model, group bricks as steps until all bricks are removed. By doing so, the final assembly order can be obtained by computing the reverse order of the disassembly order.

Our searching method (Algorithm 1) can be divide into the following two phases. In the first phase, given the generated model $\mathcal{M}$ with semantic groups $\mathcal{G}$, we compute the assembly sequence inside each group. For each group, our method first finds all bricks that can be removed from the group, denoted by $\mathcal{B}$. All bricks in $\mathcal{B}$ must satisfy 1) each brick can be removed without colliding with other bricks, and 2) the remaining bricks do not collapse or be separated after removing the brick. After that, we enumerate all subsets of bricks in $\mathcal{B}$, and select the subset that maximizes the soft constraints introduced in Section V-A. We repeat this process until no brick remains in the group to get the feasible disassembly sequence for the group. In the second phase, we compute the assembly sequence of all groups. In our implementation, we treat each group as a large brick and compute the sequence based on the same procedure as to compute the sequence for individual bricks. At last, we compute the viewing angle of the camera for each step, where we first set four fixed camera viewing angles, and evaluate which one gives maximum brick visibility.

Algorithm 1 Procedure to Generate Assembly Instruction

Input: $\mathcal{M} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$
Output: $\mathcal{R} = [[b_1, b_2, ..b_{m1}], [b_{m1+1}, b_{m1+2}, ..b_{m1+m2}], ...]$

$\mathcal{R} = []$
while $\mathcal{M}$ is not $\emptyset$ **do**
  for each group g_i in $\mathcal{M}$ **do**
    if g_i is removable with $\mathcal{M}$ and $\mathcal{M} \setminus g_i$ is self-connected **then**
      temp = []
      while g_i is not $\emptyset$ **do**
        $\mathcal{B}$ = get_all_removable_bricks(g_i)
        $[\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_m]$ = group_bricks($\mathcal{B}$)
        $\mathcal{B}_{\text{best}}$ = group_ranking($[\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_m]$)
        temp.append($\mathcal{B}_{\text{best}}$)
        g_i.pop($\mathcal{B}_{\text{best}}$)
      end while
      $\mathcal{R}$.append(reverse(temp))
      $\mathcal{M}$.pop(g_i)
      break
    end if
  end for
end while
return reverse($\mathcal{R}$)

VI. Experimental Results

We set up two separated experiments: (a) a user study collecting feedbacks from real customers to verify the performance of brick model generation and instruction generation modules, and (b) a system demonstration to show the capability of converting the captured portrait into a physical gadget made up of LEGO bricks within 3 minutes.

A. User Study

As designs of brick models and assembly instructions can hardly be evaluated with an objective function, we provide brick customization service and collect feedbacks from real customers to directly verify the performance of brick model generation and instruction generation modules.

Before launching at various of e-commerce platforms, we tested our proposed brick customization service with two key opinion leaders (KOLs) – Sinovation Ventures (a venture capital company led by Dr. Kai-Fu Lee), and the Fan Club of Zhan Xiao (a popular actor in China). We obtained positive feedbacks and understood that textures printed on bricks would be the finishing touch as shown in Fig.5(a).

To generate labeled data satisfying requirements from customers, we employed designers to extend our design

