

Self-Organization of Environmentally-Adaptive Shapes on a Modular Robot

Chih-Han Yu*

chyu@fas.harvard.edu

François-Xavier Willems*

fxwillems@gmail.com

Donald Ingber[†]

donald.ingber@childrens.harvard.edu

Radhika Nagpal*

rad@eecs.harvard.edu

*School of Engineering & Applied Sciences
Harvard University, Cambridge, MA, USA

[†]Department of Pathology
Harvard Medical School, Boston, MA, USA

Abstract—Modular robots have the potential to achieve a wide range of applications by reconfiguring their shapes to perform different functions. This requires robust and scalable control algorithms that can form a wide range of user-specified shapes, including shapes that adapt to the environment. Here we present a *decentralized* algorithm for self-organizing of *environmentally-adaptive shapes*. We apply it to a chain-style modular robot, configured to form a flexible sheet structure. We show that the proposed algorithm is capable of achieving a wide class of environmentally-adaptive shapes, and the module control is simple, scalable, robust and provably correct. The algorithm is also *self-maintaining*: the shape automatically adapts if the environment changes. Finally, we present several applications which can be achieved within this framework via robot prototypes and simulations, such as a self-balancing table. In our experiments, we demonstrate the algorithm is highly responsive and robust in the face of real-world actuation and sensing noise.

I. INTRODUCTION

Modular robots are a class of robotic systems composed of many identical, connected, programmable modules that can coordinate to change the shape of the overall robot. This versatility allows a single robot to perform multiple tasks, typically the purview of special-purpose robots, such as achieving different types of locomotion (rolling, crawling, climbing) in complex terrains [1]. Modular robots also enable new application areas, from adaptive bridges and shelters, to 3D physical dynamic rendering [2].

The success of these applications requires the ability to program the behavior of the individual modules such that the system, as a whole, achieves a global shape goal. As the number of modules grows, centralized control becomes intractable and slow. Recently, several groups have demonstrated decentralized algorithms for reconfiguration that can achieve large classes of user-specified shapes [3], [4]. These algorithms rely on simple module rules and, in some cases, are provably correct in the face of asynchronous module movement. However, these algorithms only address goal shapes that are fully specified in advance — the environment plays no important role in the final shape. For many applications, the shape may need to directly sense the environment and conform its configuration to fit the environment. For example, a modular robot forming a table (or a flat “road” structure into a collapsed building) will have to adapt to the uneven terrain in order to keep its surface level; if the underlying terrain changes then it may also need to continually adapt to maintain the desired structure.

In this paper, we present a decentralized algorithm for the self-organization of *environmentally-adaptive shapes*. We focus on a chain-style modular robot configured to form a flexible surface [5], [6]; individual modules affect the shape through local actuation, and distributed tilt sensors provide environmental feedback. The desired shape is described in terms of regional constraints on orientation with respect to the environment. We present a simple decentralized multi-agent algorithm that has several salient features: (1) it relies on simple local rules for each module, (2) it is capable of achieving a wide class of environmentally-specified global shapes, (3) if the environment changes, the global configuration automatically adapts, (4) it is both scalable and robust, (5) the distributed control is provably correct: we can guarantee convergence for the class of shapes in our framework.

The ability to achieve complex shapes and adapt to the environment leads to many potential applications. We demonstrate several examples (Fig. 1) using a prototype hardware robot and a physics-based simulator, including self-balancing furniture, a terrain-adaptive bridge, and dynamic rendering for 3D media. Using our prototype hardware robot, we present experimental results on convergence speed, response to environmental change, accuracy in the presence of sensing and actuation noise, and robustness to actuator failure. These experiments demonstrate that the algorithm is highly responsive and robust in the face of real-world noise in actuation and sensing.

The remainder of the paper is organized as follows. We present related work in Section II. The modular robot model is described in Section III. We present the distributed shape formation algorithm and theoretical results in Section IV, followed by several example applications (Section V). Finally, we present our implementation of the robot and experimental results in Section VI.

II. RELATED WORK

Several research groups have demonstrated the design of modular robots. A modular robot is composed of many connected, identically-programmed modules, where each module can communicate locally with other modules that are physically connected to it. One common style is the “chain-based” modular robot [1], [7], [8], where individual modules can change their own shape through individual actuation (e.g. changing internal angles, Fig. 2(a)) and/or reconfigure

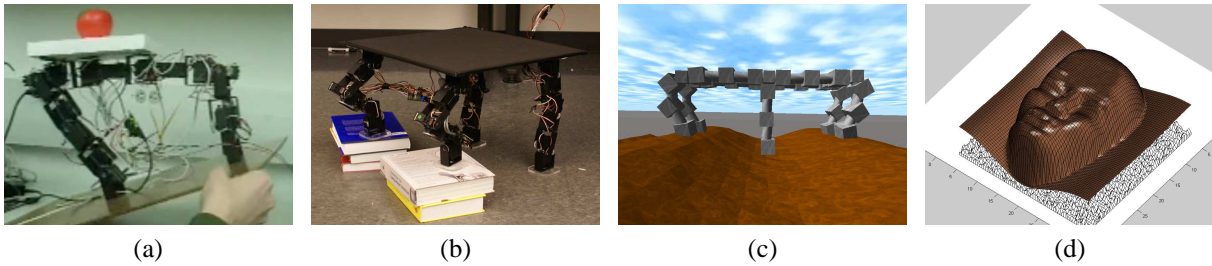


Fig. 1. Several different types of shapes that can be achieved within this framework: (a) Chain-typed structure (b) Self-balancing table prototype (c) Terrain-adaptive bridge simulation (d) Physical dynamic rendering simulation (face).

to change connectivity with neighboring modules. Another common style is the “lattice-based” modular robot, where overall shape change is achieved only through modules changing their local connectivity [3].

In chain-based robots, there are several centralized and a few decentralized algorithms for shape change [8], [9]. These algorithms focus on fixed, pre-specified global shape goals and do not involve feedback from the environment. However, there is also a considerable body of work on locomotion (dynamic shape change through module actuation) and in this context there are a few examples of using environmental feedback. For example, Yim et al. have demonstrated stair climbing where a robot uses pressure sensors to conform to the stair height [10]. In general the feedback control has been closely tied to the specific locomotion task; our work uses similar feedback control principles but in the context of shape formation.

