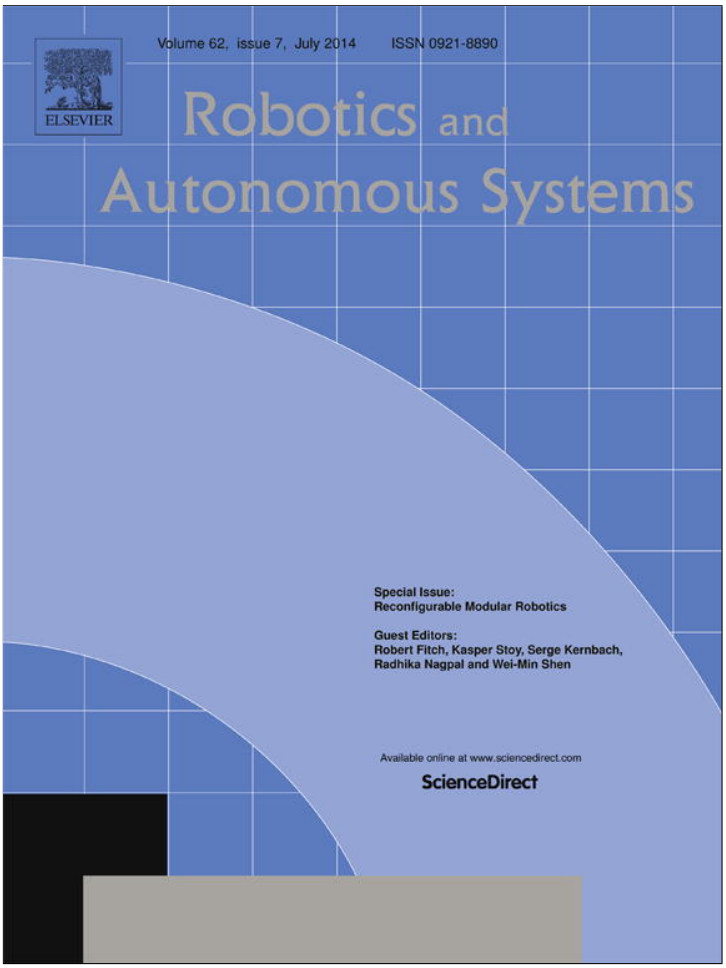


Robotics and Autonomous Systems 62 (2014) 976–1001

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Designing and programming self-folding sheets

Byoungkwon An *, Daniela Rus

MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

HIGHLIGHTS

- We propose a new programmable self-folding sheet model and architecture.
- We describe and analyze algorithms for the automatic design of self-folding sheets.
- We describe and analyze algorithms for the synthesis of sticker placement.
- We build and experiment with devices of two different types of self-folding robots.

ARTICLE INFO

Article history:
Available online 31 August 2013

Keywords:
Self-folding
Self-reconfiguration
Origami
Programmable matter
Sticker programming
Design
Compiler
Digital fabrication

ABSTRACT

This paper considers a robot in the form of a self-folding sheet that is capable of origami-style autonomous folding. The sheet is composed of triangular tiles, folding actuators and an integrated electronic substrate, and is formed as an $n \times m$ box-pleated crease pattern. The design of the sheet is generated by an automated sheet design algorithm. We control the sheet with a programming method including a hardware model and supporting algorithms. In this paper we present the programming method. We describe and analyze the algorithms that generate designs and programs for the sheet. We finally demonstrate and analyze experiments with 4×4 and 8×8 self-folding sheet devices.

1. Introduction

A self-folding sheet is a robotic sheet that autonomously transforms its shape by folding into users' desired shapes. Our vision is to develop the hardware and software technology that will allow users to transform a self-folding sheet into desired shapes by adding physical stickers to select and trigger a control sequence. We imagine sheets capable of folding into a variety of objects, such as a table, an airplane or a tent. Applications include digital fabrication, on-demand construction of objects in remote environments, on-demand creation of tools, etc. Our aim is to automate the creation of origami objects.

We developed a novel device called the self-folding sheet (Fig. 1). This device has an $n \times n$ box-pleated pattern (for $n = 4$ and $n = 8$). We associate a SMA (Shape Memory Alloy) actuator with each edge of the sheet and embed supporting electronics. The sheet can be viewed as a modular robot system, where each tile in the system corresponds to a module. The sheet can fold by following planning algorithms, such as those described in [1], to achieve a three-dimensional shape. The planner provides the required sequence of origami folds, which can be executed using the actuators embedded on the sheet.

Making three-dimensional shapes by folding has advantages over achieving shape formation using modular self-reconfiguring robot systems composed of individual independent modules. Since the modules are connected at all times, the self-folding sheet is less prone to the type of connection and disconnection errors that occur in unit-modular systems. The planning system can be computed in a centralized fashion and executed in a highly parallel fashion. The folding operation is relatively easy to control. The challenge, however, is in fabricating a self-folding sheet that is capable of physically delivering self-folding actions, especially with multiple folds on the same edge, and in the planning algorithm that will synthesize the covert folding sequence.

In our prior work, we described the self-folding concept [2] and a centralized planner for multi-origami folding from a single sheet [1]. In this paper, we describe the fabrication process for self-folding sheets (Fig. 1) with embedded electronics and actuation. We also present the design and fabrication of a controller that selects the control for one of the desired shapes associated with the sheet, and the control sequence required to actuate that shape. Our solution is called the *sticker programming*. In the sticker programming, the control sequence is achieved by adding stickers, which are small segments of conductive materials, to key locations on the sheet. The addition of the stickers completes a circuit that

* Corresponding author. Tel.: +1 617 253 6532.
E-mail addresses: dran@csail.mit.edu (B. An), rus@csail.mit.edu (D. Rus).

0921-8890/$ – see front matter
http://dx.doi.org/10.1016/j.robot.2013.06.015

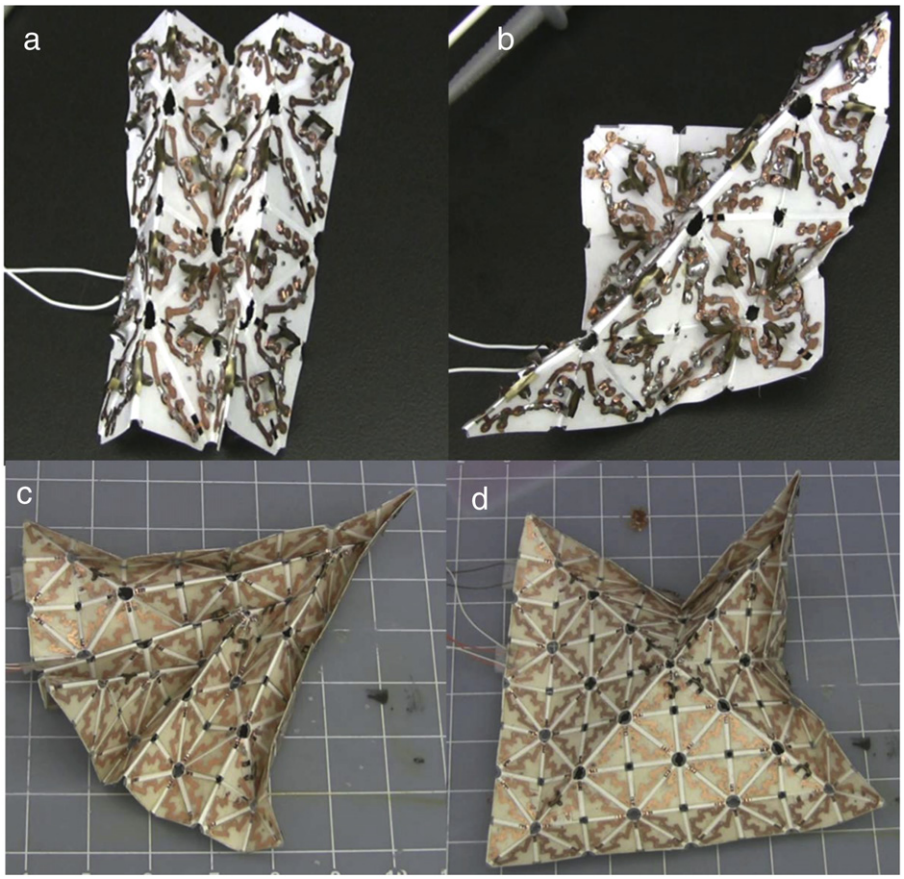


Fig. 1. Two self-folding sheets transform themselves into programmed objects. (a) Vertical folding. (b) Diagonal folding. (c) Space shuttle. (d) Hat.

triggers the function of an actuator. By adding/removing different stickers connecting other locations, we select a control sequence for achieving another described shape. Given the desired shapes, we can automatically compute the number of stickers and their placement on the sheet. Finally, we give experimental results collected from using a 4×4 self-folding sheet and an 8×8 self-folding sheet (Fig. 1). Our contributions in this paper are (1) a self-folding sheet model and its architecture, (2) algorithms for the design of self-folding sheets, (3) algorithms for the synthesis of sticker placement and (4) experiments with two different self-folding sheet devices.

1.1. Related works

In our previous work [2], we introduced a sheet that folded itself into two origami shapes. The mechanical parts of the sheet were triangular glass-fiber tiles, silicon joints and folding actuators. The sheet was formed as a 4×4 box-pleated crease pattern (Fig. 2). Demaine et al. proved that an $n \times n$ box-pleated tiling has, as a folded state, any polyhedral surface made up of $O(n)$ unit cubes on the cubic lattice [3]. They [4] showed that any folded state can be reached by a continuous folding motion without the material penetrating itself.

The multiple origami planner [1] generates a folding plan for this sheet. Given multiple target origami shapes, the origami planner identifies the groups of the actuators that simultaneously fold and the folding sequences of the groups for each target shape. For example, if a shape is achieved by folding two lines sequentially, the actuators on the first line are identified as the first group and the actuators on the second line are the second group. When we input this origami shape, the planner gives these two groups and the folding sequence.

Although the planner automatically generated the folding plan, we controlled the previous sheet [2] with a manually designed circuit. We embedded electronic routes for each actuator group and sent electronic current to the selected routes sequentially. The sheet achieved two 4×4 origami shapes with this manually designed circuit. This control method is however intractable for bigger sheets or more complex origami objects. In this paper, we describe a new control method, called the sticker programming, to solve this control challenge. This concept was introduced in [5], but the details of the hardware model and the programming algorithm were not described. The previous paper presented the old design algorithm that only works for $2^n \times 2^n$ sheets. This paper includes the detailed models for the sticker programming, the automated sheet design algorithm for $n \times m$ sheets, and theoretical correctness proofs and analyses of the designing and programming algorithms. The paper also demonstrates and analyzes the experiments with self-folding sheet devices.

Nagpal [6,7] introduced a biologically-inspired control method for a multiagent system, including a programming language that transforms into a language for the multiagents. She applied and simulated this method for a self-folding system. In her simulation, a sheet is composed of many cells (agents); each cell has simple computation and communication ability and the cells on a line can fold the structure. Some simulations ran with several thousand cells. Our programming method, in contrast, is for $n \times m$ box-pleated sheets that are constructed by connecting the triangular tiles with embedded electronic circuits and folding actuators.

The self-folding sheet autonomously transforms its 2D shape into 3D shapes as a new family of self-reconfigurable systems. Our group and other groups built the systems and the algorithms [8–25]. [26] is a good review of this field.

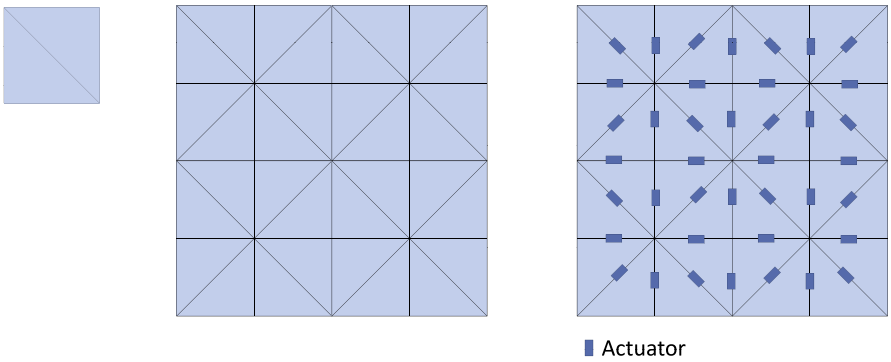


Fig. 2. A 1×1 self-folding sheet (left) is a fundamental module of self-folding sheets. A 4×4 self-folding sheet (middle), and with folding actuators (right). The 4×4 sheet is composed of 16 ($=4 \times 4$) 1×1 self-folding sheets.

Our research involves mechanical and electronic parts on a level of micro-thickness. The micro-thick folding actuators are built with SMA sheets [27,28] and Polypyrrole (PPy) [29,30]. The stretchable circuit is introduced in [31].

The origami folding is used for fabrication of micro objects. Whitesides, et al. achieved micro 3D objects by folding a planar object [32].

Balkcom and Mason [33–35] have built a robot that makes a sequence of *simple folds*—folds along a single line at a time. The robot folds a restrictive class of origami models. By contrast, our folds are generally more complicated, involving several simultaneous creases. Other works considered robots for automatic folding of cartons and packaging [36,37] with external actuation. By contrast, in our work, the actuation of the sheet is internal; the sheet itself is a self-folding robot and the robot folds itself into target objects.

2. Technical approach

Given k desired 3D objects, our goal is to design and fabricate a programmable self-folding sheet robot and to compute and program a control sequence required to fold the 3D objects from the sheet. The planning algorithm has been described in [1]. Here we discuss how to go from the theoretical plan to an executable sequence.

To facilitate automatic designing and programming, we design our sheets following the self-folding sheet architecture (Fig. 3). The architecture has a hardware part (right) and an algorithm part (left). Each part has several layers. The bottom layer of the hardware part has the kinematic structure of self-folding sheets. The next layer has actuators associated with each folding joint. The third layer has the electronic infrastructure that controls actuation systems (actuators) via sockets. The fourth layer is the programming layer. It contains small components of conductive materials that can be added to the self-folding sheet to connect and trigger the action of actuators. Each small component is called a *connector*. A set of the connectors is called a *sticker*. By adding these connectors at designated locations, we form a complete circuit that triggers an entire execution sequence to achieve a 3D shape. The sticker contains folding information of multiple origami shapes. A sticker can be removed and replaced by a different sticker to fold a different 3D object. The fourth layer also contains input signals that have the triggering sequences to achieve a selected shape.

The algorithm part has two layers. The bottom layer has an automatic designing algorithm that generates optimized self-folding sheet designs. The top layer has an automatic programming algorithm that generates *sticker programs*.

In Section 3, we introduce the theoretical model of self-folding sheets. In Sections 4–7, we describe and analyze the algorithms generating self-folding sheet designs and sticker programs. In Sections 8–10, we present the implemented devices and the experimental results. In Sections 11 and 12, we discuss and conclude our method and devices.

3. Model of self-folding sheet

In this section, we explain the theoretical models of the self-folding sheets for the algorithms and the hardware discussed in this paper. The *self-folding sheet model* is 2-dimensional rectangular-shaped and is composed of three models: a *box-pleated structure model*, an *actuator model* and a *sticker controller model*.

3.1. Box-pleated structure model

Fig. 2 shows the simplified kinematic structure of self-folding sheets. The structure is composed of rigid tiles and flexible joints (hinges) and embeds an $n \times m$ box-pleated pattern.

An $n \times m$ structure is composed of $n \times m$ square-shaped modules. Each module is composed of two triangular tiles and one diagonal joint. For instance a 1×1 self-folding sheet contains one module (Fig. 2 (left)), while a 4×4 sheet contains 16 ($=4 \times 4$) modules (Fig. 2 (middle)).

A *folding angle* is the supplement of the dihedral angle between the two faces meeting at the joint (Fig. 4). The initial state of the folding angle is 0°. The angle is in a range of $+180°$ to $-180°$. The sign of the fold angle determines the crease as either a valley fold or a mountain fold (Fig. 4).

3.2. Actuation model

In our actuation model, an actuator is placed on each joint, as shown in Fig. 2 (right), and folds its corresponding joint on demand to a specified angle, such as 0°, $+90°$, $-90°$, $+180°$, and $-180°$.

A folding actuator is modeled as a finite state transducer (Definition 1). Each actuator has n input ports and n ground ports. Fig. 5 shows the diagram of an actuator.

Definition 1. An actuator is a 6-tuple $A = (Q, \Sigma, \Gamma, \delta, w, q_0)$, where:

Q, Σ, Γ are finite sets and

1. Q is the set of the states,
2. Σ is the set of the actuator codes (input signals):
$\Sigma = \{0^n\} \cup \{0^p 10^q \mid p+q+1 = n, n$ is the length of each input signal of $A\}$,
3. Γ is the set of the folding angles,
4. δ is the transition function: $Q \times \Sigma \to Q$,
5. w is the output function: $Q \to \Gamma$, and
6. $q_0 \in Q$ is the initial state.

When the actuator receives an input signal $a_{in} \in \Sigma$, it folds its edge into an angle $a_{out} \in \Gamma$. While the actuator does not receive any input signal (all 0s), it keeps its angle until it receives the next input signal.

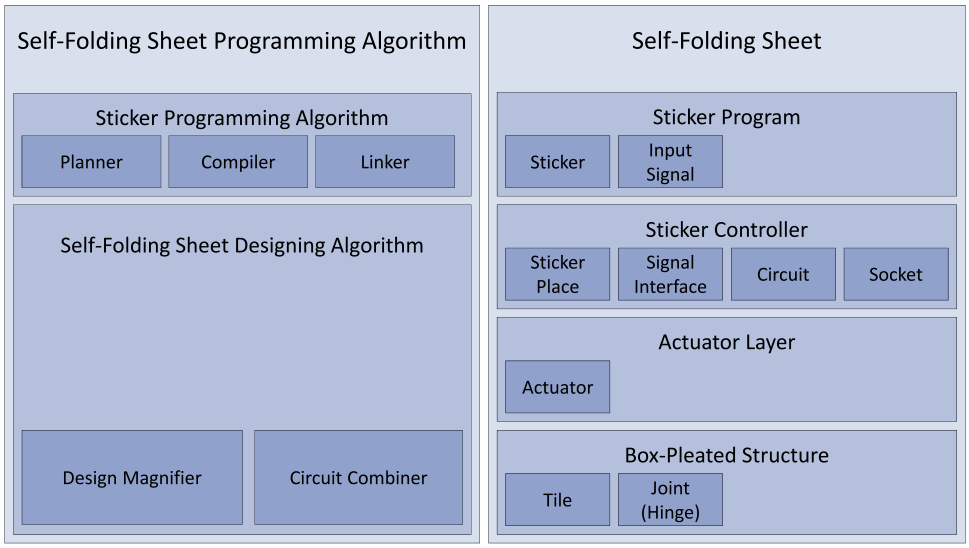


Fig. 3. Self-folding sheet architecture.

