

Distributed Formation Control for a Modular Mechanical System

Eiichi YOSHIDA Satoshi MURATA Kohji TOMITA
Haruhisa KUROKAWA Shigeru KOKAJI

Mechanical Engineering Laboratory, AIST, MITI
1-2 Namiki, Tsukuba-shi, Ibaraki 305 Japan e-mail: eiichi@mel.go.jp

Abstract

A distributed formation control method is proposed for a modular mechanical system. We have developed a totally decentralized system composed of many homogeneous mechanical units which are designed to change their connective configuration using only local information. The control method proposed in this paper enables the system to re-organize themselves so that various configurations can be formed in a robust way. Computer simulations and experiments are carried out to show its effectiveness.

1 Introduction

Mechanical systems inspired by biological organisms are studied more and more intensively in the engineering field. One of challenging subjects is the development of a distributed mechanical or robotic system consisting of many modules.

This idea comes from the fact that a collection of cells composing living beings can show a variety of complex functions like self-assembly and self-repair.

Can we construct a system composed of many simple mechanical units that can realize various functions by reconfiguration? Is it possible to let the mechanical system repair itself?

The following are the key requirements to build such a system.

Homogeneity. Each mechanical unit had better be homogeneous in terms of both hardware and software because:

any unit can play any part of the overall system and can replace any defective unit.

Local information. The behavior of unit is determined only by local information:

this enables the entire system to act in a distributed manner without central coordinator.

Dynamic reconfigurability. Units should be able to change their connective configuration by themselves:

this is necessary for self-assembling and self-repairing capacity.

Over the past few years a considerable number of studies have been made on reconfigurable mechanical or robotic systems. Polypod [1] and Tetrobot [2] are reconfigurable mobile robots. Other work [3] [4] has developed robots capable of reconfiguring dynamically. These systems, however, do not necessarily satisfy all of the requirements above.

From this viewpoint, we have been developing a distributed mechanical system which is composed of many units called “fracta” [5]. We consider that different functions are realized if the units can form various shapes by self-assembly like living things. The system can also repair itself if some part is removed, by replacing it with spare units.

This system can be a self-maintainable versatile machine which works as a mobile robot or a manipulator, as well as a static structure in hazardous environments. Since the units are homogeneous, distributed implementation of formation is an essential issue. Although there are studies on formation by cellular robots [3] [4] and by multi-agents [6]-[8], we need to consider further constraints of strict homogeneity and physical connection to realize distributed algorithms which allow those units to achieve self-assembly and self-repair.

Thus, we have been developing distributed formation algorithms together with hardware. Twenty prototype units have been made for experiments so far. That is a sufficient number to examine the features of our system. We have investigated a distributed formation control algorithm [5], but it lacks robustness and self-repairing ability. Our another algorithm [9] is

rather dedicated to hundreds of units, which does not suit the current stage of hardware development.

This paper, therefore, focuses on improving the distributed formation algorithm and demonstrating its effectiveness by applying it to the hardware setup. After introducing the hardware implementation in section 2, we will give a detailed account of the distributed formation control algorithm in sections 3 and 4. The effectiveness of the control algorithm is evaluated in section 5 by computer simulations. Section 6 presents the experimental results.

2 Hardware implementation

The overview of the developed hardware of distributed mechanical system is described here. The hardware system is mainly composed of motion generation part using electro- and permanent magnets and communication devices as shown in Fig. 1. The hardware specification is shown in Table 1.

2.1 Basic functions

We adopted three layered structure to meet both homogeneity and dynamic reconfigurability. The three permanent magnets are arranged with 120 degrees in each of the top and the bottom layer. The middle layer contains three solenoids and is rotated by 60 degrees from the outer layers.

If a solenoid has the same polarity as the permanent magnet

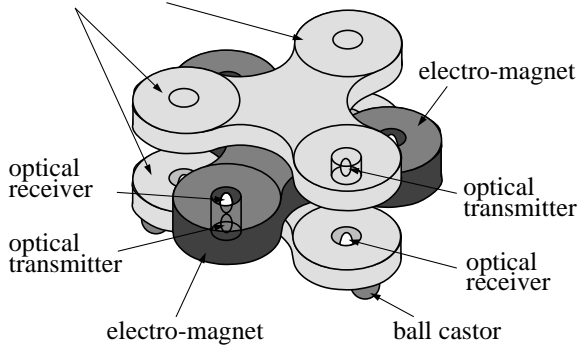


Fig. 1: Distributed mechanical unit "fractum"

Table 1: Specifications of hardware unit

CPU	Z80 (10MHz)
Diameter	8cm
Weight	0.4kg
Electric current consumption	0.9A
Communication	independent six channel I/O serial 9600bps

then it is attracted into the gap between outer layers of another unit, which makes connection between two units. A unit can connect with maximum six other units. Repulsion takes place with the inverse polarity of the solenoids. Note that a unit cannot move by itself but generate only the relative motion.