In lattice-based robots, there has been some work on distributed algorithms for shapes that adapt to the environment. Bojinov et al. presented distributed algorithms for several examples in simulation, a hand that grasps an object and a table that supports a weight [11]. This work illustrates interesting potential applications, but the algorithms are difficult to translate to real hardware. Rus et al. have presented general distributed algorithms for locomotion and shape formation, over a terrain with unknown obstacles [3], [12]. In these algorithms, modules treat obstacles as non-moving modules, allowing the system to easily generalize to complex terrains. However, this work solves a slightly different problem than the one we are interested in, since the goal shape itself is not specified with respect to the environment. Also it is not clear how easy it will be to translate the abstract model of sensing to actual hardware. One limitation of lattice-based systems is that shape change can only be achieved through module movement which is slow in hardware implementation. In chain-based systems, modules can quickly actuate to produce large shape changes. Hence we focus on the latter style of modular robots in this work.

Distributed manipulation, e.g. [13], is another field that shares similarities with our framework. Research along this line tackles tasks like part transportation and reorientation through simultaneous motion of an array of actuators. Since it only considers static and known environments, the control parameters are generally pre-computed. Our approach further deals with uncertainties in the dynamic environment through

each module’s consistently sensing environment and self-organizing its own actuation. This process is also correlated with the consensus formation described in [14].

III. MODULAR ROBOT MODEL

In this section we describe our modular robot model, in particular the capabilities assumed of each module and how the modules are connected. Briefly, we assume that we have chain-based modules connected in a fixed configuration, consisting of a flexible surface with supporting legs (Fig. 2). A module can “compress” and “expand” its height by adjusting its internal angle. Shape change is achieved by groups of modules compressing and expanding. This concept of shape change through deformation is similar to the foldable sheet and deformation models in [5], [6], but implemented using simple chain-based modules.

A. Agent (Module) Model

Essentially, each module can be viewed as a single agent. Each agent has four components:

- **Computation:** We assume that all agents have identical programs, but may have different roles and thus behave differently. Each agent within the robot has a unique ID. We assume that the computation power of a single agent is limited, and our focus is on simple local rules that do not require complex calculations.
- **Communication:** Each agent can communicate with its immediate physically-connected neighbors.
- **Actuation:** Each agent is equipped with an actuator that allows it to change its angle, as shown in Fig. 2(a). Later in section VI we describe how we implement this in hardware.
- **Sensing:** Several agents are also equipped with accelerometers that allow the agent to determine its tilt angle with respect to the environment.

B. Flexible Surface Model

In this paper, we assume that the agents are connected to form a flexible surface with supporting legs (e.g. Fig. 2(a)). Thus the agents belong to one of two classes of groups: the *surface groups* and the *supporting groups*. The flexible surface is formed by all surface groups. A supporting group is a leg of the robot. We note that, by using rearrangement algorithms such as [15], one can make a chain-based robot reconfigure into this type of flexible surface model. In our

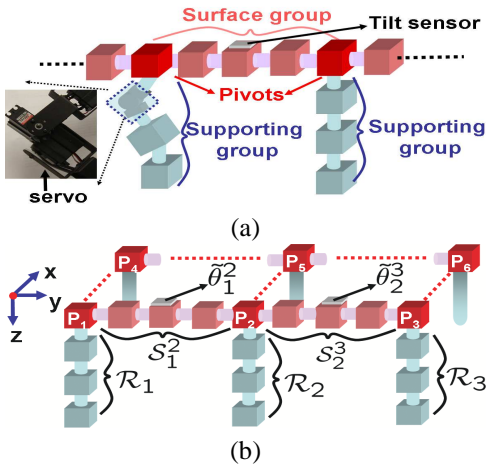


Fig. 2. (a) Flexible surface model. Each agents is equipped with a servo, and one of the agent in the surface group is also equipped with a tilt sensor. (b) An example group identity initialization, with pivots (P_i), surface groups (S_j^i), supporting groups (R_i), and tilt sensors (θ_j^i).

algorithm, we assume that this connectivity remains fixed throughout.

A surface group is a basic element of the flexible surface. When all surface groups are linked together, they form a layer of the flexible surface. Each supporting leg (group) consists of several agents that are vertically connected to the surface groups.

Shape change is achieved by the simultaneous actuation of the supporting groups. The surface groups play a mostly passive role in actuation¹. However they are equipped with the tilt sensors and thus distribute feedback about the orientation of the system.

C. Group Initialization

In our distributed control framework, each agent needs to be initialized with its group identification. These group identities remain fixed for a given robot configuration lifetime, regardless of the desired shapes being formed. We assume these identities are determined during an initialization phase

The final group identities are shown in Fig. 2(b). Surface group modules that connect directly to a supporting group play a special role. We define such a module as a “pivot”. Each pivot has a unique identity P_i . The supporting group underneath the pivot P_i is denoted as R_i . Each surface group underneath the pivot P_i is denoted as S_j^i . Note that these identities define the “locality” of information: the sensor mounted on S_j^i constantly propagates the tilt angle θ_j^i to P_i and P_j ; the pivot P_i collects sensor information only from surface groups it is connected to, and affects only its own supporting group R_i .

During the initialization phase, pivots create message gradients to help modules determine which group identity

¹In cases where the desired shape has a large slope, then surface modules will need to be stretched. In our bridge and dynamic physical rendering simulation, surface modules are connected to each other with elastic links so they can be stretched passively.

to assume. There has been considerable work on distributed algorithms for role assignment [6], therefore in the interest of space we only briefly describe the process. Pivots themselves can be identified via their physical connection, since they are the only modules which connect with other modules both horizontally (with other surface group modules) and vertically (with a supporting group module). Similarly bottom “feet” modules have only one connection. Both pivot and feet modules propagate messages that allow other modules to determine whether they are members of a surface or supporting group, and what group identity label to assume.

IV. DISTRIBUTED SHAPE FORMATION

In this section, we first describe how a desired shape is specified in our framework. Then we present our distributed feedback control algorithm which allows the robot to achieve the desired shape regardless of different initial environments. It also enables the robot to maintain the specified shape while facing changes in the environment.