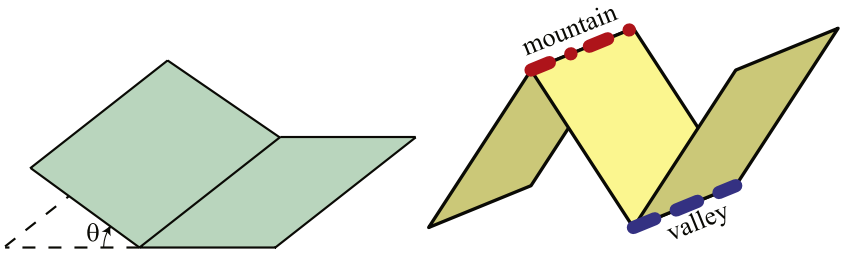


Fig. 4. (Left) The folding angle at a crease is the supplement of the dihedral angle. (Right) A crease can be folded as either a mountain fold or a valley fold [1].

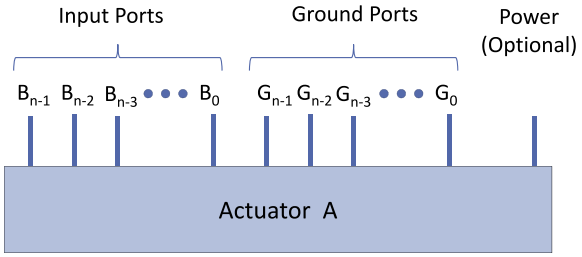


Fig. 5. An actuator model.

An input signal is a decoded binary code. Each code has only one 1 or is all 0s. We select this binary code model due to our origami plan model [1] and the implemented folding actuators [2,27,28] that work with the decoded binary codes.

For this paper, we built actuators with flat SMA (Shape Memory Alloy) sheets, which have micro-thickness and sub-centimeter width [2,27,28]. [29,30] describe the Polypyrrole (PPy) micro folding actuators.

We can classify the actuators by folding directions: unidirectional actuators and bidirectional actuators. A unidirectional actuator folds its joint into either a mountain fold or a valley fold. In this paper and previous papers [2,27], the experiments were performed with the unidirectional actuators $A_1 = (Q, \Sigma, \Gamma, \delta, w, q_0)$, $A_2 = (Q, \Sigma, \Gamma', \delta, w', q_0)$, where:

1. $Q = \{q_0, q_1\}$,
2. $\Sigma = \{0, 1\}$,
3. $\Gamma = \{0^\circ, +180^\circ\}$,
4. $\Gamma' = \{0^\circ, -180^\circ\}$,
5. $\delta(q, a) = q_0$ (if $q = q_0$ and $a = 0$), q_1 (otherwise),
6. $w(q_0) = 0^\circ$, $w(q_1) = +180^\circ$, and
7. $w'(q_0) = 0^\circ$, $w'(q_1) = -180^\circ$.

Fig. 6 shows the diagram of A_1 (left) and A_2 (right).

The bidirectional actuator folds the joint into both a mountain fold and a valley fold. [28] demonstrates the implementation of a bidirection actuator and Fig. 7 shows the model of this bidirectional actuator. The initial angle of the actuator is 0°. According to the input signal this actuator can fold either $+180^\circ$ or -180°. However, once the actuator folds into one angle, this actuator can only fold back to 0°.

As shown in Fig. 8, each actuator has its address. (a) shows the actuator addresses for a 1×1 self-folding sheet. The addresses of the actuators on the left, diagonal, and bottom edges are $l(1, 1)$, $d(1, 1)$, and $b(1, 1)$, respectively.

For the 2×2 and 4×4 self-folding sheets ((b), (c)), although the actuator addresses of the top-left module (in column 1 and row 1) are the same as the addresses of the 1×1 sheet (a), since the module in column 2 and row 1 is 90° rotated, the actuators on the top, diagonal, and left edges are $A_{l(1,2)}$, $A_{d(1,2)}$, and $A_{b(1,2)}$, respectively. The module in column 2 and row 2 is 180° rotated while the module in column 1 and row 2 is 270° rotated. The addresses of the actuators are as shown in Fig. 8.

3.3. Sticker controller model

The sticker controller contains the electronic substrate to fold the self-folding sheet into users' desired shapes. In this section, we describe the controller model, including a wire only controller model, a sticker controller unit model, a circuit model and various sticker program models.

3.3.1. Wire only controller model

A wire is a conductive material, such as copper, that transports electronic signals or power. We define the sticker controller model

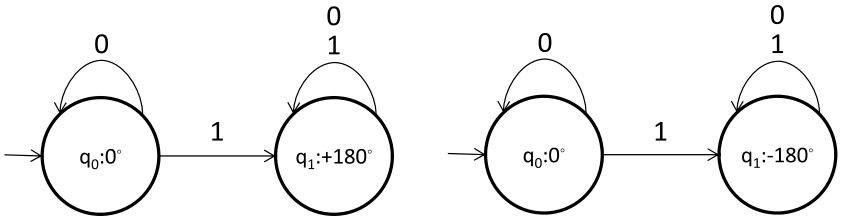


Fig. 6. Two unidirectional actuator models.

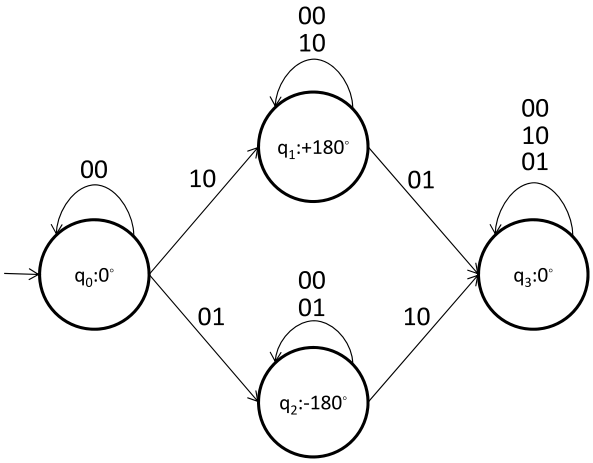


Fig. 7. A bidirectional actuator model.

with only wires; the sticker controller does not contain any other electronic parts, such as diodes, transistors or resistances. Since a wire is the thinnest and simplest electronic part, this model gives advantages to build thin devices with simple manufacturing processes. For instance, if we implement this controller model as a flexible electronic circuit, we only need a manufacturing process to build a flexible PCB (Printed Circuit Board) while other controller models with electronic parts, such as SMD (Surface-Mount Device), require additional manufacturing machines, such as SME (Surface-Mount Equipment). In semiconductor manufacturing processes, the electronic parts are on the scale of micro-thickness range. However, the parts are still thicker than wires; while the electronic parts are made with multiple depositions, the wires are built only with a couple of depositions.

Another advantage is flexibility for the implementation in various other energy systems. This model is designed in the electronic energy system; the signals and the power source are electronic energy. Since the wires for electronic energy can be replaced by water channels for water pressure, the model can thus be constructed in a water pressure system. Hand-heat, oil pressure or air pressure from exhalation can also be considered as alternative energy sources.

A disadvantage of this model is that the circuit design for this model is more complex than the other circuit designs, because there is no electronic device controlling the electronic flow. However, we solve this disadvantage by building the automated circuit design algorithm and the automated programming algorithm (Section 4).

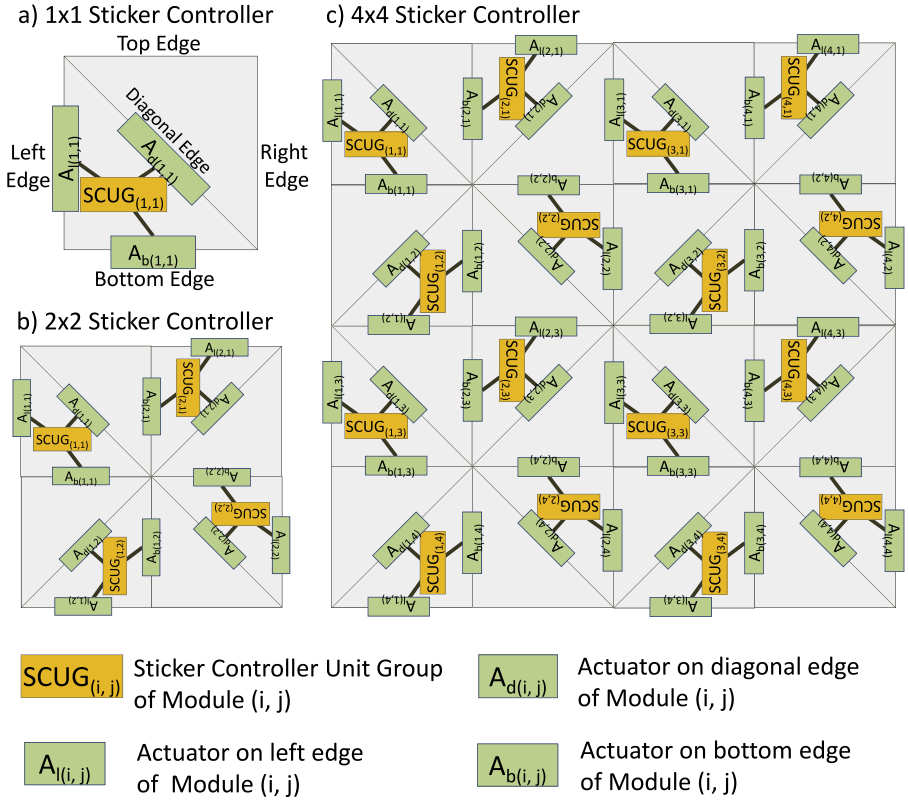


Fig. 8. Simplified models for 1 × 1, 2 × 2 and 4 × 4 sticker controllers. A 1 × 1 controller is composed of one module (a). A 2 × 2 controller is composed of four 1 × 1 controllers (b). A 4 × 4 controller is composed of four 2 × 2 controllers (c).

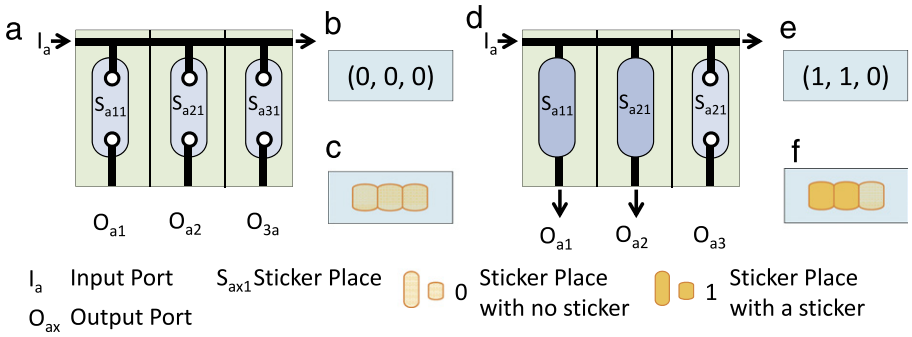


Fig. 9. (a)(b)(c) represent a 1–1 sticker controller unit group with no sticker. (d)(e)(f) represent a 1–1 sticker controller unit group with a sticker set. We draw two sticker controller unit groups in three different diagrams. Each model diagram shows the detailed information of the sticker controller unit (a)(d). The same information can be abstracted to a 3-tuple of the actuator codes as a code diagram (b)(e). It is depicted in a sticker diagram as a graphic image (c)(f).

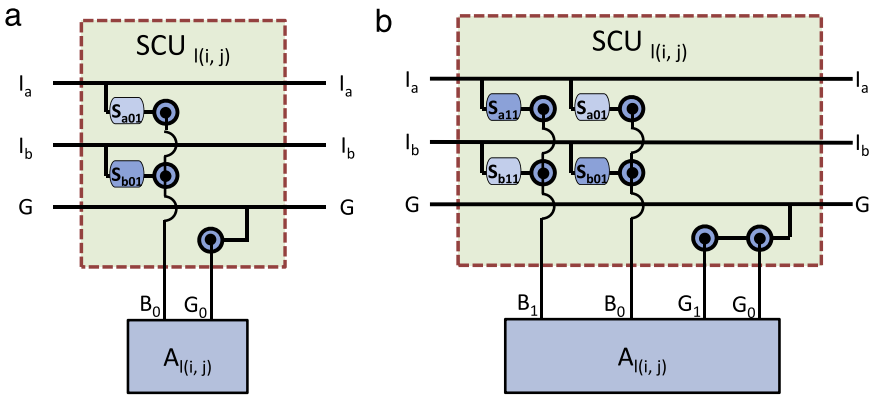


Fig. 10. (a) A 2–1 parallel sticker controller unit controls an actuator. The legs of the actuator are connected to the sockets of the unit. The sticker place S_{b01} has a connector, while the sticker place S_{a01} has no connector. (b) A 2–2 parallel sticker controller unit controls an actuator. The legs of the actuator are connected to the sockets of the unit. The sticker places S_{a11}, S_{b01} have connectors, while the sticker places S_{a01}, S_{b11} have no connector. In each (a) and (b) I_a, I_b on the left side are the input ports and I_a, I_b on the right side are the bypass ports. *G*s on both sides are the ground ports.

3.3.2. Sticker controller unit model

A *sticker controller unit* is a fundamental unit of the sticker controller (Fig. 9). The sticker controller unit controls an actuator according to input signals. It is composed of *sticker places*, *input ports*, *output ports*, *bypass ports* and *ground ports*. The output ports are connected to actuator legs via sockets. Because one module of a self-folding sheet is controlled by three actuators, a group of three sticker controller units controls one module.

A sticker controller unit is named by a *k*–*r* sticker controller unit, where:

- *k* is the number of the input ports, and
- *r* is the number of the output ports.

When a sticker controller is composed of *k*–*r* sticker controller units, we call it a *k*–*r* sticker controller.

Fig. 9 shows two 1–1 sticker controller unit groups with no sticker and with a sticker set. Each group is composed of three sticker controller units. We select the outputs by adding conductive materials, which we call connectors, on the selected sticker places. In Fig. 9(d), when input port I_a receives a signal, the units send the signals to O_{a1} and O_{a2}; O_{a1} and O_{a2} are connected to S_{a11} and S_{a21}. This causes the actuators connected to O_{a1} and O_{a2} to be activated. The input voltage of I_a and the output voltage of O_{a1} and O_{a2} are the same. After the signal is used, the signal is passed to the next sticker controller unit via bypass ports (arrow on each right side of (a) and (d)).

A sticker controller unit can be represented with a three-tuple (Fig. 9(b)(e)) or a simplified diagram (Fig. 9(c)(f)).

Parallel sticker controller unit model. In the *parallel sticker controller unit model*, actuators are parallelly connected on the sticker controller as shown in Fig. 10. The parallel sticker controller unit contains *k* input ports (left side), *k* bypass ports (right side) and two ground ports (left and right sides). All signals received from the *k* input ports are passed to the *k* bypass ports. When a connector is placed on a sticker place, such as S_{b01} in Fig. 10(a), the controller unit passes the signal to the output port connected to the socket, and then the actuator on the socket folds the joint.

Fig. 10(b) shows a 2–2 parallel sticker controller unit. In this figure, an actuator is connected to both I_a and I_b. Since the controller is in the wire only model, if I_a and I_b receive the signals together, the actuator receives the wrong input signal; we defined that each input signal has one 1 or is all 0s (Definition 1 in Section 3.2). However, this conflict does not occur because the origami planner computes correct plans in which each edge in each folding step folds to one angle [1], and the sticker programming algorithm compiles this plan correctly into the sticker program (Section 7).

Serial sticker controller unit model. In the *serial sticker controller unit model*, actuators are serially connected to the sticker controller. A *serial sticker controller unit* contains *k* input ports, *k* bypass ports and no ground port. The serial sticker controller unit model is used for our devices (Section 8). Fig. 11(a) shows the 1–1 serial sticker controller unit. The serial controller unit sends the input signal to either the bypass port or the selected output port. When the serial controller unit sends the signal to the actuator via the output port, the actuator uses the signal and passes it to the bypass port.

When a serial sticker controller unit has more than one input port, the controller unit has additional sticker places between the actuator and the bypass ports (Figs. 12 and 13). By using the additional sticker places, the signals used by the actuator are sent to the right bypass port.

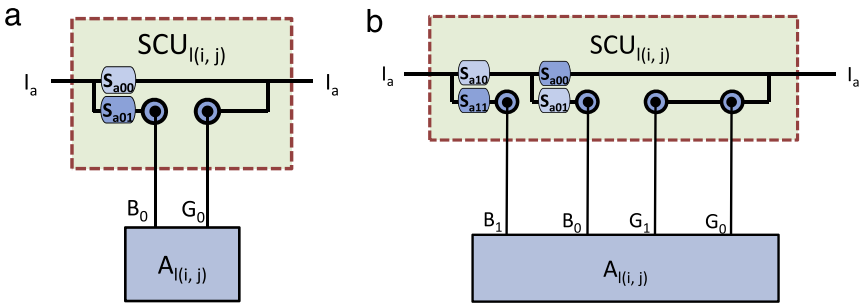


Fig. 11. (a) A 1–1 serial sticker controller unit controls an actuator. The legs of the actuator are connected to the sockets. The sticker place S_{a01} has a connector, while the sticker place S_{a00} has no connector. (b) A 1–2 serial sticker controller unit controls an actuator. The legs of the actuator are connected to the sockets of the unit. The sticker places S_{a11}, S_{a00} have connectors, while the sticker places S_{a10}, S_{a01} have no connector. In each (a) and (b) I_a on the left side is the input port and I_a on the right side is the bypass port.