Figure 2 shows how two units change their connection based on the operation of the solenoids. In the initial state, two units A, B are connected by single bond and let unit B rotate by 60 degrees clockwise. It is completed by the following operations.

- Unit B attracts the permanent magnet of unit A by controlling the solenoid b_3 .
- Unit A repels the permanent magnet of B by controlling a_1 to cut the former connecting bond.

2.2 Communication

When two units are connected, they can exchange information via optical communication channel. As shown in Fig. 1, a unit is equipped with a pair of optical transmitter and receiver at each of six bonds, which allows it to communicate with up to six neighbors independently.

3 Distributed formation control

The purpose of the distributed formation control is to achieve the goal formation from any initial configuration (where all the units are assumed to be con-

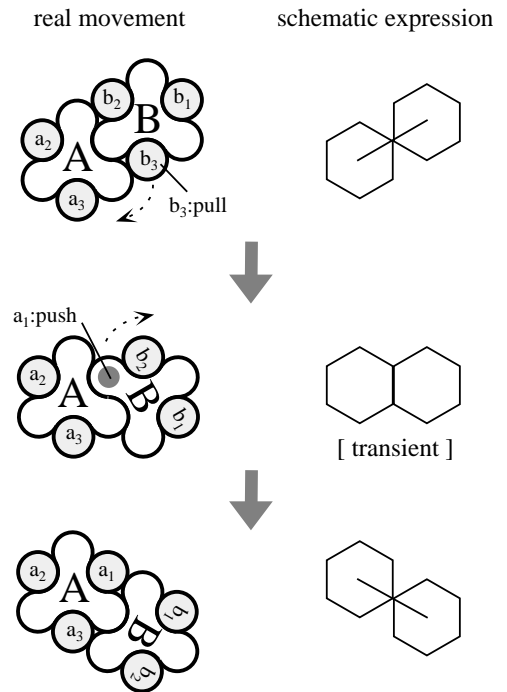


Fig. 2: Basic motion of units

nected). As mentioned in section 1, each unit should have the same software and decide its movement only according to available local information. We have been developing an algorithm for this function [5].

The algorithm consists of the following procedures. Each unit

- (i) calculates “difference” between the current and goal configuration
- (ii) obtains the average difference around the unit using a diffusion process (inter-unit communication is used)
- (iii) moves randomly if the unit has a relatively large difference

We call this calculation cycle a “step”.

After introducing the notation of “connection type” of units [5], we give a brief overview of the proposed algorithm in our previous research [5].

3.1 Connection types

In the process of formation control, each unit decides its motion according to its current local state. Thus we need to describe how a unit is connected to others. For this purpose, we classify connection types.

A unit can be formally described as a hexagon since it has six possible bonds. The possible 12 connection types and their mutual relationship are shown in Fig. 3. The link between two types denotes that one type may be changed to the other by a single basic motion like in Fig. 2.

The notation of “distance” between two types can be defined as the numbers of links in the shortest path. The larger this distance becomes, the longer time it takes to transit from one type to another.

It is important to state here that some of these connection types are not “movable”. In some cases, units may be physically impossible to move, and in other cases, the movement of units may divide the system into two unconnected parts. We therefore define a movable unit as one which:

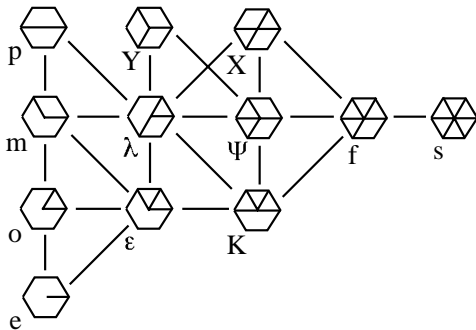


Fig. 3: Connection types and their relationship

- can rotate without carrying other units
- keeps the whole system connected after the movement.

According to criteria above, we define units with connection types “e”, “o”, and “ε” as movable.

3.2 Overview of control algorithm

The formation control algorithm is basically grounded on “difference” and diffusion process.

3.2.1 Goal description

First, we describe the goal formation using the types defined in 3.1. As an example, let us introduce a goal shape of a triangle composed of 10 units shown in Fig. 4. The unit with the goal type “K” is connected to 4 units whose types are {o, K, K, s}. We refer to this as “statement” $K\{o, K, K, s\}$. The goal shape is written as the collection of these statements:

$$\begin{aligned} & o\{K, K\} \\ & K\{o, K, K, s\} \\ & s\{K, K, K, K, K, K\} \end{aligned} \quad (1)$$

Connection types of each statement in (1) are sorted in order of the number of connection bonds.