A. Shape Specification

Given our flexible surface, we define a shape by specifying the desired tilt angles of all surface groups. This allows us to express a wide variety of shapes, as shown in Fig. 1 and Section V. However, the shape specification does not translate to a fixed module actuation. Rather the behavior of the modules depends on the terrain on which the flexible surface robot is placed.

For now, consider a simple case where our desired shape is that the surface must be always level, regardless of the underlying terrain. This case is illustrated in Fig. 3. We can describe the desired goal shape via specifying the tilt angles of each pivot’s neighboring surface groups. In the following example, all desired tilt angles $(\theta_j^i)^* = 0$.

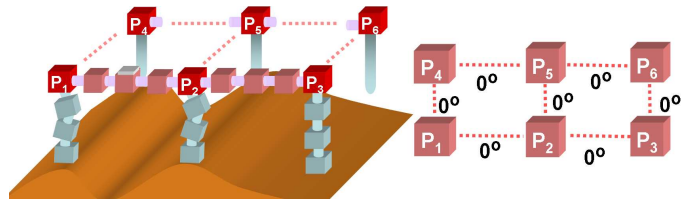


Fig. 3. An example of a shape specification.

B. Distributed Feedback Control

Our algorithmic approach is as follows: once a desired shape has been specified, all modules coordinate to deform the current shape until the desired goal is reached. As shown in Fig. 4 (a), we can think of the algorithm as continually iterating between two steps.

In Step 1, the sensor on each surface group S_j^i transmits its tilt angle θ_j^i to the neighboring pivots P_i and P_j . Each pivot P_i collects tilt sensor information from neighboring surface groups and computes the *aggregated feedback*.

In Step 2, each supporting group R_i receives an aggregated feedback from its pivot. Each module in the supporting group uses this information to control its actuation. When

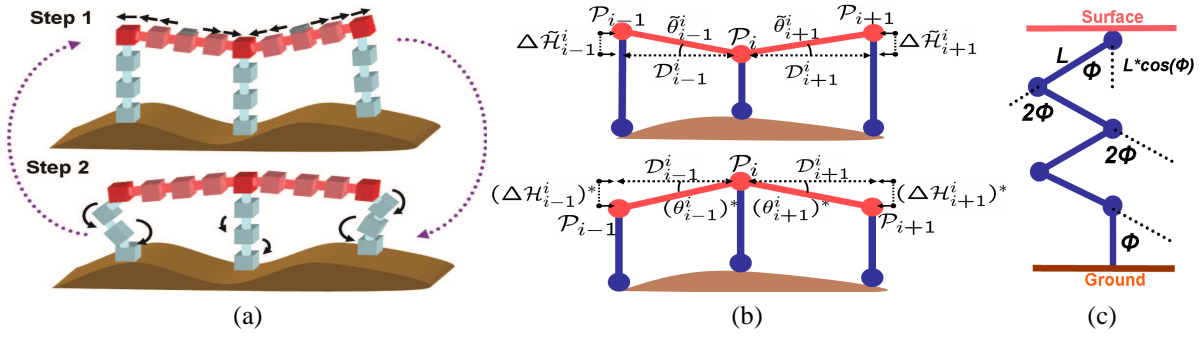


Fig. 4. (a) Step 1: Sensor on each surface group propagates current tilt angle to neighboring pivots. Step 2: Based on aggregated feedback computed by its pivot, each module in supporting group responds to the feedback by controlling its actuator. (b) Simulated link example of distributed adaptive control. Top: current state of the robot. Down: goal state of the robot. (c) Rotational angles of the supporting group agents. Agents in the middle rotate two times more than the top agent and the bottom agent.

the desired shape has been achieved (aggregated feedback is close to zero), modules no longer change actuation. If the environment changes, however, modules will respond automatically.

Each agent has only a local view, and each of them has its own goal (achieving desired local shapes). When the whole robot is placed on a new rough terrain, each agent will not be able to achieve its respective goal via merely one time actuation (since all agents are simultaneously changing their configurations). This iterative sensing-actuation distributed control scheme allows agents to collaboratively achieve their goals.

More specifically, our algorithm works as follows:

1) **Pivot Aggregated Feedback:** In this step, pivots aggregate neighboring sensor information to compute the aggregated feedback. Consider the example case in Fig. 4(b); the initial shape is shown on the top, and the desired shape is shown at the bottom.

Let us first consider the aggregated feedback for pivot \mathcal{P}_i whose neighboring pivots are \mathcal{P}_{i-1} and \mathcal{P}_{i+1} . Let $\tilde{\theta}_{i-1}^i$ denote the current tilt angle from pivot $\mathcal{P}_{i-1} \rightarrow \mathcal{P}_i$. Let \mathcal{D}_{i-1}^i be the horizontal distance between \mathcal{P}_{i-1} and \mathcal{P}_i (i.e. distance on x,y plane). Then the height difference between \mathcal{P}_{i-1} and \mathcal{P}_i is simply $\mathcal{D}_{i-1}^i \cdot \tan(\tilde{\theta}_{i-1}^i)$, which we denote as $\Delta\mathcal{H}_{i-1}^i$. Analogously, we can compute the height difference between \mathcal{P}_{i+1} and \mathcal{P}_i , denoted as $\Delta\mathcal{H}_{i+1}^i$.

In the desired shape configuration, we want to have tilt angles $(\theta_{i-1}^i)^*$ and $(\theta_{i+1}^i)^*$ with desired height differences $(\Delta\mathcal{H}_{i-1}^i)^* = \mathcal{D}_{i-1}^i \cdot \tan((\theta_{i-1}^i)^*)$ and $(\Delta\mathcal{H}_{i+1}^i)^* = \mathcal{D}_{i+1}^i \cdot \tan((\theta_{i+1}^i)^*)$ respectively.

In general, the aggregated feedback \mathcal{K}_i for all \mathcal{P}_i is computed as follows:

$$\mathcal{K}_i = \alpha \cdot \sum_{j \in \mathbb{N}_i} \xi_j^i \quad (1)$$

where control gain α is a constant and $0 \leq \alpha \leq 1$, and \mathbb{N}_i is the set of all neighboring pivots of \mathcal{P}_i . ξ_j^i is the feedback contribution from \mathcal{P}_j and is defined as:

$$\xi_j^i = (\Delta\mathcal{H}_j^i)^* - \Delta\mathcal{H}_j^i \quad (2)$$

In our example, $\mathcal{K}_i = \xi_{i-1}^i + \xi_{i+1}^i$, where $\xi_{i-1}^i = (\Delta\mathcal{H}_{i-1}^i)^* - \Delta\mathcal{H}_{i-1}^i$ and $\xi_{i+1}^i = (\Delta\mathcal{H}_{i+1}^i)^* - \Delta\mathcal{H}_{i+1}^i$.