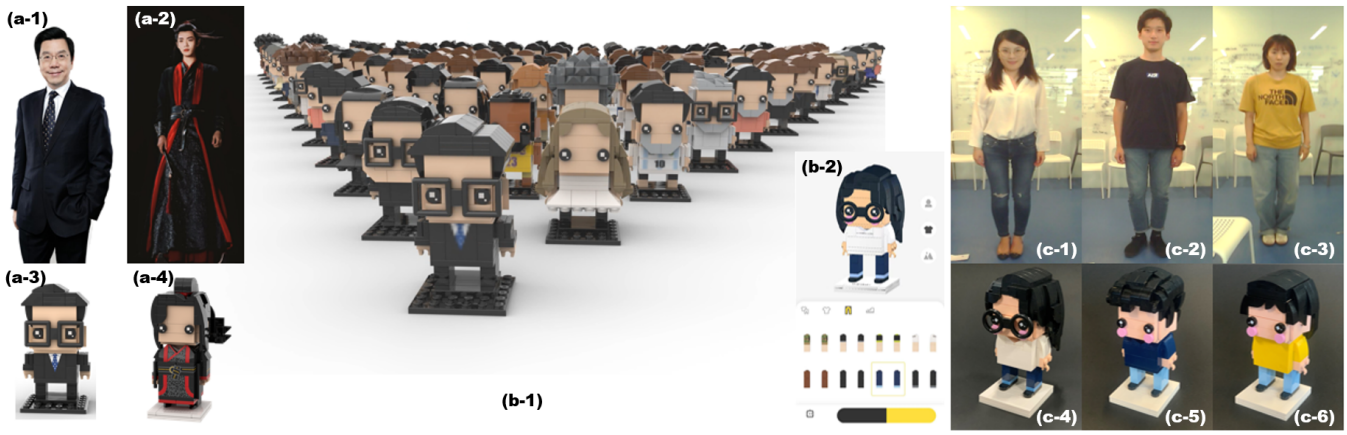

Fig. 5. Demonstrations of system performance: (a) the generated brick models with textures for KOLs, (b) a group show of the generated brick models in the early stage with the human-in-the-loop interface, and (c) the exampled input portraits captured by our proposed machine and the corresponding outputs – the physical brick models.

library in the early stage. The scale of design library showed a significant convergence trend when the number of accumulated customers reached 800. After that, we started to use the combination of AI-designed brick models and assembly instructions with human-designed textures printed on bricks. Up to this moment, over 1300 orders have been made online showing the excellent performance generating brick models, and Fig.5(b-1) shows a group of generated brick models. More guidance and information can be found at https://air.tsinghua.edu.cn/discover.

We have accumulated over 500 customers and there so far exists no customer complaint about assembly instructions. We also revisited 50 customers for enquiring assembly experience via follow-up phone callings. No negative comments came to assembly orders but some suggestions on visualization were adopted in our updated versions. It is worth mentioning that: a customer was quite surprised that her 50-year-old mother could independently follow instructions to assemble bricks.

B. System Demonstration

We set up a system demonstration of the complete workflow: portrait shooting, user interaction, design generation and fabrication. For the lack of general stock management mechanism, the brick stocks are loaded in advance based on 3 selected individuals. Meanwhile, we develop an interface as shown in Fig.5(b-2) for users to fine-tune generated brick models using the laptop. The input portraits and output assembled bricks can be referred to Fig.5(c). The detailed statistics can be found in TABLE I. It should be noted that the runtime counts fabrication time only with considerations as follows: (a) the spent time with human machine interface is uncontrollable and highly dependent on personal preferences, and (b) the pre-computing time (including brick model generation, stock control, instruction generation, and machnical motion planning) is less than 3s on a server with two GeForce RTX 3090 GPUs and the machine can start operating fixed bricks (e.g. base and internal structure) which are persistent among designs during this computation.

TABLE I
STATISTICS OF SYSTEM DEMONSTRATIONS

Portrait	Brick Number	Assembly Step	Runtime
Girl A (c-1)	178	83	155s
Boy B (c-2)	165	74	150s
Girl C (c-3)	147	57	141s

VII. CONCLUSIONS

The prototype of an automatic machine is built for converting the captured portrait into a physical gadget made up of LEGO bricks, with the purpose of enabling laymen to customize figure bricks at will. The performance of the brick model generation and instruction generation modules are evaluated extensively by collecting feedbacks from real customers. The results show positive comments on both artistic effect and assembly friendliness from the majority. Moreover, our system can work efficiently to support an onsite customization with a live captured portrait within 3 minutes, showing the opportunity to upgrade traditional make-to-stock production mode to an onsite customization mode. By simply adding more brick model templates besides portraits, our proposed system is expected to open new doors for LEGO retails.