We note here that this description has limitation. Although the description like (1) can be determined for a given goal shape, it is no unique; there can be multiple goal forms corresponding to the same description. In such a case, some additional criteria must be added, like priorities introduced later in section 4. But the method above has adequate capability to describe goal forms with up to twenty units as long as the goal shape is not too much complicated.

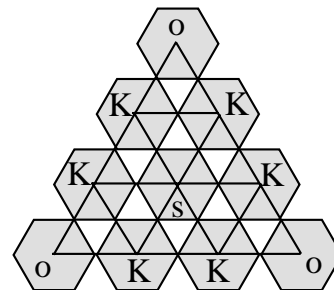


Fig. 4: Description of triangle by 10 units

3.2.2 Calculation of difference

Each unit calculates the “difference” between its current state and the goal in (1) as:

$$\text{diff}(i) = \min_{j=1}^M [\text{d}(\text{type}^g(j), \text{type}(i)) + \sum_{k=1}^6 \text{d}(\text{n\text{type}}^g(j, k), \text{n\text{type}}(i, k))] \quad (2)$$

where

- i : index of unit
- M : number of goal types
- $\text{d}(a, b)$: distance between types “ a ” and “ b ”
- $\text{type}(i)$: current type of unit i
- $\text{type}^g(j)$: j -th goal type
- $\text{n\text{type}}(i, k)$: current k -th neighbor type of unit i
- $\text{n\text{type}}^g(j, k)$: k -th neighbor type of j -th goal statement

In (2), for each goal statement, a unit calculates the sum of distances between types in each term of its current connection state and the goal statement. The difference is the minimum value of these sums. As a result, each unit evaluates the difference from the closest goal statement.

3.2.3 Diffusion process

It is preferable for units with relatively larger difference to move so that the group of units can form the desired goal configuration. To allow units to know the average difference in its neighborhood, we employ a diffusion process.

A diffusion variable $x(i, t)$ is introduced for unit i , whose dynamics at each step t is described as follows:

$$\begin{aligned} x(i, t+1) &= x(i, t) + \underbrace{K\bar{x}(i, t) - L}_{\Delta x(i, t)} \\ \bar{x}(i, t) &= \sum_{j \in n(i)} x(j, t) - c(i)x(i, t) \\ x(i, 0) &= \text{diff}(i, 0) \quad [\text{initial state}] \end{aligned} \quad (3)$$

* L is applied only to movable units

where

- $x(i, t)$: diffusion variable of the unit i at step t
- $\bar{x}(i, t)$: “flux” of $x(i, t)$ of the unit i at step t
- K : diffusion coefficient
- $n(i)$: set of indices of units neighboring the unit i
- $c(i)$: number of units connected to unit i
- L : leak const. (effective for movable units only)
- $\text{diff}(i, t)$: the value $\text{diff}(i)$ at step t (redefined)

The diffusion variable $x(i, t)$ represents the average difference around the unit i . Supposing that each unit

has a reservoir of the same size, equation (3) models a diffusion process just like water level in connected reservoirs. In this analogy, $x(i, t)$ represents the water level which averages neighboring differences through water flux between the reservoirs. Every time the connection is changed by units’ movement, $\text{diff}(i, t)$ is substituted for $x(i, t)$. The leak constant L is to make the diffusion variables approach to zero.

3.2.4 Moving strategy

Each unit determines whether it will make a motion or not according to the following inequality:

$$Gx(i, t) < \text{diff}(i, t) \quad (4)$$

The coefficient G is an activation threshold. Using (4), a unit with relatively larger difference is activated and moves in a random direction. If the goal configuration is accomplished, all the differences become zero and no more motion will be made.

4 Improved distributed formation

Although the algorithm in the previous section can realize distributed self-assembly formation, it is not provided with self-repair function. It is also prone to cause a deadlock state, that is, to converge to an undesirable shape. Moreover, there is still another problem that it takes quite a long time for units to achieve the goal shape.

Even though we have designed another algorithm [9] for many units (say, more than 30), this is not appropriate for the experimental environment with less than 20 units we are thinking about in this paper.

This section proposes a more refined algorithm to overcome these shortcomings.

4.1 Deadlock avoidance

The movable connection types “e”, “o”, and “ε” are activated by evaluating (4). This inequality implies that units never move as long as the $\text{diff}(i, t)$ equals zero. It can then happen that the formation will not proceed any more when all the $\text{diff}(i, t)$ of movable units are zero, even if non-movable units are not fit to the goal shape.

Figure 5 explains some deadlock cases. Although the goal shape is the triangle with 15 units, these deadlock shapes cannot be broken because all the movable units with type “o” have the difference 0 because they have proper neighbors $\{K, K\}$. This deadlock becomes fixed as the diffusion variable converges to zero by the leak mechanism.