The aggregated feedback for each pivot is computed in a purely distributed manner every time step. Each pivot transmits an aggregated feedback to the supporting group underneath it.

In some conditions, pivots might not know the distances to their neighbors or the distances have changed². Height differences between two pivots therefore cannot be computed. In such cases, we can simply modify feedback contribution ξ_j^i from \mathcal{P}_j in (2) as following:

$$\xi_j^i = (\theta_j^i)^* - \tilde{\theta}_j^i \quad (3)$$

2) **Supporting Group Actuation:** At a high level, each supporting group \mathcal{R}_i reacts to the aggregated feedback from pivot \mathcal{P}_i by compressing or decompressing the whole group by a small amount every time step. Aggregated feedback \mathcal{K}_i is an indicator of whether supporting group \mathcal{R}_i should increase or decrease its physical length and by how much. If $\mathcal{K}_i > 0$, it indicates \mathcal{R}_i needs to be compressed (therefore, it requires larger rotational angle) and vice versa.

The compression (or decompression) is achieved by individual modules acting in an “accordion-like” fashion (as shown in Fig. 4(c)). In order to achieve this behavior, we need modules within a support group to actuate (rotate) in opposing directions. We assume that in the initialization phase, each module agent initializes itself to either clockwise or counterclockwise alternatively, starting from the bottom “foot” module. In a supporting group \mathcal{R}_i of m agents, we denote the rotational direction of the k^{th} agent in it as $\lambda_i^k \in \{-1, 1\}$ (counterclockwise, clockwise). To maintain stability of the supporting group, the top ($k = 1$) and bottom ($k = m$) agents are programmed to rotate a half angle (as shown in Fig. 4(c)). The rotational angle of the k^{th} agent in \mathcal{R}_i (we denote it as ρ_i^k) is therefore updated as:

$$\rho_i^k \leftarrow \begin{cases} \rho_i^k + \lambda_i^k \cdot \mathcal{K}_i & \text{if } k = 1, m \\ \rho_i^k + 2 \cdot \lambda_i^k \cdot \mathcal{K}_i & \text{otherwise} \end{cases} \quad (4)$$

²For example, two pivots do not know how many surface modules there are between them. In our evaluation, the update rule based on (3) requires approximately twice as many iterations to converge to the desired shape as compared to the update rule based on (2).

Algorithm 1 PIVOT AGGREGATED FEEDBACK

1: **Input:** $\tilde{\theta}_j^i, \forall \mathcal{P}_j \in \mathbb{N}_i$
2: /* \mathbb{N}_i : set of all neighboring pivots of \mathcal{P}_i */
3: /* \mathcal{R}_i : supporting group underneath the pivot \mathcal{P}_i */
4: /* $(\theta_j^i)^*$: the desired tilt angle to neighboring pivot \mathcal{P}_j */
5: /* Pivot \mathcal{P}_i computes aggregated feedback: */
6: **for all** $\mathcal{P}_j \in \mathbb{N}_i$ **do**
7: compute $\Delta\tilde{\mathcal{H}}_j^i$
8: $\xi_j^i = (\Delta\mathcal{H}_j^i)^* - \Delta\tilde{\mathcal{H}}_j^i$ {or $(\theta_j^i)^* - \tilde{\theta}_j^i$ }
9: **if** $\xi_j^i \leq \epsilon$ **then** $\xi_j^i = 0$
10: $\mathcal{K}_i = \mathcal{K}_i + \alpha \cdot \xi_j^i$
11: **end for**
12: Send \mathcal{K}_i to every agent in \mathcal{R}_i

Algorithm 2 SUPPORTING GROUP ACTUATION

1: **Input:** \mathcal{K}_i
2: /* After receiving \mathcal{K}_i , k^{th} agent in \mathcal{R}_i computes its rotational angle */
3: $\rho_i^k \leftarrow \rho_i^k + \lambda_i^k \cdot \mathcal{K}_i$ { $2\lambda_i^k \cdot \mathcal{K}_i$ if $k = 1, m$ }
4: **if** $(\rho_i^k < -\rho_{max})$ **then** $\rho_i^k = -\rho_{max}$
5: **if** $(\rho_i^k > \rho_{max})$ **then** $\rho_i^k = \rho_{max}$
6: k^{th} agent updates its rotational angle to new ρ_i^k

Algorithm 1 and 2 shows the procedure of the distributed adaptive control³. The intuition behind the algorithm is to achieve local constraint satisfaction by collaborating with neighboring pivot modules proportionally to their “degree of dissatisfaction”. This quantity is expressed as the difference between its current configuration and its desired configuration. As each pivot simply computes the sum of this quantity induced from each of its neighbors, one can see that neighboring pivots with higher degree of dissatisfaction will contribute more to the sum. This simple scheme allows the robot to quickly achieve its desired shape.

C. Theoretical Analysis

Empirically, the update rule stated in (4) has a strong convergence property to the desired shapes (as shown in Section VI). In some cases, a pivot can calculate and control its supporting group’s height⁴. The control can be changed in such a way that the theoretical guarantees can be more strongly stated. As described before \mathcal{K}_i is an indication of the amount that \mathcal{R}_i needs to be compressed/decompressed, we use \mathcal{K}_i to update \mathcal{H}_i (the height of \mathcal{R}_i) directly:

$$\mathcal{H}_i \leftarrow \mathcal{H}_i - \mathcal{K}_i \quad (5)$$

Theorem 1 shows that the distributed feedback control algorithm based on (5) leads the robot to form the desired

³Additional constraints of the robot can be added to the control rules. For example, line 4 and 5 in Algorithm 2, we enforce each module’s rotation angle to be within ρ_{max} .