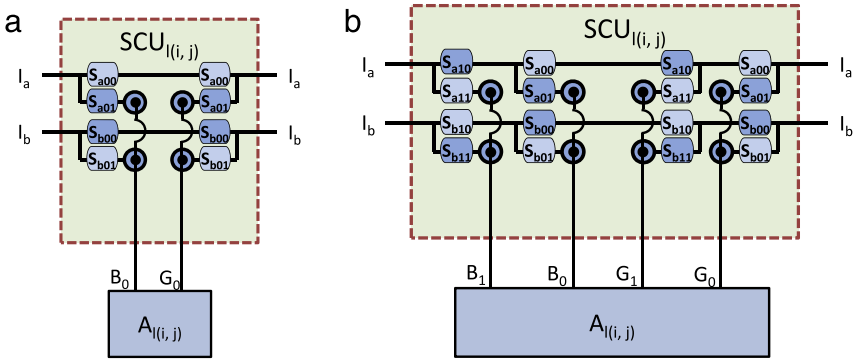


Fig. 12. (a) A 2–1 serial sticker controller unit that controls an actuator. The legs of the actuator are connected to the sockets. The sticker places S_{a01}, S_{b00} have connectors, while the sticker places S_{a00}, S_{b01} have no connector. (b) A 2–2 serial sticker controller unit that controls an actuator. The legs of the actuator are connected to the sockets of the unit. The sticker places S_{a10}, S_{a01}, S_{b11}, S_{b00} have connectors, while the sticker places S_{a11}, S_{a00}, S_{b10}, S_{b01} have no connector. In each (a) and (b) I_a, I_b on the left side are the input ports and I_a, I_b on the right side are the bypass ports.

The additional sticker places between the actuator and the bypass ports are synced with their related sticker places between the input ports and the actuator. The synced sticker places have the same address. If a sticker place has a connector, all other synced sticker places must have connectors. For example, in Fig. 12(a), sticker place S_{a00} between the input ports and the actuator and sticker place S_{a00} between the actuator and the bypass ports are synced; they have the same address. Figs. 12(b) and 13 show other examples of serial sticker controller units and synced sticker places.

The serial controller unit model requires fewer ports than the parallel controller unit model. For k input ports, the parallel controller unit model has $2(k+1)$ ports due to the ground ports, and the serial controller unit model has $2k$ ports.

More robust circuits can be built with the parallel controller unit. In the parallel sticker controller unit model, when some of the actuators or the actuator sockets break down and signals cannot pass through them, all of the unbroken actuators continue to work. However, in the serial sticker controller unit model, when one of the actuators or the actuator sockets breaks down and cannot pass the signal, all actuators stop working.

Sticker controller unit group model. Fig. 8 shows the sticker controller unit group model that controls 1×1, 2×2, and 4×4 self-folding sheets. Each sticker controller unit group $SCUG_{(i,j)}$ contains three sticker controller units $SCU_{l(i,j)}$, $SCU_{d(i,j)}$, and $SCU_{b(i,j)}$ and controls three actuators $A_{l(i,j)}$, $A_{d(i,j)}$, and $A_{b(i,j)}$ on a module (i, j).

3.3.3. Model of circuit

A *circuit* distributes the input signals, which a signal interface receives, to all connected sticker controller unit groups $SCUG_1$, $SCUG_2, \ldots, SCUG_n$. Where k is the number of the ports of the signal interface, the circuit is composed of k *independent networks*. Each independent network C_a is connected to each input port I_a of $SCUG_1, SCUG_2, \ldots, SCUG_n$. The self-folding sheet design algorithm draws the circuit design (Section 5).

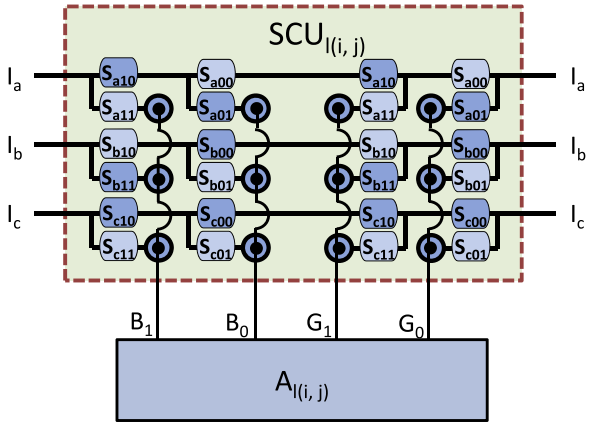


Fig. 13. A 3–2 serial sticker controller unit that controls an actuator. The legs of the actuator are connected to the sockets. The sticker places S_{a10}, S_{a01}, S_{b11}, S_{b00}, S_{c10}, S_{c00} have connectors, while the sticker places S_{a11}, S_{a00}, S_{b10}, S_{b01}, S_{c11}, S_{c01} have no connector. I_a, I_b, I_c on the left side are the input ports and I_a, I_b, I_c on the right side are the bypass ports.

Multiple layer circuit model. In the *multiple layer circuit model*, for k input ports, this model has to contain $O(k)$ layers (including insulating layers). These layers increase not only the thickness of the self-folding sheet, but also the number of manufacturing steps to stack the layers.

Single layer circuit model. In the *single layer circuit model*, a circuit is designed on one layer as shown in Fig. 14. All independent networks are on one layer.

3.3.4. Sticker program model

Given target origami shapes, the origami planning algorithm generates an origami plan [1]. The origami plan has two parts of folding information. The first part is the actuator group information; an actuator group G_i is a set of the actuators that always fold simultaneously. The second part is the folding sequences, where a folding sequence is a list of actuator groups $(G_1, G_2, \ldots, G_i, \ldots, G_k)$ and G_i folds in the ith folding step.

A sticker program, which is a method to input an origami plan into a sticker controller, is composed of a sticker and sequences of input signals. The sticker programming algorithm embedding the origami planner generates the graphic design of the sticker and the sequences of the input signals (Section 7). The graphic design of the sticker is called a sticker design and the set of the sequences of the input signals is called an actuation sequence.

We defined a sticker as a set of conductive materials (connectors) covered by a wide adhesive tape. When the sticker is attached on the top of the sticker controller, its conductive materials (connectors) fill the selected sticker places of the sticker controller units to trigger the selected actuators for the desired objects.

Since each input port I_a of all SCUs is connected on an independent network of the circuit, a signal sent to this network is transmitted to all I_a of the SCUs and the SCUs then trigger the selected actuators.

3.4. Composition

We build the model to minimize the layers when the self-folding sheet is manufactured. If the sheet has one input port, then all parts of the sheet design can be placed in one layer because no part of the sheet has to stack on the other parts, like the 1–1 and 1–2 serial sticker controller unit models (Fig. 11).

If the sheet has k input ports ($k \geq 2$), then all parts including the actuators can be placed in two layers because the legs of the actuators must cross over the sticker controller units, like the 2–1, 2–2 and 3–2 serial sticker controller unit models (Figs. 12 and 13) and all the parts of the sheet except the actuators can be placed in one layer.

Our implemented 4×4 self-folding sheet (Section 9) is composed of 9 layers; there are 3 layers for the box-pleated structure, 2 layers for the actuators on the top and bottom sides, 4 layers for the sticker controllers including 2 layer sockets, and 2 layers sharing the actuator layers for the sticker. Our 8×8 self-folding sheet (Section 10) is also composed of 9 layers; there are 3 layers for the box-pleated structure, 6 layers for the sticker controller, and 4 layers sharing the sticker controller layers for the actuators and the sticker.

3.5. Alternative sticker models

We described the basic sticker model in Section 3.3.4. Our 4×4 self-folding sheet was built with the basic sticker model. In this section, we discuss alternative sticker models.

Embedded sticker model. While a sticker in the basic sticker model is placed on the controller of a sheet as a separated layer, in the embedded sticker model a selected sticker and a controller are combined as one circuit layer. The design of this circuit layer is achieved by merging output images of the sticker programming and sheet design algorithms (Section 4). Like a basic sticker, an embedded sticker includes the multiple-step sequences of origami objects, although the sheet cannot be reprogrammed by changing its sticker.

This model has advantages of constructing micro 3D origami structures, despite its inability to reprogram. It resolves the challenges on the micro scale, including adding/removing stickers and maintaining the electronic connections between a sticker and a controller.

Actuator sticker model. In the actuator sticker model, by placing actuators or dummy metals, we select the actuating joints.

The SMA sheet actuator is one of the most expensive parts of the sheet device due to not only its expensive price, but also its complex building process. Since most origami shapes do not need to fold all edges, by placing actuators on selected joints, we can optimize the number of the actuators.

By replacing the actuators and the dummy metals, we can input new programs on the sheet. Our 8×8 self-folding sheet is implemented in this model (Section 10).

Electronic controllable sticker model. Although the thickness of the sheet will increase, stickers can be replaced by transistors or relays to reduce the reprogramming time. This model can be used for interactive transformation with touch sensors on surfaces or dynamic transformation with angle feedback sensors on joints.

3.6. Self-folding sheet design

A *self-folding sheet design* is a manufacturing-ready and programming-ready design that is automatically drawn by the *self-folding sheet designing algorithm* (Section 5).

The sheet design is composed of a *box-pleated structure design*, a *sticker controller design* and a *sticker place design*. The box-pleated structure design and the sticker controller design are composed of the images representing each layer of the self-folding sheet, like layered 2D CAD (computer-aided design) drawings.

The sticker place design contains an image of the sticker places and their addresses. The sticker place design is one of the inputs of the sticker linker (Section 7). We use $S_{l(i,j)abc}$ to denote a sticker place whose address is $l(i, j)abc$, where the sticker place is on input port a, output port b and signal $c \in \{0, 1\}$ of $SCU_{l(i,j)}$ (Figs. 10–13).

4. Self-folding sheet programming algorithm

Given k target shapes, an actuator model, a 1×1 self-folding sheet design and a combining circuit set, the self-folding sheet programming algorithm builds a sticker program and a self-folding sheet design. Fig. 15 shows an overview of the algorithm. The algorithm is composed of the sticker programming algorithm and the self-folding sheet designing algorithm.

Given k shapes and an actuator model, the sticker programming algorithm generates a sticker program including a sticker design and an actuation sequence. While the algorithm generates the sticker program, the origami planner generates an origami plan including the size of the sheet for all input shapes. The self-folding sheet designing algorithm uses this size as a target size.

Given a target size, a 1×1 self-folding sheet design and a combining circuit set, the self-folding sheet designing algorithm generates an $n \times m$ self-folding sheet design containing a box-pleated structure design, a sticker controller design and a sticker place design. The box-pleated structure design and the sticker controller design are manufacturing-ready designs. The sticker place design is used by the sticker linker of the sticker programming algorithm.

The sticker linker generates a sticker program that is composed of a sticker design and an actuation sequence. The sticker design is also a manufacturing-ready design and the user can manufacture the sticker from the sticker design. According to the element of the actuation sequence, the user inputs the input signals into the self-folding sheet.

In the next section, we describe the details of the self-folding sheet designing algorithm. We describe the sticker programming algorithm in Section 7.

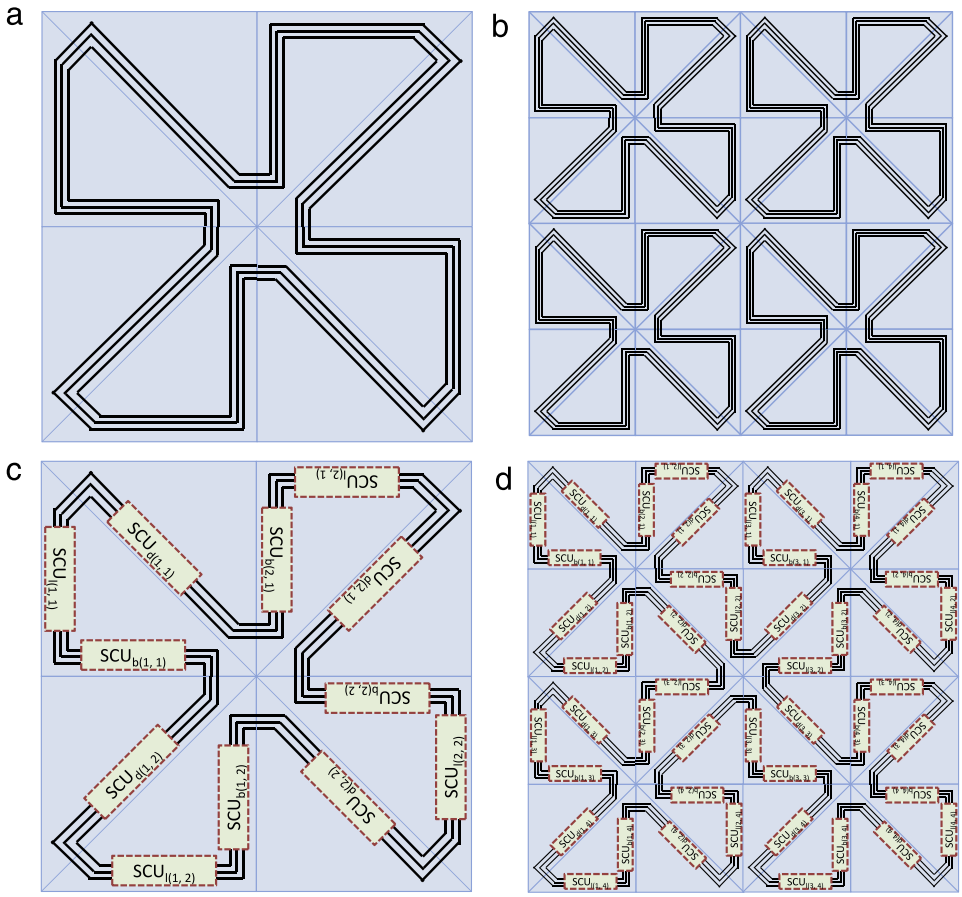


Fig. 14. Examples of the single layer circuit model having 3 independent networks. (a) 2×2 and (b) 4×4 single layer circuits. (c) 2×2 and (d) 4×4 single layer circuits with sticker controller units.

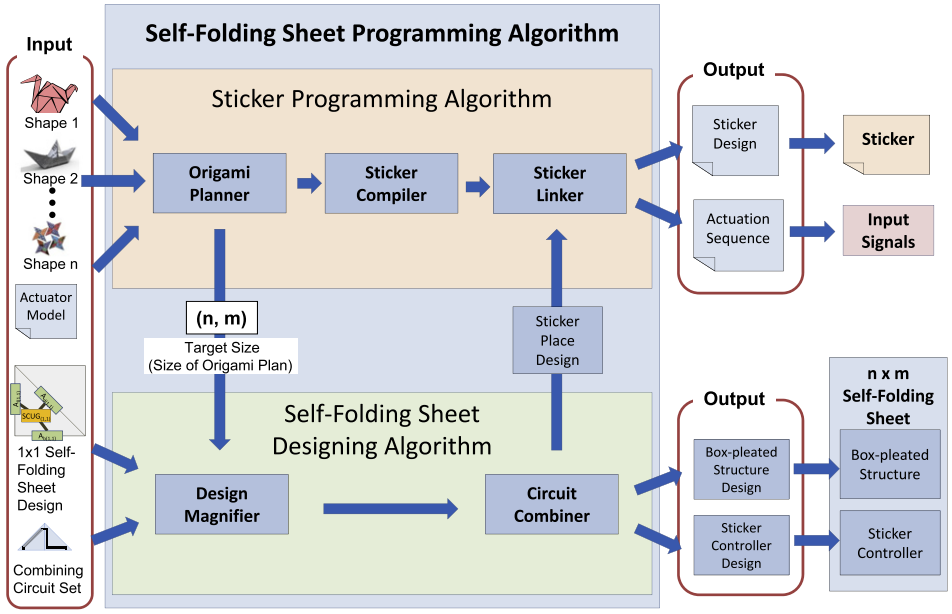


Fig. 15. Visual overview of self-folding sheet programming algorithm.

5. Self-folding sheet designing algorithm

Given a target size (n, m), a 1×1 self-folding sheet design and a *combining circuit set*, the self-folding sheet designing algorithm builds an $n \times m$ self-folding sheet design (Fig. 16). Fig. 17 gives an overview of the main steps of the algorithm.

n, m of the target size are base-k numbers ($k \geq 2$). The 1×1 self-folding sheet design is composed of a 1×1 box-pleated structure

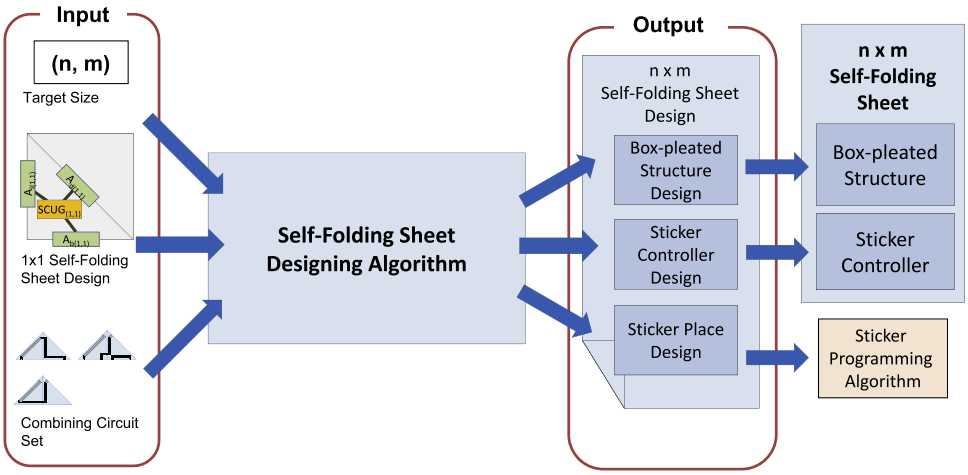


Fig. 16. Visual overview of self-folding sheet designing algorithm.

Self-Folding Sheet Designing Algorithm

1. Given a 1×1 self-folding sheet design, construct a 2×2 self-folding sheet design.
2. Given a target size (n, m), by copying the 2×2 sheet design $\frac{n}{2} \times \frac{m}{2}$ times, construct an $n \times m$ self-folding sheet design.
3. Correct the addresses of the sticker place design of the constructed sheet design. (because the address of the sticker controller unit group of the given 1×1 self-folding sheet design is $(1, 1)$, the algorithm needs to change $(1, 1)$ to (i, j) for each column i, row j module.
4. By the circuit combining algorithm, combine the separated circuits of the constructed sheet design into a fully connected circuit (Sec. 6).
5. Output the $m \times n$ self-folding sheet design composed of the $m \times n$ box-pleated structure design, the $m \times n$ sticker controller design and the $m \times n$ sticker place design.