One may notice that our results may partially limited by the expressiveness of LEGO bricks, strict requirements to supply chain, and absence of brick texture print process. Longer studies targeting AI-based brick texture design, data-driven stock re-arrangement need to be conducted to optimize our system further.

ACKNOWLEDGEMENT

We are grateful to Qianzhi Technology and Qimeng for supporting this project, especially on the operation phase including supply chain, e-commerce and customer service.

REFERENCES

[1] D.W. Currier. Automation of sheet metal design and manufacturing. In *17th Design Automation Conference*, pages 134–138, 1980.

[2] Satyandra K Gupta and Dana S Nau. Systematic approach to analysing the manufacturability of machined parts. *Computer-Aided Design*, 27(5):323–342, 1995.

[3] Cheng-Hua Wang and R. Sturges. Bendcad: a design system for concurrent multiple representations of parts. *Journal of Intelligent Manufacturing*, 7:133–144, 1996.

[4] Jay Patel and Matthew I. Campbell. An Approach to Automate and Optimize Concept Generation of Sheet Metal Parts by Topological and Parametric Decoupling. *Journal of Mechanical Design*, 132(5), 04 2010. 051001.

[5] Yuki Mori and Takeo Igarashi. Plushie: An interactive design system for plush toys. *ACM Trans. Graph.*, 26(3):45–es, July 2007.

[6] Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive couture for interactive garment modeling and editing. *ACM Trans. Graph.*, 30(4), July 2011.

[7] Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. Computational design of linkage-based characters. *ACM Trans. Graph.*, 33(4), July 2014.

[8] Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Transactions on Graphics (TOG)*, 33(4):1–10, 2014.

[9] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015.

[10] Adriana Schulz, Cynthia Sung, Andrew Spielberg, Wei Zhao, Robin Cheng, Eitan Grinspun, Daniela Rus, and Wojciech Matusik. Interactive robogami: An end-to-end system for design of robots with ground locomotion. *The International Journal of Robotics Research*, 36(10):1131–1147, 2017.

[11] Jeffrey I Lipton, Adriana Schulz, Andrew Spielberg, Luis Trueba, Wojciech Matusik, and Daniela Rus. Robot assisted carpentry for mass customization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3540–3547. IEEE, 2018.

[12] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, 34(6):1–16, 2015.

[13] Ryota Natsume, Shunsuke Saito, Zeng Huang, Weikai Chen, Chongyang Ma, Hao Li, and Shigeo Morishima. Siclope: Silhouette-based clothed people. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[14] Jason M. Saragih, Simon Lucey, and Jeffrey F. Cohn. Real-time avatar animation from a single image. In *2011 IEEE International Conference on Automatic Face Gesture Recognition (FG)*, pages 117–124, 2011.

[15] Dina R Berkowitz and John Canny. Designing parts feeders using dynamic simulation. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1127–1132. IEEE, 1996.

[16] Yukiyasu Domae, Akio Noda, Tatsuya Nagatani, and Weiwei Wan. Robotic general parts feeder: Bin-picking, regrasping, and kitting. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5004–5010. IEEE, 2020.

[17] Rebecca A. H. Gower, Agnes E. Heydtmann, and Henrik G. Petersen. Lego: Automated model construction. In *European Study Group with Industry*, pages 81–94, 1998.

[18] Pavel Petrovič. Solving LEGO brick layout problem using evolutionary algorithms. In *Proc. NIK (Norsk Informatikkonferanse)*, pages 87–97, 2001.

[19] David V. Winkler. Automated brick layout. In *Proc. BrickFest*, pages 145–166, 2005.