⁴We illustrate a mechanism for \mathcal{P}_i to control the height of \mathcal{R}_i : as shown in Fig. 4(c), the overall height of \mathcal{R}_i is simply: $\mathcal{L} + (m-1)\mathcal{L} \cdot \cos(\phi)$, where \mathcal{L} is the length of a module. The angle ϕ that leads to new \mathcal{H}_i is thus: $\phi = \cos^{-1}(\frac{\mathcal{H}_i - \mathcal{L}}{(m-1)\mathcal{L}})$. Instead of sending \mathcal{K}_i , pivot \mathcal{P}_i propagates ϕ to every agent in \mathcal{R}_i . The update rule for k^{th} agent in \mathcal{R}_i is thus modified to be: $\rho_i^k \leftarrow 2 \cdot \lambda_i^k \cdot \phi$ (if $k = 1, m$); $\rho_i^k \leftarrow \lambda_i^k \cdot \phi$ (otherwise).

shape with either feedback contribution computation scheme (Eq. (2) or (3)).

THEOREM 1 (CONVERGENCE) $\forall \mathcal{P}_i$ and $\forall \mathcal{P}_j \in \mathbb{N}_i$. If current tilt angle $\tilde{\theta}_j^i$ is accessible to \mathcal{P}_i and \mathcal{H}_i is updated based on \mathcal{K}_i , the algorithm will drive $\tilde{\theta}_j^i$ to the desired $(\theta_j^i)^*$, regardless of initial conditions and whether or not \mathcal{D}_j^i (distance between \mathcal{P}_i and \mathcal{P}_j) is known.

PROOF see Appendix.

V. APPLICATIONS

Our algorithm for shape formation has several important features: (1) The algorithm involves simple, local behavior by each agent, which scales as we add more supporting groups to the flexible sheet; (2) the algorithm is guaranteed to converge to the target shape; (3) if the terrain changes, the robot automatically adjusts to maintain the desired shape.

These features lead to many potential applications. Here we describe some examples that we have constructed via hardware prototypes or simulations:

- **Self-balancing Chain/Table:** Fig. 1(a) and (b) show hardware prototypes of the self-balancing chain and table respectively. In our demonstration, the top portions of the chain and table remain level regardless of terrain conditions. This could be useful in many circumstances, e.g. stabilizing instruments on a boat.
- **Terrain-Adaptive Bridge:** In our framework, one can achieve a modular robotic bridge that can adapt to different terrains. We constructed a terrain-adaptive bridge simulator with Open Dynamics Engine [16] (Fig. 1(c)). When it is placed on an unknown rough terrain, the robot can automatically form a flat surface or a smooth incline. Even if the terrain changes over time, the modular robot adapts to maintain a level surface. Robot locomotion over rough terrains has been a challenging problem. A modular robotic bridge can automatically form a smooth roadway over the rough terrain for the other robots.
- **Dynamic Physical Rendering:** Dynamic Physical Rendering is an application where a modular robot forms arbitrary shapes as a novel form of 3D media and visualization. Our proposed flexible surface can act as a “relief” display, since the distributed algorithm can easily achieve complex shapes (as shown in Figure 1(d)). Applications in this domain require efficient transformation from one shape to another. The distributed property of our algorithm makes this high dimensional control problem scalable and allows efficient shape transformation (as shown in Section VI-D).

Note that our approach can be used in combination with traditional rearrangement reconfiguration, e.g. a modular robot can locomote quickly on smooth terrain using a track-like configuration and then configure to form a bridge over rough terrain. We also expect that this distributed control approach can be extended to dynamic shape descriptions

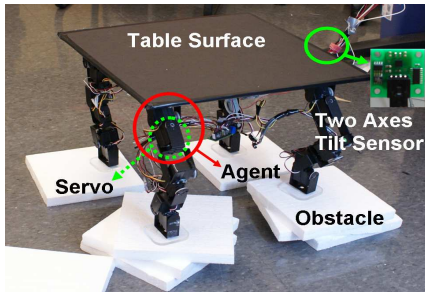


Fig. 5. A self-balancing table robot. Each agent controls a rotary servo. A two axes tilt sensor (accelerometer) is mounted on the table surface.

and other types of sensing (e.g. pressure), opening up many application possibilities.

VI. EXPERIMENTAL RESULTS

In this section, we describe experimental results of our distributed shape formation algorithm. We begin by describing the hardware prototype in Section VI-A. In Section VI-B, we present results from examining how fast the robot can adapt to environment changes and how well it performs in different types of terrains. Section VI-C provides an analysis of the robustness of the algorithm towards servo failures. In the last set of experiments, we examine the scalability properties of the algorithm in large scale shape formation simulations.

Our experimental results show that our distributed algorithm allows the robot to respond to different environmental changes effectively. In addition, it is mostly robust to servo failures and scalable to a large number of agents.

A. Hardware Prototype

We designed and implemented a self-balancing table robot to test how well our approach works in real world scenarios. The robot is composed of four supporting groups (legs), and each composed of three agents. Since the table surface is designed to be flat, the surface group modules are replaced by a single rigid surface (43cm X 43cm square) that forms the table and has the tilt sensor mounted in the middle. In addition to its original role, the top agent of each supporting group is also programmed to be a pivot.

Fig. 5 shows each component of the robot. Each agent controls a Hitec standard servo which can perform a rotation of 90° in either a clockwise or counterclockwise direction. We mounted a two-axis (x and y) tilt sensor (Analogue Devices ADXL311 accelerometer) on the table surface. Each of the pivots can receive from this sensor, instead of having their own tilt sensors.

For simplicity of implementation, the distributed shape formation algorithm is run on a laptop computer (2GHZ CPU) that simulates purely distributed control. Although the distributed control is simulated, our hardware implements the sensing and distributed actuation so that we can directly test the algorithm in the face of real-world noise. After each agent computes the new angle of its servo, the control signal is sent to the hardware robot via serial port. It takes approximately 50 milliseconds for all agents to finish one iteration. This

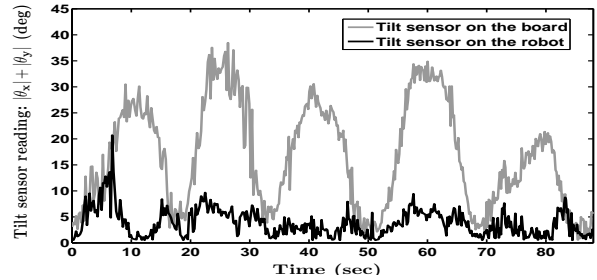


Fig. 6. The robot's response time to repeated environment changes. The robot was able to maintain its surface's tilt angle within $5^\circ - 8^\circ$ even when the board is tilted $30^\circ - 40^\circ$ over a few seconds.

hardware prototype robot is used in the experiments of the next two sections.