Fig. 17. Algorithmic overview of self-folding sheet designing algorithm.

design, a 1×1 sticker controller design and a 1×1 sticker place design. The combining circuit set contains three combining circuits. The *pseudo-fractal combining circuit*, the 1-*to*-1 *combining circuit* and the 2-*to*-2 *combining circuit* are used for the pseudo-fractal, 1-to-1, and 2-to-2 combiners, respectively (Section 6).

The first step of the algorithm is to build a 2×2 self-folding sheet design, given the 1×1 self-folding sheet design (Fig. 17 Step 1). By rotating the 1×1 sheet design to 0°, 90°, 180°, and 270°, the algorithm has four modules (the four rotated 1×1 sheet designs) for the 2×2 sheet design. By assembling the rotated modules, the algorithm constructs the 2×2 sheet design.

In the next step, by copying the 2×2 sheet design $\frac{n}{2} \times \frac{m}{2}$ times, the algorithm constructs the $n \times m$ sheet design (Fig. 17 Step 2). If n is an odd number, the algorithm places the top-left and bottom-left modules of the 2×2 sheet design on the nth column. If m is an odd number, the algorithm places the top-left and top-right modules of the 2×2 sheet design on the mth row. In this step, the box-pleated structure design is completed but the circuit design layer is not completed; each independent network of the circuit layer is not connected as one circuit yet. The circuit design is completed in the fourth step by the *circuit combining algorithm*.

In the third step, the algorithm corrects the addresses of the sticker place design (Fig. 17 Step 3). Because the algorithm builds the $n \times m$ sticker place design with the given 1×1 sticker place design in the first and second steps, all addresses of the sticker controller unit groups of the sticker places are $(1, 1)$. By changing $(1, 1)$ to (i, j) for the address of each column i and row j module, each sticker place of the sticker place design has the correct address. After this step, the sticker place design of the $n \times m$ sheet design is complete.

Before the fourth step, if n or m is greater than or equal to 3, the circuit of the $n \times m$ sticker controller design is not fully connected; Fig. 19 shows an example of the design which is not fully connected while Fig. 29 shows an example of a fully connected circuit. In the fourth step, the circuit combining algorithm combines the separated circuits into one connected circuit correctly. The combining algorithm is run in $O(k^2)$ time and $O(k)$ space, where k is the input size of the combining algorithm. The details and the correctness proof of the circuit combining algorithm are described in Section 6. After this step, the sticker controller design is complete. In the fifth step, the algorithm outputs the $n \times m$ self-folding sheet design composed of the box-pleated structure design, the sticker controller design and the sticker place design.

An advantage of the algorithm is that the algorithm generates the sheet design whose size is exponentially bigger than the size of the input. Thus, we can generate a big sheet by giving a relatively small input. Where r is the size of the 1×1 self-folding sheet design and the combining circuit set, p and q are the sizes of m and n of the target size, and m and n are in the base-k numeral system ($k \geq 2$), the size of the input is $r + p + q$ and the size of the output design is $O(r \times k^{(p+q)})$. The algorithm runs in $O(r^2 \times k^{(p+q)})$ time and $O(r \times k^{(p+q)})$ space.

Circuit Combining Algorithm

1. Given an input circuit design, construct a circuit design by collecting all 2×2 closed circuits.
2. Run the pseudo-fractal combiner on the constructed circuit design (Sec. 6.1).
3. Given the input circuit design, construct a circuit design by collecting all circuits that are not a 2×2 closed circuit.
4. Run the 1-to-1 combiner on the constructed design (Sec. 6.2).
5. Construct an $n \times m$ circuit design by combining the outputs of the pseudo-fractal combiner and the 1-to-1 combiner.
6. Run the 2-to-2 combiner on the constructed circuit design (Sec. 6.3).
7. Output the circuit design that the 2-to-2 combiner outputs.

Fig. 18. Algorithmic overview of circuit combining algorithm.

6. Circuit combining algorithm

Given a combining circuit set and an $n \times m$ circuit design composed of separated circuits, the circuit combining algorithm combines the separated circuits into one fully connected circuit. The algorithm is composed of the three combiners: the *pseudo-fractal combiner*, the 1-*to*-1 *combiner* and the 2-*to*-2 *combiner*. The combiners replace some parts of the input design with the combining circuits in the combining circuit set. Fig. 18 shows an overview of the combining algorithm.

Fig. 19 is an example of an input of the circuit combining algorithm. All circuits on the ninth column or the seventh row are open circuits and the other circuits are 2×2 closed circuits.

6.1. Pseudo-fractal combiner

Given a circuit design composed of closed 2×2 circuits and a pseudo-fractal combining circuit, the pseudo-fractal combiner combines the closed circuits. The pseudo-fractal combiner locally transforms separated circuits into a fractal patterned circuit as shown in Fig. 22. However, the combiner does not always transform the whole circuits into a fractal patterned circuit; although the combiner always transforms a $2^n \times 2^n$ circuit into a fractal patterned circuit where $n \geq 1$ [5]. Fig. 20 describes an overview of the pseudo-fractal combiner.

In the first step, the combiner builds a 2×2 *pseudo-fractal combining patch* composed of four rotated pseudo-fractal combining circuits (Step 1 in Fig. 20). Fig. 21 shows an example of a pseudo-fractal combining circuit and its 2×2 pseudo-fractal patch.

In the second step, for each $j = 1, 2, \ldots, m-1$ and each $i = 1, 2, \ldots, n-1$, the algorithm checks the circuits of the modules (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$. If the circuits are four separated circuits, the algorithm replaces the modules with the 2×2 combining patch as shown in Fig. 22. The patch combines the four circuits into a connected closed circuit (Theorem 1). If the four circuits are not four separated circuits, the algorithm checks the next four modules.

Theorem 1. *Given a circuit design and its pseudo-fractal patch, iff the circuits on modules (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$ are four separated circuits, the patch combines the four circuits into one fully connected closed circuit.*

Proof. We use $C_{(i,j)}$ to denote a circuit crossing over a module (i, j). We use $(i, j)'$ to denote a module replaced with the module of a patch.

Let B be a set $\{((i, j), (i+1, j)), ((i+1, j), (i+1, j+1)), ((i+1, j+1), (i, j+1)), ((i, j+1), (i, j))\}$. If $C_{(i,j)}, C_{(i+1,j)}, C_{(i,j+1)}, C_{(i+1,j+1)}$ are four separated circuits, after patching, $C_{a'}$ and $C_{b'}$ connect C_a and C_b, where $(a, b) \in B$ (Fig. 27). Then, $C_{(i,j)}, C_{(i+1,j)}, C_{(i,j+1)}, C_{(i+1,j+1)}$ become one fully connected closed circuit.

Let (i, j) and (r, s) be two of four such modules. If $C_{(i,j)}$ and $C_{(r,s)}$ are not separated circuits, the circuits on the modules (i, j) and (r, s) are connected. After (i, j) is replaced by $(i, j)'$, the two ends of $C_{(i,j)'}$ are connected via $C_{(r,s)}$. However, after (r, s) is replaced with $(r, s)'$, the circuit on the $(r, s)'$ disconnects $C_{(i,j)'}$ and the two ends of $C_{(i,j)'}$ are disconnected. Thus, the patch does not combine the circuits of the four modules. Theorem 1 is true. □

Fig. 23 shows an example of the algorithm. The blue dash–dot–dot line rectangular area shows the input circuit design, which is composed of 2×2 closed circuits only. The red dash line and the purple dash–dot line squares are the patched circuits. Before the algorithm runs, the input design is composed of 12 ($=4 \times 3$) 2×2 closed circuits (Fig. 19). After the algorithm combines the input, the output is composed of 3 closed circuits (Fig. 23).

The algorithm runs in $O(p \times q)$ time and $O(p+q)$ space where p is the size of the input circuit and q is the size of the input combining circuit.

One of the fragile points of self-folding sheets is the wires that cross over the joints. Although we can combine all circuits with only the 1-to-1 and 2-to-2 combiners, we first run the pseudo-fractal combiner to minimize the number of the crossing wires. The pseudo-fractal patches do not have any additional crossing wires; each patching area of an input circuit design (Fig. 22 left) and a patch (Fig. 22 right) contains 4 crossing wires in the center. A 1-to-1 combining patch has one additional crossing wire (Fig. 26) while a 2-to-2 combining patch has two additional crossing wires (Fig. 28).

6.2. 1-to-1 combiner

The 1-to-1 combiner combines the circuits that are not 2×2 closed circuits. The blue dash–dot–dot line area in Fig. 24 is an example input of this combiner. The red dash line triangle areas are the patches.

The combiner uses two rotated patches for each side of the circuit as shown in Fig. 25. Each patch, called a 1-*to*-1 *patch*, combines the circuits as shown in Fig. 26.

After 1-to-1 combining, the circuit connects to each sticker controller unit with an actuator. In Fig. 26 (top-left), the patch connects two separated circuits by connecting the wire on the diagonal edge of the second module and the wire on the left edge of the third module (top-right). Although the bottom sticker

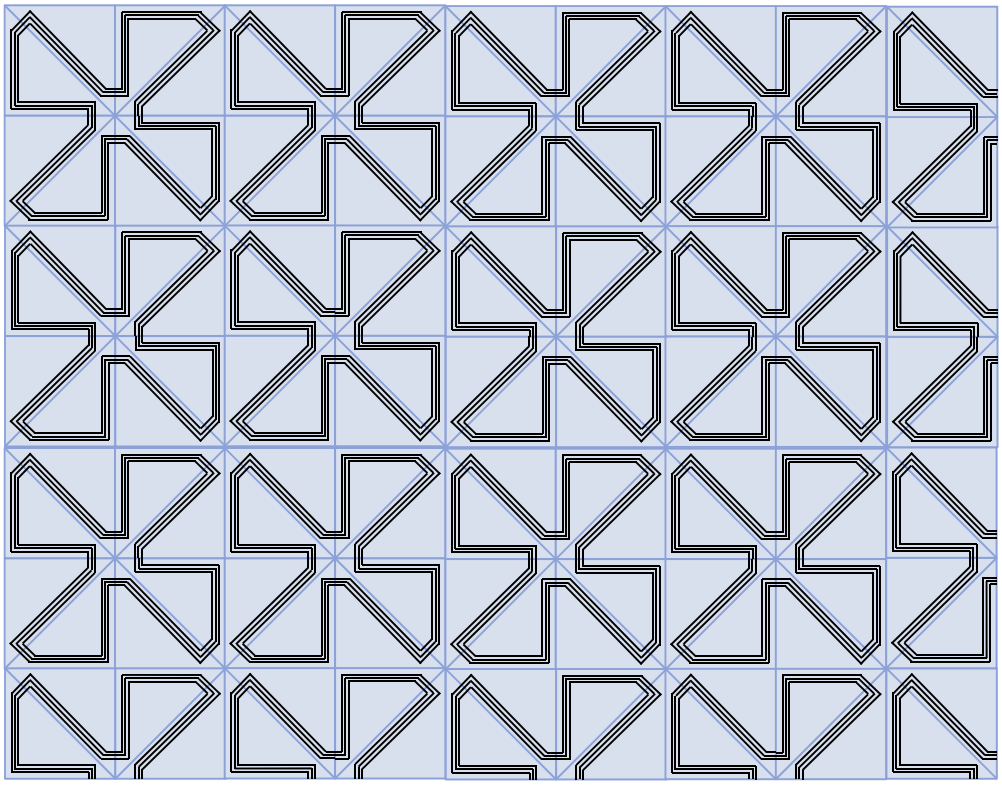


Fig. 19. Example of an input of the circuit combining algorithm. This 9 × 7 circuit design is generated by the designing algorithm (Fig. 17 Step 2).

Pseudo-Fractal Combiner

1. Given a pseudo-fractal combining circuit, construct a 2 × 2 pseudo-fractal combining patch (Fig. 21).
2. Given a circuit design composed of 2 × 2 closed circuits, combine the 2 × 2 closed circuits as the following:
   (a) For $j = 1 \rightarrow m - 1$:
      i. For $i = 1 \rightarrow n - 1$:
         A. If the circuits of modules (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$ are four separated circuits, replace the four modules with the 2 × 2 combining patch.
3. Output the circuit design.

Fig. 20. Algorithmic overview of the pseudo-fractal combiner.

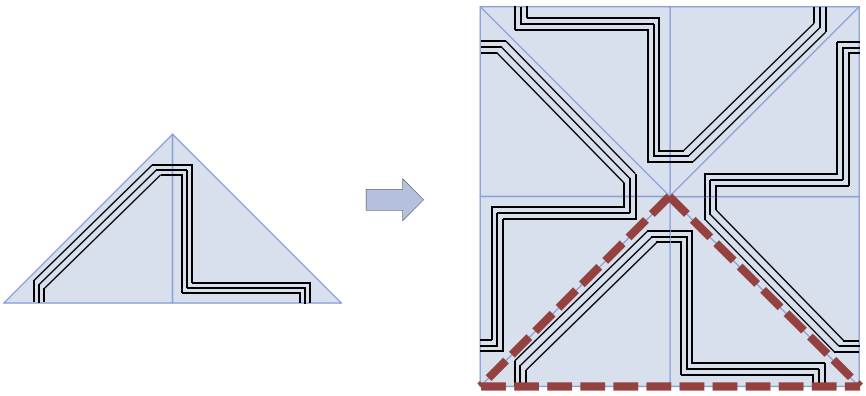


Fig. 21. A pseudo-fractal combining circuit and a 2 × 2 pseudo-fractal combining patch.

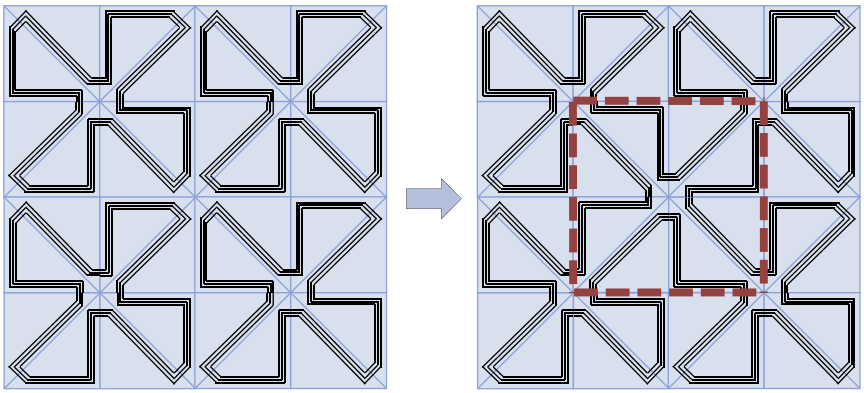


Fig. 22. Combining four separated circuits by a pseudo-fractal combining patch.

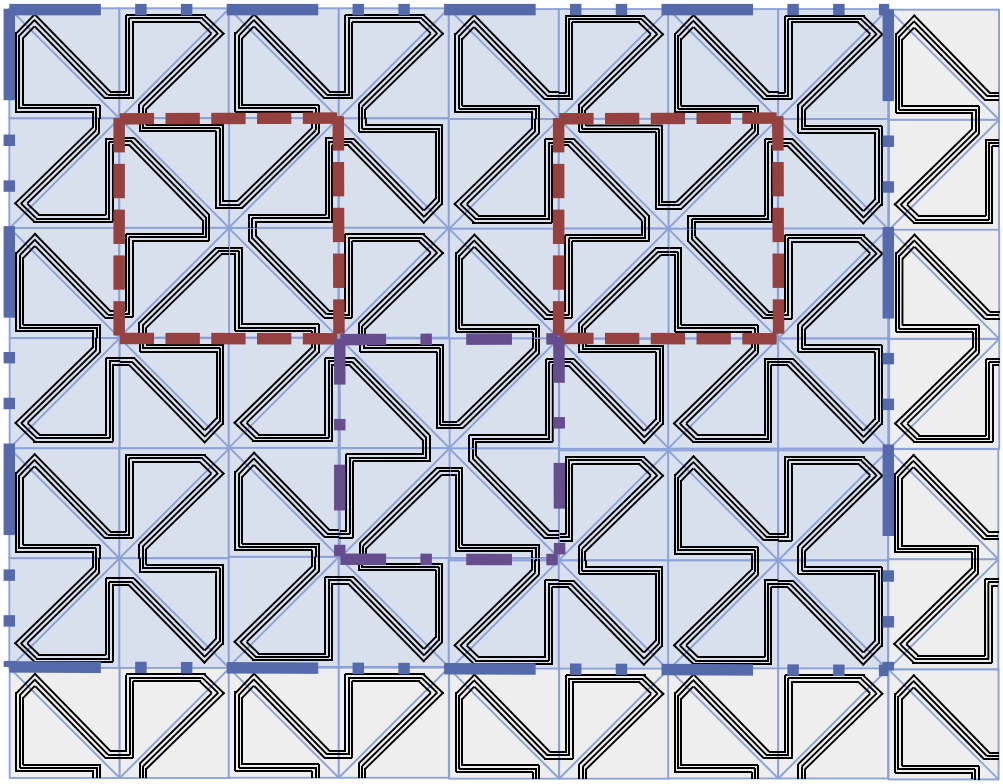


Fig. 23. Example of pseudo-fractal combining.

controller unit on the third module is disconnected from the circuit after patching, the circuit stays connected to each unit with an actuator, because the bottom edge is on the bottom outline of the circuit design and no actuator is on this edge. For the same reason, the circuit in Fig. 26 (bottom-right) also connects to each unit with an actuator. The output circuit of the 1-to-1 combiner connects to each unit with an actuator (Fig. 24).

The algorithm runs in $O(p \times q)$ time and $O(p+q)$ space where p is the size of the input circuit and q is the size of the input combining circuit.

6.3. 2-to-2 combiner

The 2-to-2 combiner combines two separated circuits. Fig. 27 shows a 2-to-2 combining circuit and 2-to-2 horizontal and vertical combining patches.

If two circuits on four modules face each other as shown in Fig. 28, the circuits are separated, and one of the two circuits is a closed circuit, then the patch can combine the two circuits into a connected circuit (Theorem 2). We call the two such circuits 2-*to*-2 *combinable circuits*.