[20] Romain Testuz, Yuliy Schwartzburg, and Mark Pauly. Automatic generation of constructible brick sculptures. In *Eurographics (short paper)*, pages 81–84, 2013.

[21] Ben Stephenson. A multi-phase search approach to the lego construction problem. In *Proc. Symposium on Combinatorial Search (SoCS)*, pages 89–97, 2016.

[22] Seung-Mok Lee, Jae Woo Kim, and Hyun Myung. Split-and-merge-based genetic algorithm (sm-ga) for lego brick sculpture optimization. *IEEE Access*, 6:40429–40438, 2018.

[23] Bram Lambrecht. Voxelization of boundary representations using oriented plates, 2006. University of California, Berkeley, `http://lego.bldesign.org/` [Online; accessed 18-May-2018].

[24] Grim Yun, Cheolseong Park, Heekyung Yang, and Kyungha Min. Legorization with multi-height bricks from silhouette-fitted voxelization. In *Proc. CGI*, 2017. Article No. 40.

[25] Martin Waßmann and Karsten Weicker. Maximum flow networks for stability analysis of structures. In *Proc. Annual European Conference on Algorithms (Lecture Notes in Computer Science, vol 7501)*, pages 813–824, 2012.

[26] Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. Legolization: Optimizing LEGO designs. *ACM Trans. on Graph. (SIGGRAPH Asia)*, 34(6), 2015. Article no. 222.

[27] Ming-Hsun Kuo, You-En Lin, Hung-Kuo Chu, Ruen-Rone Lee, and Yong-Liang Yang. Pixel2brick: Constructing brick sculptures from pixel art. *Computer Graphics Forum (Pacific graphics)*, 34(7):339–348, 2015.

[28] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics (TOG)*, 22(3):828–837, 2003.

[29] J.D. Wolter. On the automatic generation of assembly plans. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 62–68 vol.1, 1989.

[30] L.S. Homem de Mello and A.C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, 7(2):228–240, 1991.

[31] Steven Feiner. Apex: An experiment in the automated creation of pictorial explanations. *IEEE Computer Graphics and Applications*, 5(11):29–37, 1985.

[32] Jock Mackinlay. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics (Tog)*, 5(2):110–141, 1986.

[33] Niloy J Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. Illustrating how mechanical assemblies work. *ACM Transactions on Graphics-TOG*, 29(4):58, 2010.

[34] Sankar Jayaram, Hugh I Connacher, and Kevin W Lyons. Virtual assembly using virtual reality techniques. *Computer-aided design*, 29(8):575–584, 1997.

[35] Lei Hou, Xiangyu Wang, Leonhard Bernold, and Peter ED Love. Using animated augmented reality to cognitively guide assembly. *Journal of Computing in Civil Engineering*, 27(5):439–451, 2013.

[36] Ankit Gupta, Dieter Fox, Brian Curless, and Michael Cohen. Duplotrack: a real-time system for authoring and guiding duplo block assembly. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 389–402, 2012.

[37] Man Zhang, Yuki Igarashi, Yoshihiro Kanamori, and Jun Mitani. Component-based building instructions for block assembly. *Computer-Aided Design and Applications*, 14(3):293–300, 2017.

[38] Ludwig Nägele, Alwin Hoffmann, Andreas Schierl, and Wolfgang Reif. Legobot: Automated planning for coordinated multi-robot assembly of lego structures. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9088–9095. IEEE, 2020.

[39] Mingxing Tan and Quoc V Le. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*, 2021.

[40] Menglin Jia, Mengyun Shi, Mikhail Sirotenko, Yin Cui, Claire Cardie, Bharath Hariharan, Hartwig Adam, and Serge Belongie. Fashionpedia: Ontology, segmentation, and an attribute localization dataset. In *European Conference on Computer Vision (ECCV)*, 2020.

[41] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[42] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[43] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming: A new approach to integrate cp and mip. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 6–20. Springer, 2008.

[44] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.