B. Environment Response Experiments

In the first experiment, we examine how quickly and accurately the robot responds to consistent, rapid environmental changes. In this experiment, we fix the robot's four supporting groups to a rigid board. We repeatedly changed the orientation of the board to examine the robot's response (as shown in Fig. 7(a) - (d)). One additional tilt sensor is mounted on the board to record environmental changes. This sensor does not supply input to the robot. Empirically, the sensors we use are somewhat noisy, especially under high speed motion e.g. first five seconds of Fig. 6.

Agents are programmed to maintain a surface level surface; i.e. tilt angles in x axis and y axis, θ_x and θ_y , equal to zero at all times. Therefore, $|\theta_x| + |\theta_y|$ is an error measure of how far the table surface is from a level state.

Fig. 6 shows the results of the experiment. We can see that even when the tilt angle of the floor is changed by $30^\circ - 40^\circ$ over a few seconds, the table is able to quickly respond and keep the surface level. The table never tilts more than $5^\circ - 8^\circ$ after the initial correction.

In the second experiment, we examine how the robot responds to different rough terrains. As shown in Fig. 7(e)-(h), the robot's four supporting groups were placed on four obstacles of different heights. Each foot placement position was placed with several bricks (a brick's thickness is 2.5 cm). Through different combinations of bricks, we can generate different rough terrains. The robot's upper left supporting group position is denoted as *UL*, lower left is *LL*, upper right is *UR*, and lower right is *LR*. The number following each denotes how many bricks were placed at that position.

Fig. 8 shows the robot's response time to achieving levelness under different terrains. In the first five experiments, two legs on one side are lifted. In the last two experiments, the robot's four legs are lifted simulating fully irregular terrain scenarios. As shown in the figure, the robot is capable of achieving levelness ($|\theta_x| + |\theta_y| \leq 3^\circ$) within 2 seconds (~ 40 iterations) in most of the cases. Experiment 6 is the only case which takes the robot > 2 seconds to achieve levelness, since its setup requires all pivot to collaborate with its neighbors rigorously.

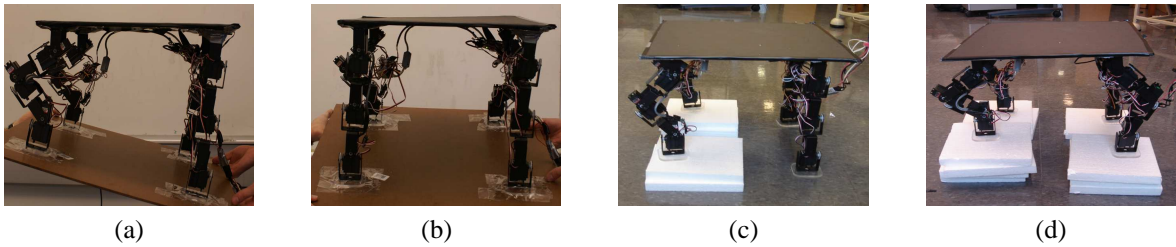


Fig. 7. (a) - (b) The robot is fixed to a board. The robot keeps its surface level when we repeatedly change the orientation of the board. (c) - (d) The robot is placed on obstacles of different height. It adapts to different obstacles and forms a flat surface.

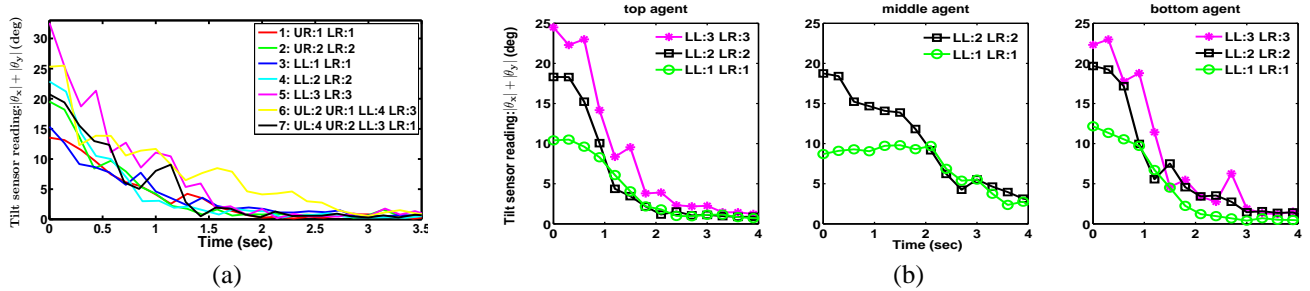


Fig. 8. (a) The robot’s response time to different initial conditions. In seven out of eight experiments, it achieves $|\theta_x| + |\theta_y| \leq 3^\circ$, within 2 seconds (30 iterations). (b) Robustness test results when one of the agents does not respond. The left, middle, and right figures are scenarios when top, middle, and bottom agents do not respond respectively. It is most critical when the middle agent fails.

We note that as the table surface is a rigid object and cannot be stretched, the horizontal between two pivots might change in the process. Nevertheless, the algorithm still behaves correctly even if we treat the horizontal distance as a fixed constant over the process.

C. Robustness Experiments

We also perform robustness experiments by observing the robot’s reaction when one of the agents fails. We tested it under different task difficulties and in different positions in the group. We tested two situations which an agent fails to respond: (1) the agent’s servo is disabled and becomes a passive link, so it freely takes on any angle with no resistance to movement; and (2) the agent’s servo remains stuck at the zero degree position at all times. We discovered that the first case does not affect the effectiveness of the algorithm, while the second case affects a few scenarios. It implicitly means the algorithm is robust to hardware failure of the first case. We only discuss the second case here.