More precisely, 2-*to*-2 *horizontal combinable circuits* are the two circuits on four modules (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$, where the circuits on (i, j) and $(i, j+1)$ are connected and the circuits on $(i+1, j)$ and $(i+1, j+1)$ are connected as shown in Fig. 28(a) (left). 2-*to*-2 *vertical combinable circuits* are the two circuits of four modules (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$, where the circuits on (i, j) and $(i+1, j)$ are connected and the circuits on $(i, j+1)$ and $(i+1, j+1)$ are connected as shown in Fig. 28(b) (left).

For each group of modules (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$, if the circuits on the modules are 2-to-2 combinable circuits, then the combiner joins the circuit into one combined circuit (Theorem 2).

Fig. 29 shows an example of the 2-to-2 combining. The blue dash–dot–dot line rectangular area is the input circuit design, and the red dash line rhombuses are the patched circuits.

Theorem 2. *Given a circuit design and its* 2-*to*-2 *combining patches, iff the circuits on modules* (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$

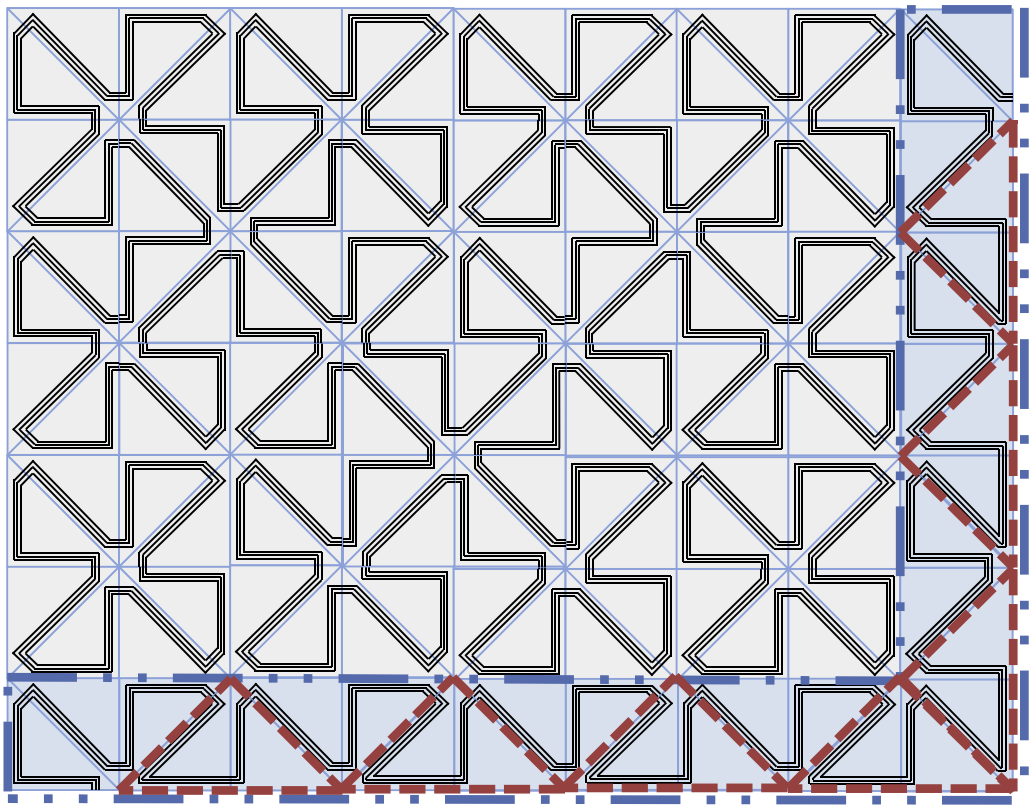


Fig. 24. Example of 1-to-1 combining.

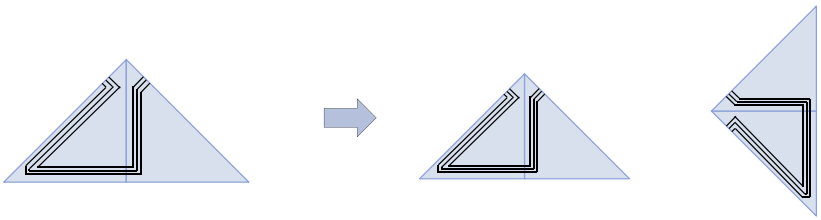


Fig. 25. A 1-to-1 combining circuit and two 1-to-1 rotated patches.

are 2-to-2 combinable circuits, then the patch combines the circuits into one connected circuit.

Proof. We use $C_{(i,j)}$ to denote a circuit crossing over a module (i, j). We use $(i, j)'$ to denote a module combined with the module of a patch.

If the circuits on modules (i, j), $(i + 1, j)$, $(i, j + 1)$, $(i + 1, j + 1)$ are 2-to-2 horizontal combinable circuits, $C_{(i,j)}$ and $C_{(i,j+1)}$ are connected and $C_{(i+1,j)}$ and $C_{(i+1,j+1)}$ are connected. After patching, $(i, j)'$ and $(i+1, j)'$ are connected, and $(i, j+1)'$ and $(i+1, j+1)'$ are connected. Because one of the circuits is closed, $(i, j)'$ and $(i, j+1)'$, or $(i + 1, j)'$ and $(i + 1, j + 1)'$ are connected. Thus, $C_{(i,j)'}$, $C_{(i+1,j)'}$, $C_{(i,j+1)'}$, $C_{(i+1,j+1)'}$ are a connected circuit.

For the same reason, if the circuits on modules (i, j), $(i + 1, j)$, $(i, j + 1)$, $(i + 1, j + 1)$ are 2-to-2 vertical combinable circuits, they become a connected circuit after patching.

If two circuits facing each other are not 2-to-2 combinable circuits because they are one connected circuit, after patching, the circuit becomes two separated circuits. Fig. 31 shows how one connected circuit is separated by patching.

We use C_a and C_b to denote two circuits. Let a be two modules including C_a. Let b be two modules including C_b. Let a' be two patched modules of a. Let b' be two patched modules of b. After a is patched by a', the two ends of $C_{a'}$ are connected via C_b. However, after b is replaced by b', the two ends of $C_{b'}$ are disconnected. Thus the ends of $C_{a'}$ must be disconnected.

If two circuits C_a, C_b facing each other are not 2-to-2 combinable circuits because both circuits are opened, after patching, the circuits become two separated circuits. Since C_a, C_b are opened, after a, b are replaced by a', b', the two ends of $C_{a'}$ on a' are disconnected and the two ends of $C_{b'}$ on b' are disconnected. There is no way to connect the ends of $C_{a'}$. Thus, the circuits are not one connected circuit. Theorem 2 is true. □

Fig. 30 shows an overview of the 2-to-2 combiner. In the first step, the algorithm generates two 2-to-2 patches. In the second step, the algorithm combines 2-to-2 combinable circuits until there is no 2-to-2 combinable circuit. In the third step, the algorithm outputs the circuit design.

The algorithm runs in $O(p \times q)$ time and $O(p+q)$ space where p is the size of the input circuit and q is the size of the input combining circuit.

The algorithm always constructs the correct design because there is at most one open circuit (from the 1-to-1 combiner), all other circuits are closed circuits (from the 2-to-2 combiner), all the separated circuits facing each other are 2-to-2 combinable circuits (Theorem 2) and the separation of the two circuits can be detected by the *PATH* algorithm.

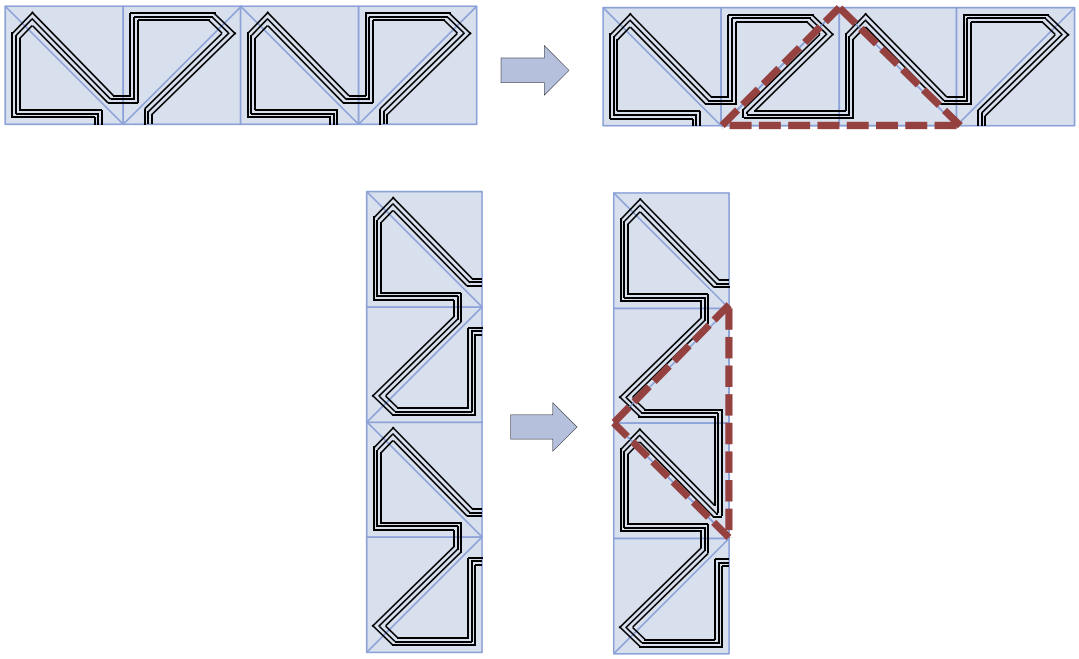


Fig. 26. Combining two separated circuits by 1-to-1 patching.

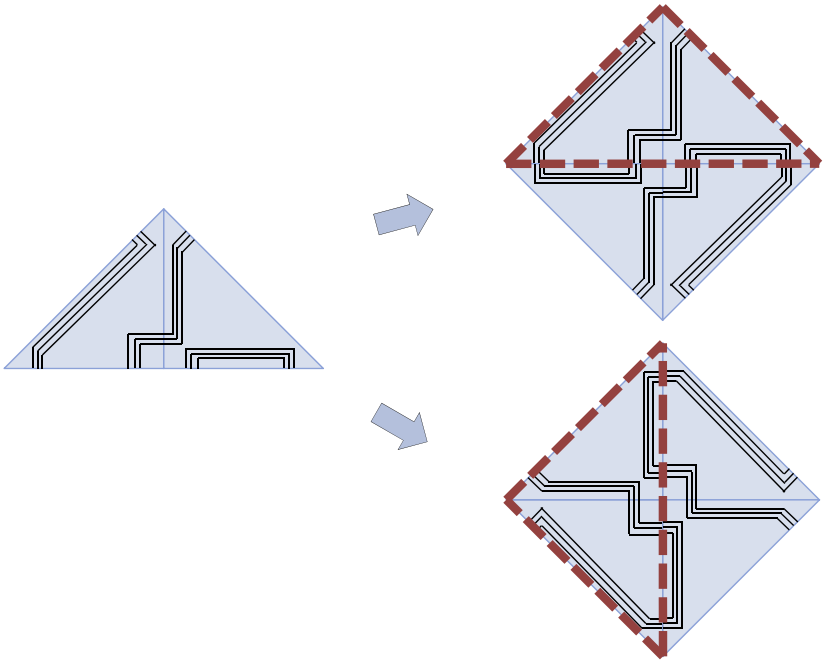


Fig. 27. A 2-to-2 combining circuit and 2-to-2 horizontal and vertical patches.

7. Sticker programming algorithm

Given k target shapes, an actuator model and a sticker place design, the sticker programming algorithm generates a sticker program; a sticker program is composed of a sticker design and an actuation sequence of the target shapes. Fig. 32 shows an overview of the sticker programming algorithm.

7.1. Origami planner

Given multiple target shapes, the origami planner generates an optimized origami plan for the target shapes. For each shape, the planner determines the sequence of folds required to achieve the shape. Fig. 33 shows an example of an origami plan for a space shuttle shape and a hat shape. Details about the origami planner are presented in [1].

7.2. Sticker compiler

Given the group information of an origami plan and an actuator model, the sticker compiler generates a sticker object and an actuation sequence. Fig. 34 shows the five step process overview.

7.2.1. Generating sticker object

The first step (Step 1 in Fig. 34) is to convert all angles of the plan to their corresponding actuator codes (Section 3.3.2). For example,

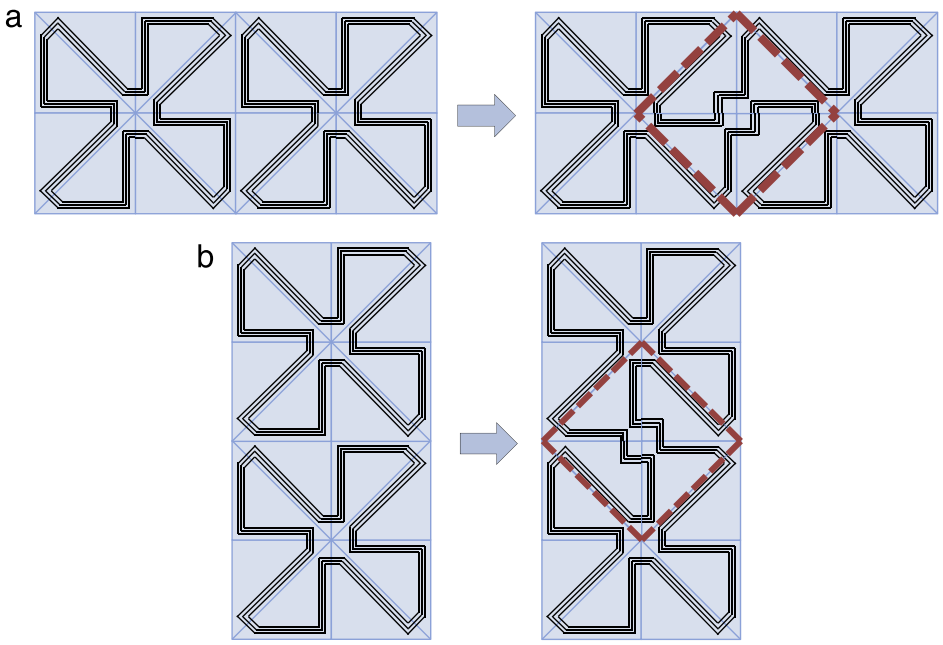


Fig. 28. Combining two separated circuits by 2-to-2 patching.

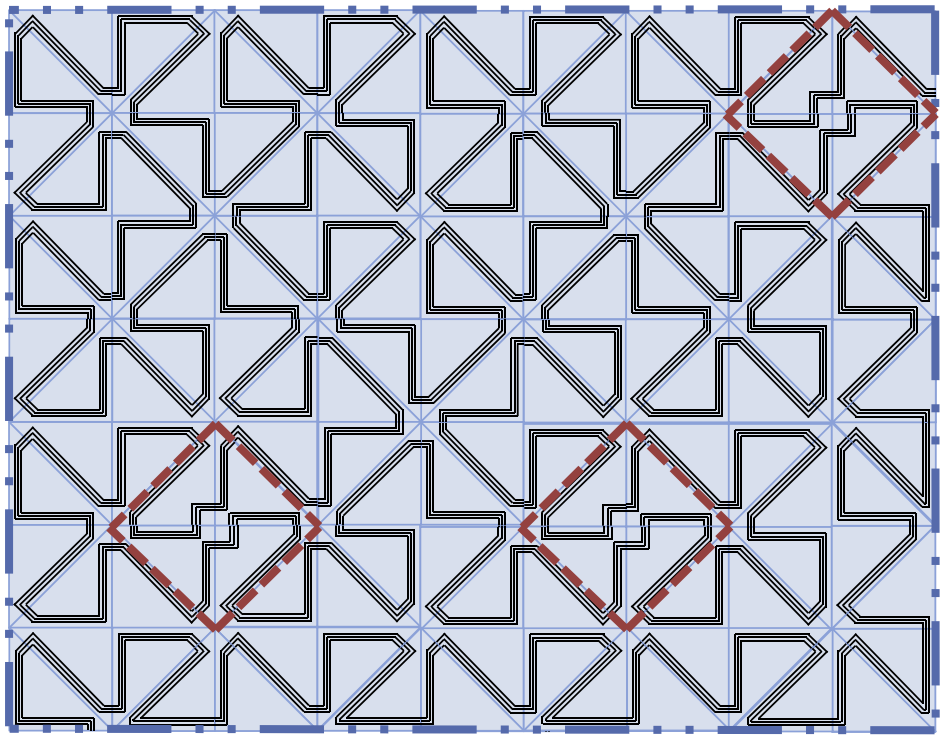


Fig. 29. Example of 2-to-2 combining.

2-to-2 Combiner

1. Given a 2-to-2 combining circuit, construct two 2×2 combining patches (Fig. 21).
2. Given a circuit design, combine 2-to-2 combinable circuits, until there is no combinable circuit.
3. Output the circuit design.

Fig. 30. Algorithmic overview of the 2-to-2 combiner.

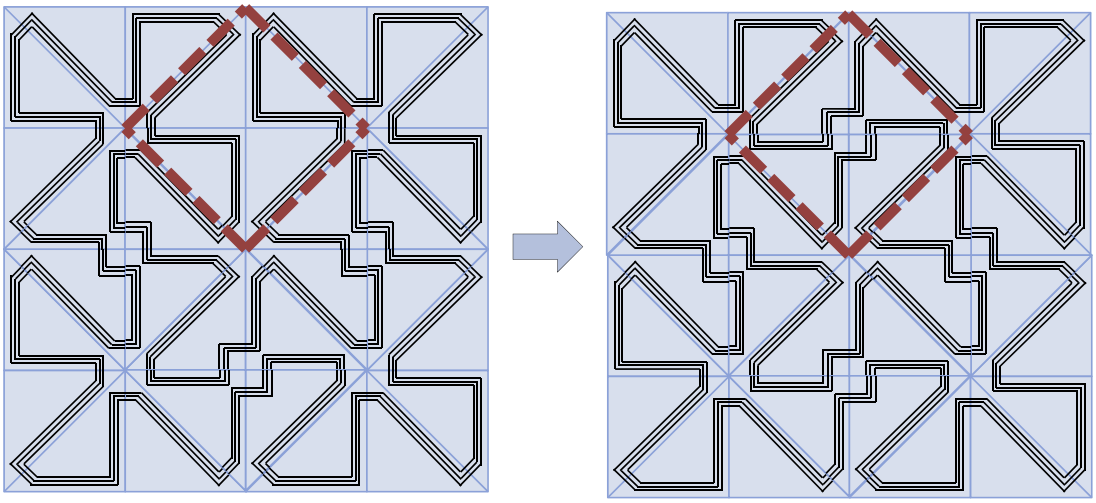


Fig. 31. Placing the 2-to-2 patch on the already connected circuit (left). The circuit becomes two circuits: inside and outside circuits (right).

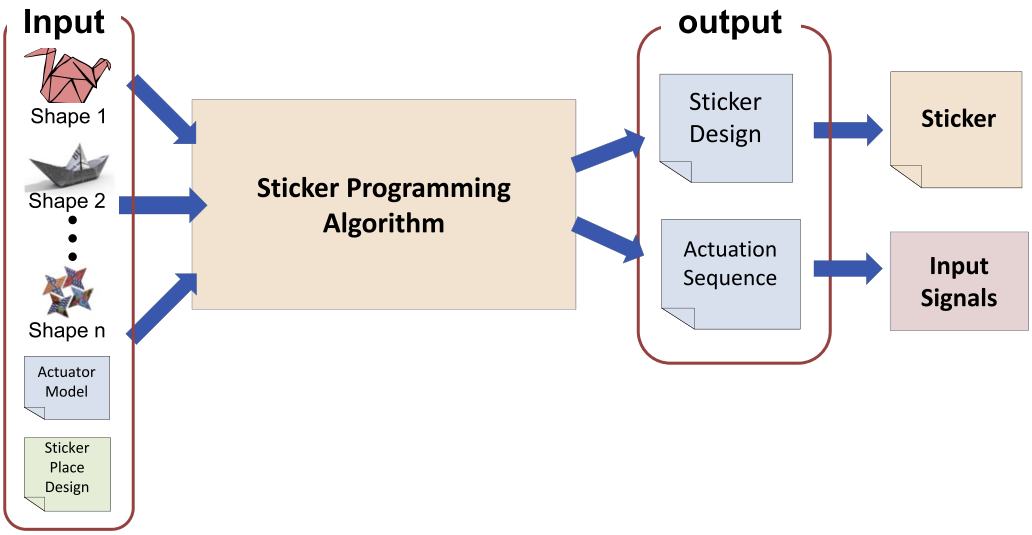


Fig. 32. Visual overview of sticker programming algorithm.

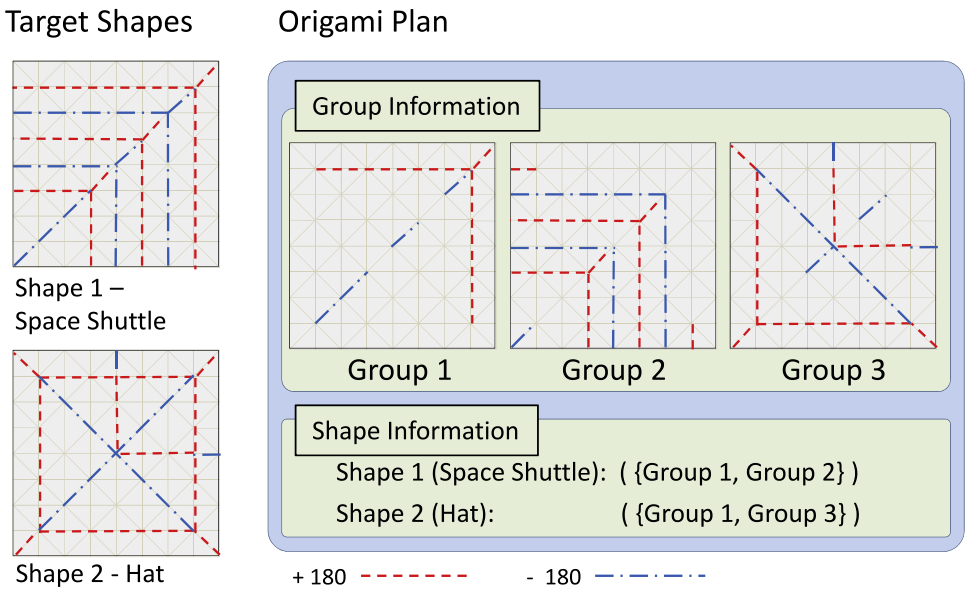


Fig. 33. (Left) Input target shapes for the 8 × 8 sheet. (Right) Origami plan generated by the origami planner. According to Shape Information, Group 1 ∪ 2 is space shuttle shape and Group 1 ∪ 3 is hat shape.

Sticker Compiler

1. Given the group information of an origami plan and an actuator model, convert each actuation angle of the plan to its corresponding actuator code.
2. Combine the actuator codes of all groups of each edge.
3. Construct a sticker object by collecting the combined actuator codes (Fig. 35).
4. Construct an actuation sequence by converting the folding sequence of each shape (the vectors of the shape information of the origami plan) into a sequence of the actuator codes (input signals).
5. Output the sticker object and the actuation sequence.

Fig. 34. Algorithmic overview of the sticker compiler.

Constructing Sticker Object (Step 3)

- Given the converted actuator codes of each edge (From step 2).
- Let $ac_{(i,j)-(k,l)}$ be a converted actuator code on an edge $(v_{(i,j)}, v_{(k,l)})$, where $v_{(i,j)}$ is a column i, row j vertex.
- Let SO be a sticker object.
- Let $sou_{i,j}$ be a sticker object unit of SO, where i is a column and j is a row.
- For each column i and row j:
  1. If $i = odd$ and $j = odd$, $sou_{i,j} \leftarrow (ac_{(i,j)-(i,j+1)}, ac_{(i,j)-(i+1,j+1)}, ac_{(i,j+1)-(i+1,j+1)})$.
  2. If $i = even$ and $j = odd$, $sou_{i,j} \leftarrow (ac_{(i,j)-(i+1,j)}, ac_{(i+1,j)-(i,j+1)}, ac_{(i,j)-(i,j+1)})$.
  3. If $i = odd$ and $j = even$, $sou_{i,j} \leftarrow (ac_{(i,j+1)-(i+1,j+1)}, ac_{(i+1,j)-(i,j+1)}, ac_{(i+1,j)-(i+1,j+1)})$.
  4. If $i = even$ and $j = even$, $sou_{i,j} \leftarrow (ac_{(i+1,j)-(i+1,j+1)}, ac_{(i,j)-(i+1,j+1)}, ac_{(i,j)-(i+1,j)})$.

Fig. 35. Details of Step 3 of the sticker compiler (Fig. 34).

in Fig. 33 for the top-left modules (1, 1) of groups 1, 2 and 3, each diagonal edge of group 1 and 2 is angle 0° and the diagonal edge of group 3 is angle +180°. If the actuating model is $A(00) \rightarrow 0°$ and $A(01) \rightarrow +180°$, the algorithm converts the angle 0° of groups 1 and 2 to a code 00 and the angle +180° of group 3 to a code 01.

The second step is to combine all actuator codes of each edge into the combined actuator codes for the sheet. For example, if the diagonal joints of the top-left module (1, 1) of the groups 1, 2, 3 contain codes 00, 00, 01, respectively, after the second step, the diagonal edge of the combined group contains a code 000001; the combiner simply attaches the three codes 00, 00, 01.

The third step is to construct a sticker object by collecting the combined actuator codes of the edges. After the second step, each edge contains a combined code. As we explained in Section 3 and Fig. 8, each module includes a sticker controller unit group. The group is composed of three sticker controller units that control the actuators on the left, diagonal, and bottom edges. The algorithm collects the codes on the edges and completes the sticker object which is an $n \times m$ matrix. Each element of the sticker object is a three-tuple. Each three-tuple contains three elements for the three sticker controller units of each module. For instance, after the second step, the left, diagonal, and bottom edges of the module (1, 1) contain codes 000000, 000001 and 000100, respectively. The algorithm collects the left edge code to the first element of the three-tuple, the diagonal edge code to the second element and the bottom edge code to the third element. Then, the element (1, 1) of the matrix, which the compiler generates, is (000000, 000001, 000100). The details of this step are in Fig. 35.

7.2.2. Generating actuation sequence

In the fourth step, the sticker compiler converts the shape information of an origami plan into an actuation sequence by replacing the group names by binary codes.

An actuation sequence is a set of pairs (*shape name*, *input signal vector*). Each pair represents a shape. Each vector contains k elements where the sheet transforms into the shape in k steps. Each element of the vector is a binary code that is the input signal of self-folding sheets.

Each bit of the binary code represents each group. For example, when an origami plan having 5 actuator groups is achieved by the actuators of groups 1 and 3, the algorithm generates a 5-bit code (10100) as an actuation sequence. When another shape is achieved by the actuators of groups 1 and 3, followed by the actuators of groups 4 and 5, the algorithm generates a vector with two 5-bit codes (10100, 00011).

Fig. 33 shows an origami plan for two shapes. If we input the shape information of the plan into the compiler, the output is {(*shape*1, (110)), (*shape*2, (101))}. Since the plan has three groups, each code of the actuation sequence is three bits. The actuators of groups 1 and 2 simultaneously fold for shape 1 while the actuators of groups 1 and 3 fold for shape 2. The compiler generates vectors (110) and (101) for shape 1 and 2, respectively, and constructs the output.

Because each bit of an actuation sequence represents each group of each phase of each shape of an origami plan, the algorithm correctly transforms the shape information of the origami plan into the actuation sequence. The sticker controller generates a sticker

Table 1
Overview of 4 × 4 and 8 × 8 self-folding sheets.

	4 × 4 sheet	8 × 8 sheet
Crease pattern	4 × 4 box-pleated	8 × 8 box-pleated
Size	96 mm × 96 mm	192 mm × 192 mm
# of edges	40	176
# of actuators	40	36
Current	1.5 A	5.0 A
Average folding time	21.6 s	5.0 s
Sticker controller model	Sticker controller model	Socket controller model
Reprogramming	Very easy	Easy
Actuator	Y-type actuator	Y-type actuator
Programming	By sticker programming algorithm	By sticker programming algorithm

object and an actuation sequence in $O(N^2)$ time and $O(N)$ space, where N is the size of the input.

7.3. Sticker linker

Given a sticker object and a sticker place design, the sticker linker generates a sticker design. Fig. 36 shows an overview of the sticker linker algorithm.

Sticker objects, generated by the sticker compiler, contain the abstracted information of sticker designs. With the abstracted information, they can transform into the sticker designs for various manufactured self-folding sheets.

As each sticker place of a sticker place design has an address (Section 3.6), each alphabet (bit) of a sticker object has an address. The address of the alphabet w is a 4-tuple (sid, a, b, c), where $sid \in \{l(i, j), d(i, j), b(i, j)\}$ is the address of the sticker controller unit, a is the input port, b is the output port, and $c \in \{0, 1\}$ is the value of w. The address of the alphabet contains its value in the fourth element. This address format is the same as the format for the sticker place design (Fig. 37).

The sticker linker generates a sticker design by copying the selected sticker places of the sticker place design. For each bit, if the address of the bit and the address of the sticker place (graphic image) are the same, the linker copies the selected sticker place to the generating sticker design. Fig. 37 shows an example of the linking process. The second element of the top-right tuple of the input sticker object (1) contains 01 and each bit has the address (a), (b). Since the address (b) matches the address of the sticker place (d), the linker copies the sticker place (d) to the sticker design (f). However, since the address (a) does not match any sticker place of the sticker place design (2), the sticker linker keeps the sticker place of the sticker place design (3) as an empty place (e); (a) and (c) are different due to the fourth element of each address. By copying the matched sticker place to the sticker design, the linker constructs the sticker design (3).

Let p be the size of the sticker object and let q be the size of the sticker place design. For each address of the sticker object, if a sticker place having the same address exists, the algorithm finds and copies it in $O(q)$ time. If no sticker place having the same address exists, the algorithm knows it in $O(q)$ time. Thus, the linking algorithm runs in $O(p \times q)$ time and $O(p + q)$ space.

8. Physical self-folding devices

We built the 4 × 4 and 8 × 8 self-folding sheets. Table 1 shows the overview of the sheets. The 4 × 4 sheet was used to evaluate the low-level self-folding control using straight-line folding and diagonal folding. The 8 × 8 sheet was used to evaluate the self-folding control using two complex shapes: a space shuttle and a hat.

Self-folding sheets are composed of four parts: a box-pleated structure, actuators, a sticker controller, and a sticker program. Since one of our visions is for the self-reconfiguration sheet to be used in various environments including space shuttles, offices, and houses, we selected most of the materials that are commonly available. Our current sheets are built with lamination film tiles, paper joints, and copper sheets, while our previous self-folding sheet [2] was built with glass-fiber tiles, silicon compound joints, and copper–kapton films. The actuators for the current sheets and our previous sheet are built with SMA sheets.

8.1. Box-pleated structure

The box-pleated structure is the kinematic structure of self-folding sheets (Fig. 40). The structure is composed of three layers: a lamination film, paper, and a lamination film. Its materials are lamination films (Heatseal, 0.7 mil), an anti-aging paper (Staples, 32 lb, 633 213), and micro bolts and nuts (Scale Hardware, 0.5 mm) and they are cut by Versalaser Cutting System. To attach the layers, we stack them and put them into a laminator (GBC, HeatSeal H425 Laminator). We use the micro bolts and nuts to align the layers (Fig. 40).

8.2. Actuators

The actuators of both self-folding sheets are Y-shape SMA actuators (Fig. 41), which are developed by modifying Z-type actuators in [27].

When current passes through the actuator, the actuator is heated and transforms into its annealed shape. This motion generates the folding force and the actuators fold the joints of the self-folding sheet. We unfold the actuators manually to set them back to their initial state.

8.3. Sticker controller

The sticker controller is composed of a circuit, sockets, a sticker (a set of connectors) and a signal interface (Fig. 38).

8.3.1. Circuit

The circuit is manufactured in the single layer circuit model (Section 3.3.3) and its material is copper tape (McMaster-Carr, 76555A716). The sockets and the stickers are also made of this copper tape.

To generate two circuit designs, we followed each step of the self-folding sheet designing algorithm (Section 5). Each circuit design has a fractal pattern, which is an output of the pseudo-fractal combiner for a $2^k \times 2^k$ sheet.

8.3.2. Socket

A socket connects a circuit and an actuator as shown in Figs. 41 and 38. The socket of our device is made of copper tape and is connected with the circuit; the circuit and the sockets are one metal piece. The socket is peanut shaped and is composed of two circles which cover the top and bottom sides of the leg of an actuator. We use 0.5 mm bolts and nuts to attach the actuators.

8.3.3. Sticker

In the model, a sticker is a rectangular adhesive tape embedding a set of connectors (Section 3). By covering a sticker upon a self-folding sheet, the connectors of the sticker fill selected sticker places on the sheet. The sticker programming algorithm constructs the design of the sticker. For our experiments, we added and removed the connectors according to the sticker design. Each connector is a 2 mm × 5.6 mm copper tape piece. The enabling and disabling sticker places are as shown in Figs. 41, 39 and 38.

The origami planning software we have implemented [1] generates the origami plans. We convert the plans to the sticker designs by following the steps of the sticker compiler and linker (Section 7).

Sticker Linker

1. For each alphabet (bit) of a given sticker object,
   (a) Construct the address of the alphabet.
   (b) Given the sticker place design, find a sticker place with the address.
   (c) Copy this sticker place to the sticker design.
2. Output the sticker design.

Fig. 36. Algorithmic overview of the sticker linker.

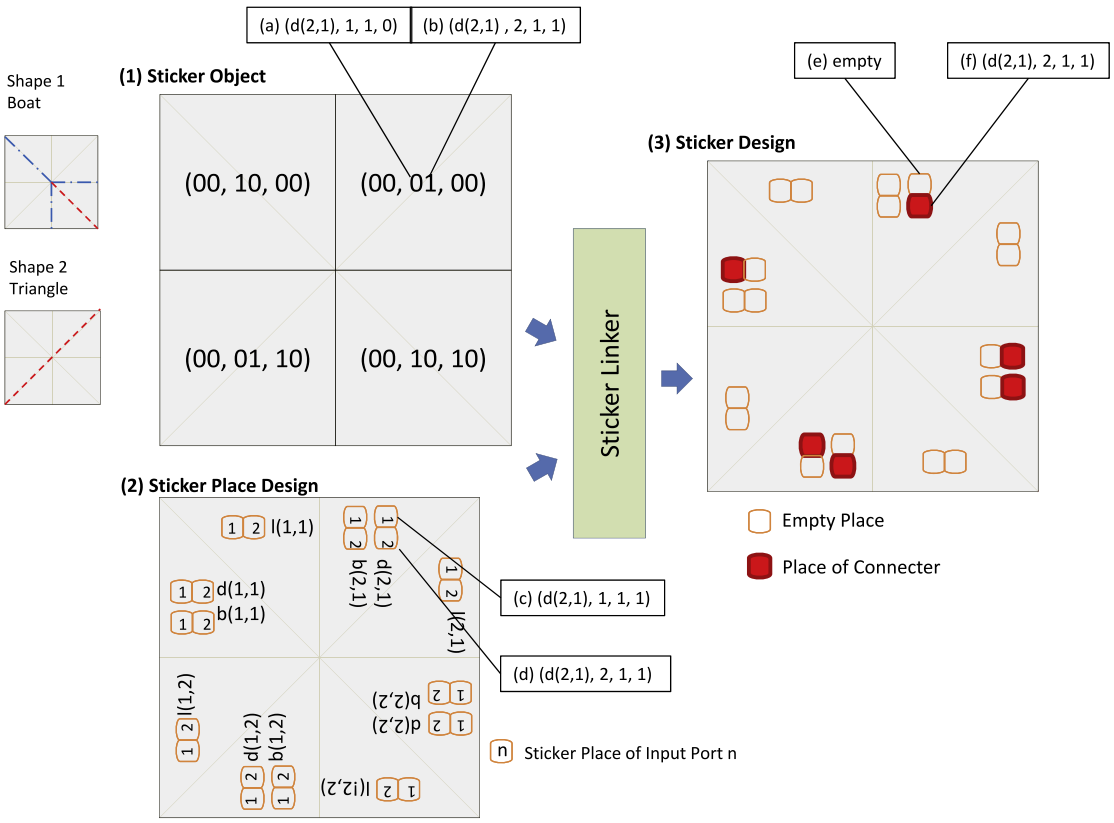


Fig. 37. Example of sticker linking.