Fig. 8(b) shows the robot’s responding time to achieve levelness while different agents do not participate the task. We lifted one side of the robot to 1 to 3 bricks high respectively. At each height, one of the three agents in the supporting groups that needs to be compressed is disabled (top, middle, or bottom). We repeated this process four times and Fig. 8(b) shows the average of robot’s tilt angle across time. We can see from Fig. 8(b) that the middle agent’s failure is more critical than top and bottom agents. When the middle agent fails, the robot generally falls over in the third experiment (3 bricks). When the obstacle is one or two bricks high, it achieves $< 4^\circ$ of tilt angle in ~ 4 seconds.

We also observed that when the middle agent fails, it leads to a more unstable state of the robot. This is primarily

because it is responsible for two times more rotation than either top or bottom module (as shown in Fig. 4(c)). One possible solution is to have more modules in each leg which allows a greater flexibility to compensate for individual failure, as well as increase the range over which the leg can compress and uncompress.

D. Scalability Experiments

The distributed algorithm can provably form arbitrary shapes, and the pivot actions remain local even when the number of surface groups increases. Here we evaluate the scalability of our system by observing how convergence time is affected by a large number of surface groups and different shape complexities. We implement a simulation of a 64×64 flexible sheet in MATLAB, which includes 4096 pivots/supporting legs (16384 agents) for tasks of forming a pre-defined 3D shape. We assume surface groups are formed by elastic materials. In simulation, we added Gaussian noise to both servo actuation and sensor readings.

The robot is programmed to render six 3D models: a statue, teapot, knot, bunny, donut, and face. 3D depth information is used to transform these models into tilt angles for shape specification. We started the simulation by placing the robot on a randomly generated terrain. We define the initial state as 100% error to the desired shape and 0% error when the desired shape is perfectly achieved. Table I shows the mean and standard deviation for the robot to reach 10% of error. In our previous experiments, the self-balancing table (12 agents) achieves 10% of error around 40 iterations. We can see from Table I that our algorithm scales well with the number of agents: when the number of agents increases from 12 to 16384 and the shapes become much more complex, the number of iterations required increases only 10 \sim 15 times.

	statue	teapot	knot	bunny	donut	face
mean	696.1	417.3	678.2	443.3	675.2	528.8
std.	4.4	17.4	33.6	68.5	25.6	136.7

TABLE I

MEAN AND STANDARD DEVIATION OF NUMBER OF ITERATIONS FOR 16384 AGENTS TO REACH 10% OF ERROR IN DIFFERENT 3D SHAPES.

Videos of the robot are also online at:
<http://www.eecs.harvard.edu/~chyu/ModularRobot>

VII. CONCLUSIONS

We have presented a decentralized control framework that allows a chain-style modular robot to achieve various environmentally-adaptive shapes. The control algorithm is shown to provably converge: it leads the robot to form the desired shapes regardless of its initial conditions and environmental changes. Through our experiments, we demonstrate that the proposed algorithm is effective in real world applications. In contrast to centralized algorithms, it is more scalable and robust to individual module failures. Future work will focus on extending our framework to other classes of shapes and applications. Furthermore, we are interested in investigating how the robot can autonomously form shapes to achieve a specific task, e.g., generating a sequence of shapes to transport people from a collapsed building. Ultimately, we wish to elevate the user specification from describing desired shapes to higher levels of task abstractions.

ACKNOWLEDGEMENT

We thank to J. Werfel and N. Khaneja for many helpful discussions; R. Wood, I. Rose, C. Lim, H. Pon-Barry, and W. Lee for proofreading the paper; C. Su and J. Ma for helping with experimental setup. This work is partially funded by the National Science Foundation under Grant No. 0523676.

REFERENCES

- [1] W.-M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein, and J. Venkatesh, "Multimode locomotion for reconfigurable robots," *Autonomous Robots*, vol. 20, no. 2, pp. 165–177, 2006.
- [2] S. C. Goldstein, J. Campbell, and T. C. Mowry, "Programmable matter," *IEEE Trans. Comput.*, vol. 38, no. 6, pp. 99–101, 2005.
- [3] D. Rus, Z. Butler, K. Kotay, and M. Vona, "Self-reconfiguring robots," *Communications of the ACM*, vol. 45, no. 3, pp. 39–45, 2002.
- [4] K. Støy and R. Nagpal, "Self-repair and scale independent self-reconfiguration (for a modular robot)," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.
- [5] K. Støy, "The deformatron robot: a biologically inspired homogeneous modular robot," in *Proc. IEEE International Conference on Robotics and Automations (ICRA)*, 2006.
- [6] R. Nagpal, "Programmable self-assembly using biologically-inspired multiagent control," in *Proc. First International Joint Conference on Autonomous Agents and Multiagent System (AAMAS)*, 2002.
- [7] M. Yim, Y. Zhang, K. Roufas, D. Duff, and C. Eldershaw, "Connecting and disconnecting for chain self-reconfiguration with polybot," *IEEE/ASME Trans. Mechatron.*, vol. 7, no. 4, pp. 442–451, 2002.
- [8] S. Murata, E. Yoshida, A. Kamimura, K. T. H. Kurokawa, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *IEEE/ASME Trans. Mechatron.*, vol. 7, no. 4, pp. 431–441, 2002.
- [9] E. H. Ostergaard, K. Tomita, and H. Kurokawa, "Distributed metamorphosis of regular m-tran structures," in *Proc. 7th International Symposium on Distributed Autonomous Robotic Systems*, 2004.

- [10] M. Yim, C. Eldershaw, Y. Zhang, and D. G. Duff, "Limless conforming gaits with modular robots," in *Proc. 9th International Symposium on Experimental Robotics (ISER04)*, 2004.
- [11] H. Bojinov, A. Casal, and T. Hogg, "Emergent structures in modular self-reconfigurable robots," in *Proc. IEEE International Conference on Robotics and Automations (ICRA)*, 2000.
- [12] K. Kotay and D. Rus, "Generic distributed assembly and repair algorithms for self-reconfiguring robots," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.
- [13] K. F. Bohringer and H. Choset, Eds., *Distributed Manipulation*. Kluwer Academic Publishing, 2000.
- [14] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," in *Proc. of IEEE*, 2007.
- [15] K. Payne, B. Salemi, P. Will, and W.-M. Shen, "Sensor-based distributed control for chain-typed self-reconfiguration," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2004.
- [16] "Open dynamics engine." [Online]. Available: <http://www.ode.org>