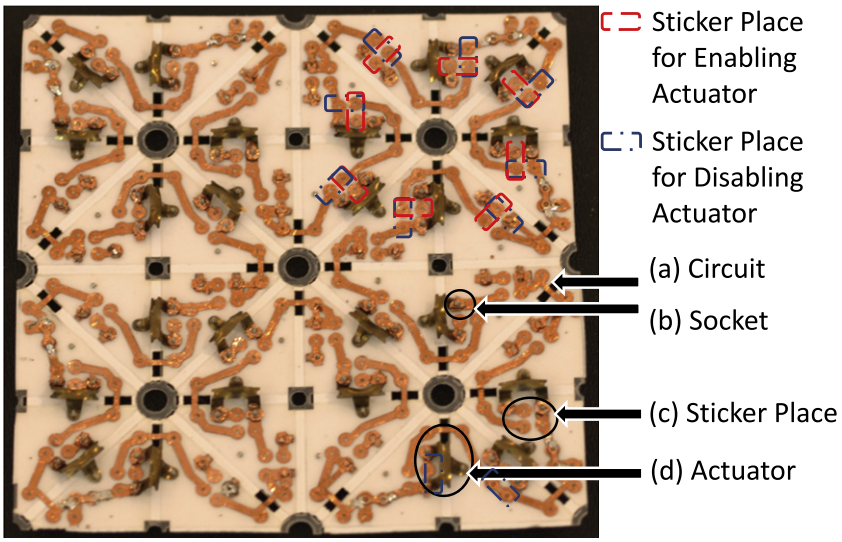


Fig. 38. 4 × 4 sticker controller with no sticker.

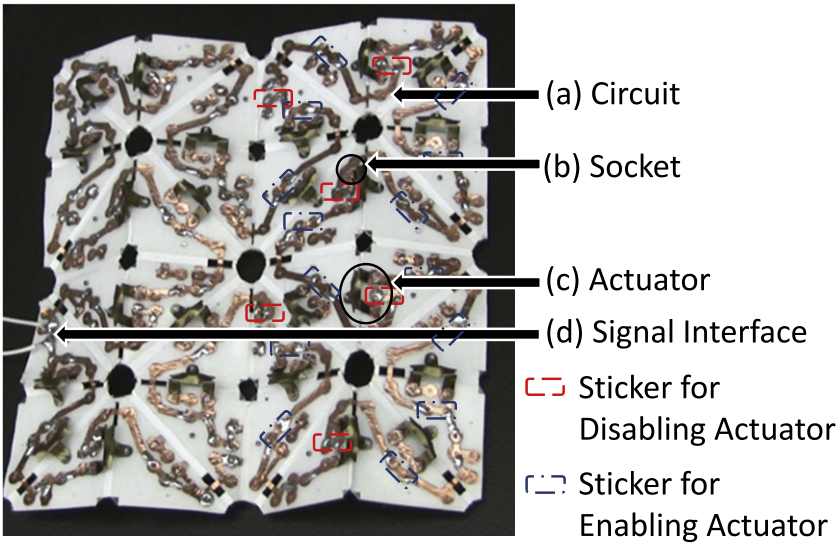


Fig. 39. 4 × 4 sticker controller with a sticker (vertical folding program).

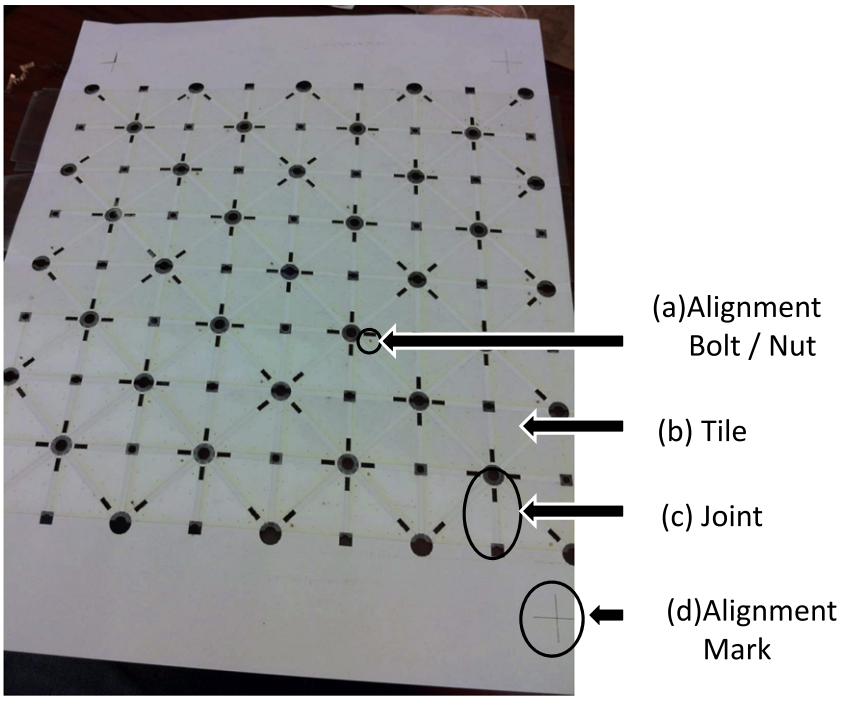


Fig. 40. Box-pleated structure.

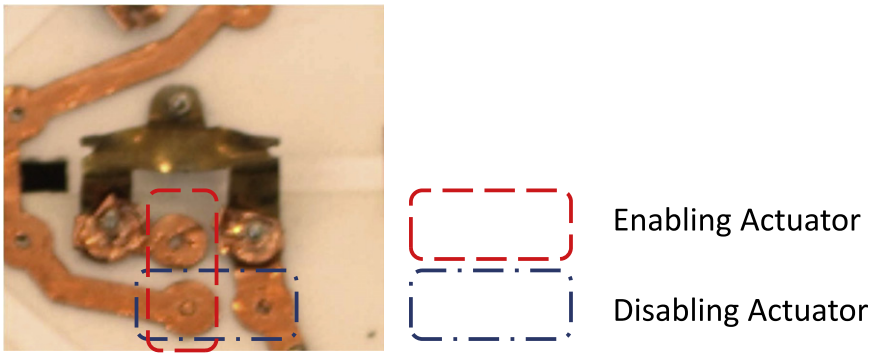


Fig. 41. Y-type actuator and two sticker places.

Table 2
Origami planning time for vertical and diagonal folding.

Analysis time for vertical	3.6 s (3600 ms)
Building time for vertical	17 ms
Analysis time for diagonal	4.2 s (4200 ms)
Building time for diagonal	16 ms
CPU	Intel core 2 quad 2.83 GHz (Q9550)
Storage	3 GB RAM, Seagate 750 GB 300 MBps 7200 rpm HDD
Graphics	NVIDIA quadro FX 1700

Table 3
Actuators (ac.) of 4 × 4 sheet.

	Folding ac.	Total ac.	Total edges	Folding ac./total ac.	Total ac./total edges
Vertical	12	40	40	30.0%	100.0%
Diagonal	8	40	40	20.0%	100.0%
Total	20	40	40	50.0%	100.0%

Table 4
Folding time and current of 4 × 4 sheet.

	# of runs	Current (A)	Average folding time (Standard deviation)
Vertical	14	1.5	21.0 s (2.8 s)
Diagonal	13	1.5	22.4 s (4.7 s)
Total	27	1.5	21.6 s (3.7 s)

8.3.4. Signal interface

A sticker controller receives input signals through its signal interface (Section 7). The 4 × 4 sheet device controls one actuator group and its input interface has one input port (Fig. 39). The input interface of the 8 × 8 sheet device has three input ports.

9. Experiment with the 4 × 4 self-folding sheet

This experiment is performed to understand the programmability and the fundamental folding ability of a self-folding sheet with the sticker programming. We ran the 4 × 4 sheet device with the sticker programs for two basic folding motions, vertical and diagonal (Fig. 42 and Movie S1). This experiment was executed according to the following steps:

1. construct two sticker designs for the vertical folding motion and the diagonal folding motion;
2. attach a set of connectors (a sticker) on the sheet according to the sticker design for the vertical folding motion;
3. run the sticker program on the sheet;
4. remove all connectors;
5. attach the connectors according to the sticker design for the diagonal folding motion;
6. run the sticker program on the sheet.

9.1. Results and discussions

The sticker programming algorithm generates two sticker designs (Section 7). Table 2 shows the planning times.

In our sticker controller model, each edge contains one actuator. Since each edge of the 4 × 4 sheet contains one actuator, the sheet has 40 edges and 40 actuators. 50.0% of the actuators were used (Table 3).

We executed the vertical folding program on the 4 × 4 self-folding sheet 14 times and we then executed the diagonal folding program 13 times. The 4 × 4 sheet achieved the vertical and diagonal folding reliably (Fig. 42). We set the current at 1.5 A and the average folding time of the shapes was 21.6 s (Table 4).

Table 5
Folding angle and folding achievement of 4 × 4 sheet.

	Average folding angles (Standard deviation)	Target angles	Folding achievement (folding angle/target angle)
Vertical	141.6°(10.1°)	180.0°	78.7%
Diagonal	126.4°(19.7°)	180.0°	70.2%
Total	134.0°(16.3°)	180.0°	74.5%

Table 6
Failure of 4 × 4 sheet.

	# of runs	# of failures	Average failure
Vertical	14	1 (of 14 runs)	0.7 (of 10 runs)
Diagonal	13	2 (of 13 runs)	1.5 (of 10 runs)
Total	27	3 (of 27 runs)	1.1 (of 10 runs)

Table 7
Disabled actuators (ac.) of 4 × 4 sheet.

	Average # of disabled ac.	Folding ac.	Disabled ac./folding ac.	Disabled ac./total ac.
Vertical	0.77	12	6.4%	1.9%
Diagonal	1.36	8	17.0%	3.4%
Total	1.04	10	10.4%	2.6%

While each actuator in the model folds its joint into 180°, the actuators on the real device achieved 134.0°, which is 74.5% of the target angle 180.0° (Table 5); we measured the folding angle of the edges on the center line of each shape. The individual actuator, which was not on the sheet, achieved 180° folding and each edge without an embedded actuator was folded into 180°. However, the actuator on the edge could not achieve 180° folding because it could not generate enough force to fold the edge into 180°.

Each center line of both diagonal and vertical folding shapes was folded by 4 actuators. While the diagonal center line was 41.4% longer than the vertical center line, the angle of the diagonal center line was 10.7% smaller than the angle of the vertical center line.

We ran the sticker programs 27 times. We defined a success round as when the sticker controller delivered the signals to the selected actuators and a failure round as when the sticker controller did not deliver the signals. The average folding angle of the success rounds was 134.0° (Table 5).

Although the 4th round of the vertical folding and the 1st and 9th rounds of the diagonal folding were failure rounds (Table 6), we were able to continue the experiment by disabling actuators. The main reason for the failures was weak connections between the socket and the SMA actuator. The SMA is a material that is hard to solder. We thus made the electronic connection with conductive micro bolts and nuts. However, while the sheet folded several times, the electronic connection became weak, which made the socket difficult to recover. In this case, we fixed the device by disabling (removing) the broken actuators; we also changed the position of the connector to pass the signals. For each failure, we disabled one actuator and the sheet continued its transformation reliably. The average number of disabled actuators was 1.04 (Table 7).

Most of the results of the two basic shapes on the 4 × 4 sheet were similar except their resistances. The resistance was 19.1 Ω for the vertical folding while the resistance was 28.9 Ω for the diagonal folding. Even though the number of selected actuators for the diagonal folding is 1.5 times smaller than the number for the vertical folding, the resistance of the sheet increased 1.51 times (Table 8).

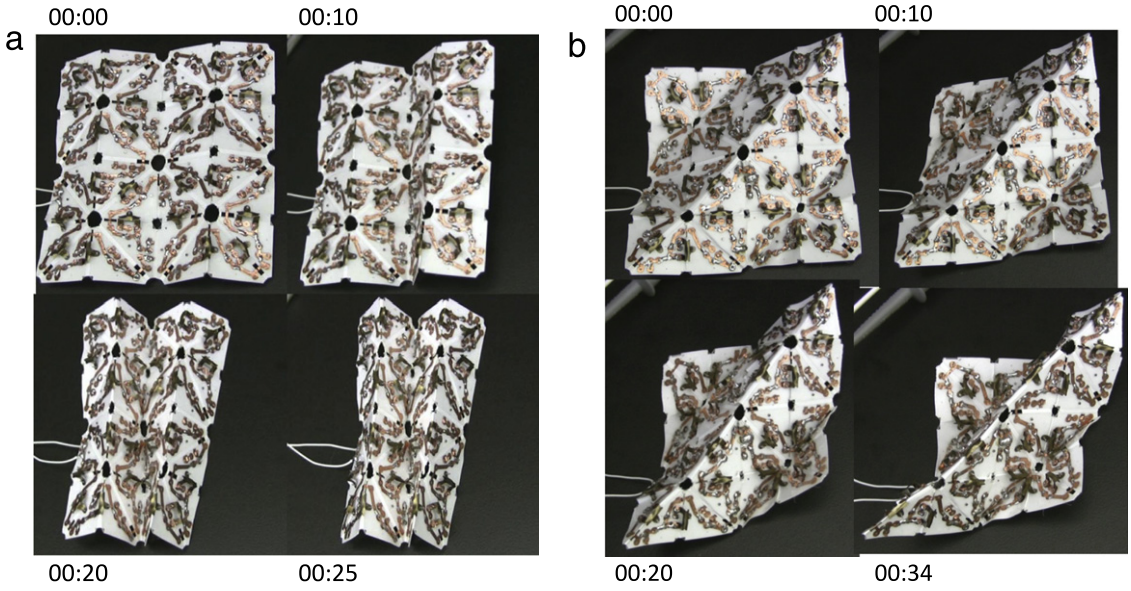


Fig. 42. Snapshots from programming and controlling 4 × 4 self-folding sheet for diagonal folding (a) and vertical folding (b).

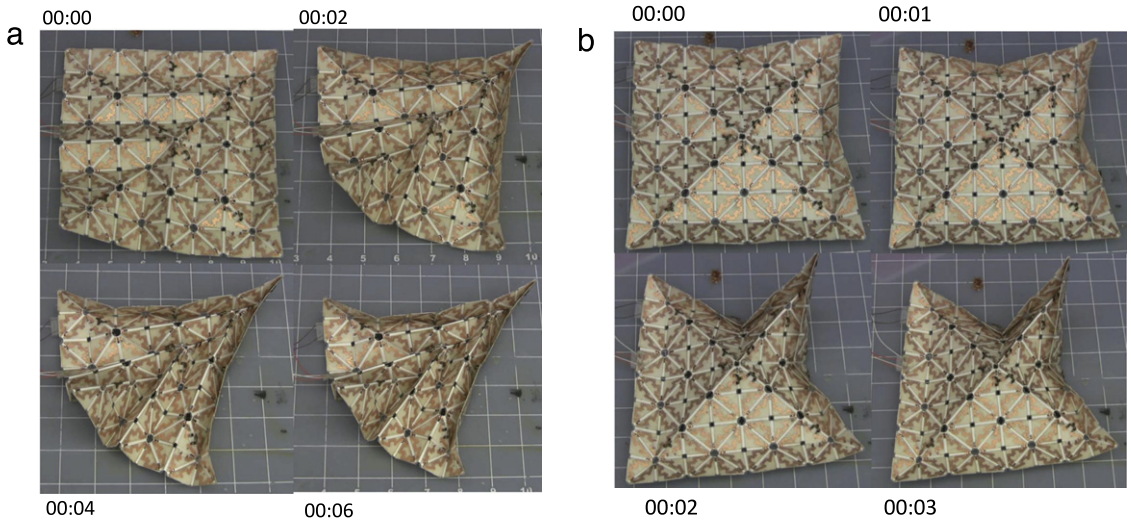


Fig. 43. Snapshots of the 8 × 8 self-folding sheet for the space shuttle shape (a) and the hat shape (b).

Table 8
Resistance of 4 × 4 sheet.

	Average resistance (Ω)
Vertical	19.1
Diagonal	28.9
Total	23.6

10. Experiment with the 8 × 8 self-folding sheet

We designed the experiment with the 8 × 8 sheet to evaluate the programmability and the scalability of the sticker programming as a new control method. In order to focus more on programming challenges than on mechanical challenges, we selected a space shuttle shape and a hat shape (Fig. 43 and Movie S1). The shapes require no double folded edges, three actuator groups and an 8 × 8 box-pleated crease pattern. The 8 × 8 pattern contains 4.4 times more edges than the 4 × 4 pattern.

The 8 × 8 sheet includes a socket controller (Table 1, Fig. 45), which is the implementation of an alternative sticker model (Section 3.5).

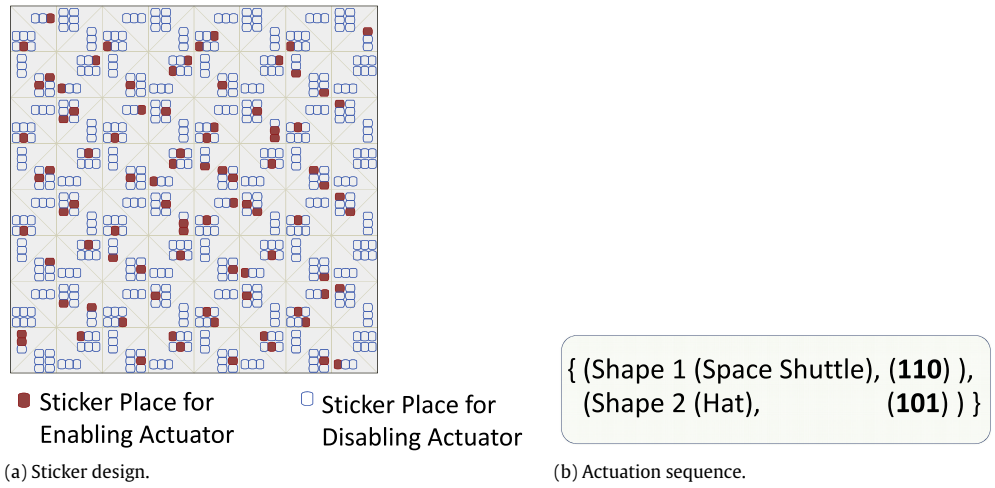
We executed the experiment according to the following steps:

1. generate a sticker design containing the space shuttle and hat shapes;
2. optimize the sticker design for the socket type sticker controller;
3. place actuators according to the sticker design;
4. run the sticker program for the space shuttle shape;
5. run the sticker program for the hat shape.

10.1. Results and discussions

We generated the sticker design for the two shapes with the sticker programming algorithm (Section 7). The sticker programming algorithm planned the folding of the two target shapes with the origami planner (Fig. 33), and the sticker compiler and the linker then transformed the plan into the sticker program (Fig. 44). In the actuator sticker model, the number of connectors was equal to the number of actuators (Section 3.5). By minimizing the number of actuators involved in the passive folding, we optimized the sticker design (Fig. 46). Table 9 shows the planning times.

Since we built the 8 × 8 sheet in the socket controller model (Section 3.5), the 8 × 8 sheet ran with the relatively small number



(a) Sticker design. (b) Actuation sequence.

Fig. 44. Result of the sticker programming algorithm for the 8 × 8 sheet. The input target shapes are the space shuttle and hat shapes (Fig. 33 (left)).

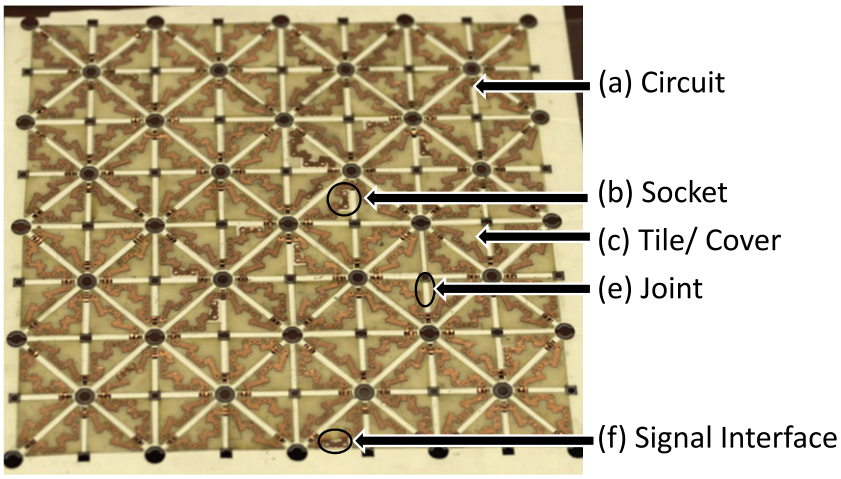


Fig. 45. 8 × 8 self-folding sheet without a sticker. For the controller of this sheet, a sticker is a set of actuators.

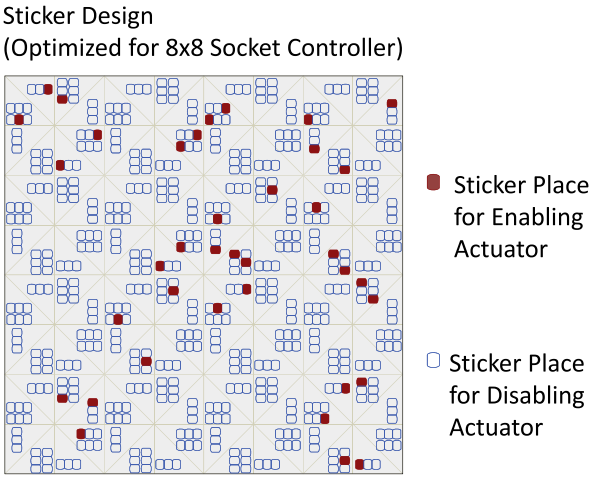


Fig. 46. Sticker design optimized for the socket controller of the 8 × 8 sheet. The actuation sequence for this design is shown in Fig. 44(b). The original sticker design is shown in Fig. 44(a). The target shapes are the space shuttle and hat shapes (Fig. 33 (left)).

of actuators (only 20.5% of edges have the actuators). The 8 × 8 sheet has 18.2% less actuators than the 4 × 4 sheet, while the edges of the 8 × 8 sheet are 4.4 times more than the edges of the 4 × 4 sheet (Tables 1 and 3). While the 8 × 8 sheet transformed into the shapes, 61.1% of the actuators were used (Table 10).

We executed the space shuttle shape folding (shape 1) 14 times and the hat shape folding (shape 2) 12 times. The 8 × 8 sheet achieved the shapes reliably with the optimized number of actuators (Fig. 43). We set the current at 5.0 A and the average folding time was 5.0 s (Table 11).

Table 9
Multiple origami planning time.

Analysis time for space shuttle	5.3 s (5300 ms)
Building time for space shuttle	19 ms
Analysis time for hat	4.9 s (4900 ms)
Building time for hat	17 ms
Building time for multiple origami plan	25 ms
Total time	10.3 s (10 261 ms)

Table 10
Actuators (ac.) of 8 × 8 sheet.

	Folding ac.	Total ac.	Total edges	Folding ac./total ac.	Total ac./total edges
S. shuttle	20	36	176	55.6%	20.5%
Hat	24	36	176	66.7%	20.5%
Total	36	36	176	100.0%	20.5%

Table 11
Folding time and current of 8 × 8 sheet.

	# of runs	Current (A)	Average folding time (Standard deviation)
Space shuttle	14	5.0	5.9 s (0.6 s)
Hat	12	5.0	4.5 s (0.7 s)
Total	26	5.0	5.0 s (1.5 s)

Table 12
Failure of 8 × 8 sheet.

	# of runs	# of failures	Average failure
Space shuttle	14	3 (of 14 runs)	2.1 (of 10 runs)
Hat	12	2 (of 12 runs)	1.6 (of 10 runs)
Total	26	5 (of 26 runs)	1.9 (of 10 runs)

Table 13
Disabled actuators (ac.) of 8 × 8 sheet.

	Average # of disabled ac.	Folding ac.	Disabled ac. /folding ac.	Disabled ac./total ac.
Space shuttle	0.82	20	4.1%	2.3%
Hat	0.80	24	3.3%	2.2%
Total	0.81	22	3.7%	2.2%

For this experiment, there were 5 failures; we defined success and failure rounds in Section 9. The transformation failed in the 3rd, 10th and 14th rounds for shape 1 and the 3rd and 12th rounds for shape 2.

The 3rd round for shape 1 and the 3rd round for shape 2 failed because the actuators broke. By replacing these actuators with metal pieces, we disabled the actuators. The edges with the disabled actuators folded passively, which allowed us to continue the experiments. In each 3rd round, one actuator broke. Since the broken actuator for shape 1 was in actuator group 2, the disabled actuator did not affect the folding for shape 2.

The 10th and 14th rounds for shape 1 and the 12th round for shape 2 failed due to the weak connections between the actuator sockets and the actuators. The structure of the sheet was built with lamination tiles and paper joints. Micro bolts were used to attach the actuators. While we ran the sheet around 10 times for each shape, these connections were too weak to continue the experiments. In the 10th and 14th rounds for shape 1, by tightening the bolts on the broken sockets, we could continue the experiments. In the 12th round of shape 2, however, we could not continue the experiment because while we tightened the bolts on the sockets to fix the connections, the connections of the other sockets became too weak to continue the experiment.

We found a couple of broken connections on the joints in the 14th rounds for shape 1 and the 12th round for shape 2. We fixed the connections by patching the metal pieces on the joints (Table 12).

The average number of disabled actuators was 0.81. The disabled actuators were 3.7% of the folding actuators and 2.2% of the total actuators (Table 13).

The resistance for shape 1 was 17.4 kΩ and the resistance for shape 2 was 80.15 Ω. While we executed shape 1, the resistance of group 3 was 1.71 MΩ. However, because group 3 was not used for shape 1, the sheet achieved the shape (Table 14).

Table 14
Resistance of 8 × 8 sheet.

	Average resistance Group1, Group2, Group3	Average resistance of folding groups
Space shuttle	44.8 Ω, 34.8 kΩ, 1.71 MΩ	17.4 kΩ (Group1 + Group2)/2
Hat	109.3 Ω, 14.7 kΩ, 51.0 Ω	80.15 Ω (Group1 + Group3)/2

11. Discussions

In this paper, we introduced a new control method for self-folding sheets. We discussed the details of the model and the algorithms that correctly design and control the self-folding sheet within polynomial time and space. We also implemented the 4 × 4 and 8 × 8 self-folding sheet devices. We designed the sheets with the automated sheet design algorithm and generated the sticker programs with the automated sticker programming algorithm. The sticker controller for each sheet successfully controlled the actuators and achieved our target origami shapes.

There are however gaps between the model and the physical devices. Although actuation does not fail to transform the model, the device can fail to transform; the 4 × 4 sheet failed 11% of the total runs and the 8 × 8 sheet failed 19% of the total runs. The main reason was weak connections between the actuator sockets and the actuators. The devices also had other disconnection issues, such as broken actuators or disconnected wires at the joints.

In the model, the folding speed is not a concern, but all shapes for the physical devices required time to complete the shapes. In our experiment, the folding speed is not a factor because the time is relatively short (4–25 s) and the shapes are simple enough to be folded in one step. However, for more complex shapes with many folding steps, we need a method which notices the ending time of each step. Adding an electronic route to collect the output of all folding feedback sensors can be a possible solution to this challenge.

The passive edges are not clearly defined for the alternative sticker model. Since the 4 × 4 sheet is in the basic sticker controller model, the sheet contains one actuator for each edge. On the other hand, the 8 × 8 sheet in the alternative sticker model contains some edges that are passively folded. While we experimented with the space shuttle shape, a couple of the edges at the end of the left wing were not folded in a few rounds. We marked them as success rounds because all actuators worked correctly; the definition of the success round is described in Section 10.

12. Conclusions and future works

This paper contains the hardware model and the algorithms for $n \times m$ self-folding sheets transforming into origami shapes. We presented and analyzed the model and the algorithms. We demonstrated and analyzed two sheet devices built with the model and the algorithms.

We believe this method can be applied for bigger sheets transforming into complex shapes with many steps. To achieve this goal, we are implementing the algorithms as a development tool for self-folding sheets. We are developing solutions to simplify the sticker attachment process and to manufacture sheet devices containing better mechanical features.

Acknowledgments

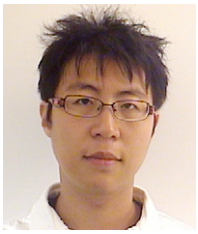
This work was funded in part by NSF grants 1240383 and 1138967. We are grateful for this support. We thank Prof. Jamie Baik, Rob Wood, Erik Demaine, Martin Demaine and Sangbae Kim for insightful discussions and feedback on this research.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.robot.2013.06.015.

References

[1] B. An, N. Benbernou, E. Demaine, D. Rus, Planning to fold multiple objects from a single self-folding sheet, Robotica, Special Issue on Robotic Self-X Systems 29 (1) (2011) 87–102.
[2] E. Hawkes, B. An, N. Benbernou, H. Tanaka, S. Kim, E. Demaine, D. Rus, R. Wood, Programmable matter by folding, Proceedings of the National Academy of Sciences 107 (28) (2010) 12441–12445.
[3] N. Benbernou, E. Demaine, M. Demaine, A. Ovadya, A universal crease pattern for folding orthogonal shapes, arXiv:0909.5388, September 2009.
[4] E. Demaine, S. Devadoss, J. Mitchell, J. O'Rourke, Continuous foldability of polygonal paper, in: The 16th Canadian Conference on Computational Geometry, CCCG'04, August 2004, pp. 64–67.
[5] B. An, D. Rus, Programming and controlling self-folding robots, in: IEEE International Conference on Robotics and Automation, ICRA, May 2012, pp. 3299–3306.
[6] R. Nagpal, Programmable self-assembly using biologically-inspired multiagent control, in: International Conference on Autonomous Agents and Multiagent Systems, July 2002, pp. 418–425.
[7] R. Nagpal, Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics, Ph.D. Thesis, Massachusetts Institute of Technology, 2001.
[8] M. Yim, W-M. Shen, B. Salemi, D. Rus, H. Lipson, E. Klavins, G. Chirikjian, Modular self-reconfiguring robot systems: opportunities and challenges, IEEE/ASME Transactions on Mechatronics (2006).
[9] K. Gilpin, K. Koyanagi, D. Rus, Making self-disassembling objects with multiple components in the robot pebbles system, in: IEEE International Conference on Robotics and Automation, ICRA, May 2011, pp. 3614–3621.
[10] K. Gilpin, K. Kotay, D. Rus, I. Vasilescu, Miche: modular shape formation by self-disassembly, International Journal of Robotics Research (IJRR) 27 (3–4) (2008) 345–372.
[11] Z. Butler, K. Kotay, D. Rus, K. Tomita, Generic decentralized control for lattice-based self-reconfigurable robots, International Journal of Robotics Research 23 (9) (2004) 919–937.
[12] B. An, D. Rus, Making shapes from modules by magnification, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, October 2010, pp. 1140–1145.
[13] B. An, Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces, in: IEEE International Conference on Robotics and Automation, ICRA, May 2008, pp. 3149–3155.
[14] M. Yim, Y. Zhang, K. Roufas, D. Duff, C. Eldershaw, Connecting and disconnecting for self-reconfiguration with polybot, in: IEEE/ASME Transaction on Mechatronics, special issue on Information Technology in Mechatronics, 2003.
[15] A. Castano, A. Behar, P. Will, The conro modules for reconfigurable robots, IEEE/ASME Transactions on Mechatronics 7 (4) (2002) 403–409.
[16] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, S. Kokaji, Automatic locomotion design and experiments for a modular robotic system, IEEE/ASME Transactions on Mechatronics 10 (3) (2005) 314–325.
[17] D. Rus, M. Vona, Crystalline robots: self-reconfiguration with compressible unit modules, Autonomous Robots 10 (1) (2001) 107–124.
[18] A. Castano, P. Will, Mechanical design of a module for reconfigurable robots, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, October 2000, pp. 2203–2209.
[19] A. Pamecha, I. Ebert-Uphoff, G. Chirikjian, Useful metrics for modular robot motion planning, IEEE Transactions on Robotics and Automation 13 (4) (1997) 531–545.
[20] C. Ünsal, P. Khosla, Mechatronic design of a modular self-reconfiguring robotic system, in: IEEE International Conference on Robotics and Automation, ICRA, April 2000, pp. 1742–1747.
[21] P. White, V. Zykov, J. Bongard, H. Lipson, Three dimensional stochastic reconfiguration of modular robots, in: Proceedings of Robotics: Science and Systems, Cambridge, USA, June 2005.
[22] P. Bhat, J. Kuffner, S. Goldstein, S. Srinivasa, Hierarchical motion planning for self-reconfigurable modular robots, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, October 2006, pp. 886–891.
[23] K. Kotay, D. Rus, Locomotion versatility through self-reconfiguration, Robots and Autonomous Systems 26 (2–3) (1999) 217–232.
[24] K. Kotay, D. Rus, Algorithms for self-reconfiguring molecule motion planning, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, October 2000, pp. 2184–2193.
[25] R. Nagpal, Self-assembling global shape, using ideas from biology and origami, in: Thomas Hull (Ed.), Origami^3: Third International Meeting of Origami Science, Math and Education, A K Peters, 2002, pp. 219–231.
[26] M. Yim, Wei-Min Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, G.S. Chirikjian, Modular self-reconfigurable robot systems (grand challenges of robotics), Robotics & Automation Magazine, IEEE 14 (1) (2007) 43–52.
[27] J. Paik, E. Hawkes, R. Wood, A novel low-profile shape memory alloy torsional actuator, Smart Materials and Structures 19 (12) (2010) 125014.
[28] J. Paik, R. Wood, A bidirectional shape memory alloy folding actuator, Smart Materials and Structures 21 (6) (2012) 065013.
[29] E. Smela, O. Ingans, I. Lundstrm, Controlled folding of micrometer-size structures, Science 268 (5218) (1995) 1735–1738.
[30] E. Smela, A microfabricated movable electrochromic pixel based on polypyrrole, Advanced Materials 11 (16) (1999) 1343–1345.
[31] J. Paik, R. Kramer, R. Wood, Stretchable circuits and sensors for robotic origami, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2011, pp. 414–420.
[32] S. Brittain, O. Schueller, H. Wu, S. Whitesides, G. Whitesides, Microorigami: fabrication of small, three-dimensional, metallic structures, Journal of Physical Chemistry B 105 (2) (2001) 347–350.
[33] D. Balkcom, M. Mason, Robotic origami folding, International Journal of Robotics Research 27 (5) (2008) 613–627.
[34] D. Balkcom, M. Mason, Introducing robotic origami folding, in: IEEE International Conference on Robotics and Automation, ICRA, April 2004, pp. 3245–3250.
[35] D. Balkcom, Robotic origami folding, Ph.D. Thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2004.
[36] L. Lu, S. Akella, Folding cartons with fixtures: a motion planning approach, IEEE Transactions on Robotics and Automation 16 (4) (2000) 346–356.
[37] J. Dai, D. Caldwell, Origami-based robotic paper-and-board packaging for food industry, Trends in Food Science & Technology 21 (3) (2010) 153–157.



Byoungkwon An is interested in algorithms and computation for distributed robotics, distributed systems, programmable matter, origami and artificial intelligence systems. An earned his Master of Science in computer science from the Massachusetts Institute of Technology.

Daniela Rus is a Professor of Electrical Engineering and Computer Science and Director of the Computer Science and Artificial Intelligence Laboratory at MIT. Her research interests include distributed robotics and mobile computing and her application focus includes transportation, security, environmental modeling and monitoring, underwater exploration, and agriculture. Rus is the recipient of the NSF Career Award and an Alfred P. Sloan Foundation Fellow. She is a Class of 2002 MacArthur Fellow and a Fellow of AAAI and IEEE. Before receiving her appointment at MIT, Rus was a professor in the Computer Science Department at Dartmouth, where she founded and directed two laboratories in robotics and mobile computing. Rus earned her Ph.D. in computer science from Cornell University.