APPENDIX: PROOF OF THEOREM 1

Proof: Here we prove the case when each pivot has two neighboring pivots (as shown in Fig. 4). It can be easily generalized to the case of Fig 3. As the distances between two neighboring pivots are assumed to be unknown, the aggregated feedback contribution can only be computed from the raw sensor readings: $\xi_j^i = (\theta_j^i)^* - \tilde{\theta}_j^i$. We denote pivot \mathcal{P}_i 's set of two neighbors as $\Phi_i = \{\mathcal{P}_{i-1}; \mathcal{P}_{i+1}\}$. The desired tilt angles of the two neighboring surface groups are $(\theta_{i-1}^i)^*$ and $(\theta_{i+1}^i)^*$ respectively, and their current tilt angles are denoted as $\tilde{\theta}_{i-1}^i$ and $\tilde{\theta}_{i+1}^i$. We note that any $\theta_i^j = -\theta_j^i$ in our coordinate system. We can reverse $(\theta_{i-1}^i)^* = -(\theta_{i+1}^i)^*$ and $\tilde{\theta}_{i+1}^i = -\tilde{\theta}_{i-1}^i$ accordingly. The aggregated feedback \mathcal{K}_i is therefore: $\mathcal{K}_i = \alpha \cdot [\xi_{i-1}^i + \xi_{i+1}^i] = \alpha \cdot [((\theta_{i-1}^i)^* - \tilde{\theta}_{i-1}^i) + ((\theta_{i+1}^i)^* - \tilde{\theta}_{i+1}^i)] = \alpha \cdot [((\theta_{i-1}^i)^* - \tilde{\theta}_{i-1}^i) - ((\theta_{i+1}^i)^* - \tilde{\theta}_{i+1}^i)]$.

We define $\Delta\theta_{i-1}^i = \tilde{\theta}_{i-1}^i - (\theta_{i-1}^i)^*$ and $\Delta\theta_{i+1}^i = \tilde{\theta}_{i+1}^i - (\theta_{i+1}^i)^*$ to represent the difference between current tilt angle and the desired tilt angle, so $\mathcal{K}_i = \alpha \cdot [-\Delta\theta_{i-1}^i + \Delta\theta_{i+1}^i]$. Following (5), the height of p_i is updated as:

$$\mathcal{H}_i^{n+1} = \mathcal{H}_i^n - \mathcal{K}_i = \mathcal{H}_i^n + \alpha \cdot \{\Delta\theta_{i-1}^i - \Delta\theta_{i+1}^i\} \quad (6)$$

The Lyapunov function \mathcal{V}^{n+1} is defined as the distance measure between the current shape and the desired shape in the $(n+1)^{th}$ iteration. Let Ω be the set of all pivots, \mathcal{V}^{n+1} is defined as:

$$\mathcal{V}^{n+1} = \sum_{p_i \in \Omega} |(\Delta\tilde{\mathcal{H}}_{i-1}^i)^{n+1} - (\Delta\mathcal{H}_{i-1}^i)^*| \quad (7)$$

where $(\Delta\tilde{\mathcal{H}}_{i-1}^i)^{n+1} = \mathcal{H}_{i-1}^{n+1} - \mathcal{H}_i^{n+1}$ is the height difference between p_{i-1} and p_i in the $(n+1)^{th}$ iteration, and $(\Delta\mathcal{H}_{i-1}^i)^*$ is the desired height difference. \mathcal{V}^{n+1} can be further expanded:

$$\begin{aligned} \mathcal{V}^{n+1} &= \sum_{p_i \in \Omega} |\mathcal{H}_{i-1}^n + \alpha \cdot \{\Delta\theta_{i-2}^{i-1} - \Delta\theta_{i-1}^i\} \\ &\quad - \mathcal{H}_i^n - \alpha \cdot \{\Delta\theta_{i-1}^i - \Delta\theta_{i+1}^i\} - (\Delta\mathcal{H}_{i-1}^i)^*| \quad (8) \end{aligned}$$

$$\begin{aligned} &\leq \sum_{p_i \in \Omega} |(\Delta\tilde{\mathcal{H}}_{i-1}^i)^n - (\Delta\mathcal{H}_{i-1}^i)^* - 2\alpha \cdot \Delta\theta_{i-1}^i| \\ &\quad + \alpha|\Delta\theta_{i-2}^{i-1}| + \alpha|\Delta\theta_{i+1}^i| \quad (9) \end{aligned}$$

$$\begin{aligned} &= \sum_{p_i \in \Omega} |(\Delta\tilde{\mathcal{H}}_{i-1}^i)^n - (\Delta\mathcal{H}_{i-1}^i)^* - 2\alpha \cdot \Delta\theta_{i-1}^i| \\ &\quad + \sum_{p_i \in \Omega} 2\alpha \cdot |\Delta\theta_{i-1}^i| \quad (10) \end{aligned}$$

$$= \sum_{p_i \in \Omega} |(\Delta\tilde{\mathcal{H}}_{i-1}^i)^n - (\Delta\mathcal{H}_{i-1}^i)^*| = \mathcal{V}^n \quad (11)$$

Eq. (8) to (9) follows from the triangle inequality. (9) to (10) holds due to conservation property of our distributed control: the number of $\alpha|\Delta\theta_{i-1}^i|$ being subtracted equals the number of \mathcal{P}_i 's neighbors, and each \mathcal{P}_i 's neighbors adds a quantity of $\alpha|\Delta\theta_{i-1}^i|$. (10) to (11) follows from the fact that $\|x - y\| + \|y\| = \|x\|$ if $x \cdot y > 0$ and $(x - y) \cdot x > 0$. The first condition holds since $(\Delta\tilde{\mathcal{H}}_{i-1}^i)^n - (\Delta\mathcal{H}_{i-1}^i)^*$ and $\theta_{i-1}^i - (\theta_{i-1}^i)^*$ are the same sign, and the second holds since α is a small positive constant.

We can see that the $\mathcal{V}^1, \mathcal{V}^2, \mathcal{V}^3, \dots$ is a non-increasing sequence (decreasing sequence except for the fixed point). The only fixed point for \mathcal{V}^k sequence is when $(\Delta\tilde{\mathcal{H}}_{i-1}^i)^k = (\Delta\mathcal{H}_{i-1}^i)^*$. Based on LaSalle's Invariance Principle, the robot will converge to the desired shape. ■