To solve this problem, we add another variable called “irritation”, which is augmented in situations that are considered to be in deadlock. In the deadlock

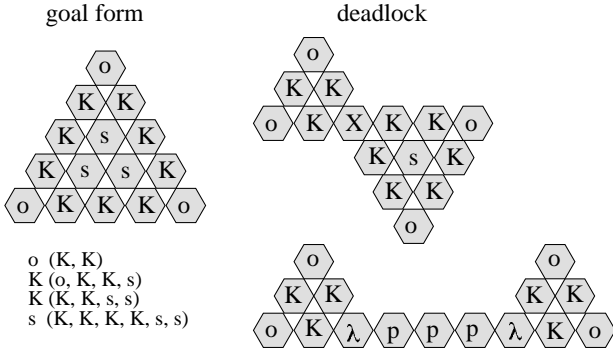


Fig. 5: Examples of deadlock

state, the non-movable units still have non-zero difference whereas the diffusion variable $x(i, t)$ converges to zero.

The unit i computes irritation $\text{irr}(i, t)$ as follows:

$$\begin{aligned}
 &\text{if } \Delta x(i, t) < x_{th} \text{ then} \\
 &\quad \text{if } \max(|\bar{x}(i, t)|, |x(i, t) - \text{diff}(i, t)|) < x_r \text{ then} \\
 &\quad\quad \text{irr}(i, t+1) = 0 \\
 &\quad \text{else } \text{irr}(i, t+1) = \text{irr}(i, t) \\
 &\quad\quad + \max(|\bar{x}(i)|, |x(i, t) - \text{diff}(i, t)|) \\
 &\quad \text{else } \text{irr}(i, t+1) = 0
 \end{aligned} \tag{5}$$

In (5), the diffusion variable $x(i, t)$ is regarded to converge when the increment $\Delta x(i, t)$ is smaller than a threshold x_{th} . The value $\text{irr}(i, t)$ then increases as $x(i, t)$ converges to zero with non-zero $\text{diff}(i, t)$.

The variable $\text{irr}(i, t)$ is reset to zero if both the average $\bar{x}(i)$ and $|x - \text{diff}(i, t)|$ become smaller than a certain value x_r . This is to prevent the units from breaking the converge to the goal shape.

We add an activation condition for movable unit:

$$\text{irr}(i, t) > I_{th} \tag{6}$$

using a threshold value I_{th} . To break the undesirable convergence of $x(i, t)$, every time (6) is satisfied, $x(i, t)$ is replaced by $|x(i, t) - \text{diff}(i, t)|$.

4.2 Motion toward smaller difference

In the previously developed formation algorithm [5], a movable unit selects its moving direction randomly, whether clockwise or counterclockwise.

In the new algorithm, the movable unit compares differences in both directions before the movement and moves in the direction of the smaller value. In addition, if a loop is detected from the local map, the direction will not be selected (see Appendix). This helps the system reduce the convergence time.

4.3 Self-repair function

The algorithm in the previous sections does not include a self-repair mechanism. Embedding this func-

tion into the algorithm has a great significance to make use of the fault-tolerant feature of a modular mechanical system. We therefore use a goal shape description (1) with priorities explained below.

Suppose a group of 6 units is going to form a triangle expressed by the description shown in Fig. 6(a). In adding a spare unit, we introduce descriptions with priorities as shown in Fig. 6(b). We give the first priority to the original triangle shape and the second to the shape with the spare unit. These descriptions are applied to the calculation of difference (2) in order of these priorities.

This simple method allows units first to build the structure with the spare unit, and to rebuild the goal shape even if one of units is removed.

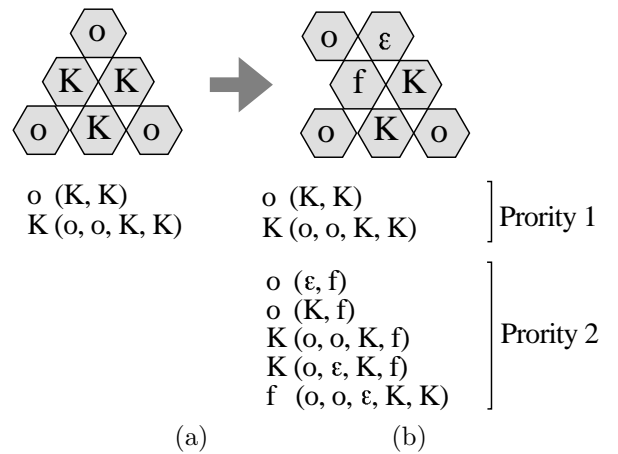


Fig. 6: Description with priorities for self-repair

5 Computer simulations

We will demonstrate the effectiveness of the distributed formation control we have developed in this paper. The improvement of convergence to the goal shape is examined first, and next, self-repair capacity.

5.1 Performance improvement

We simulated the formation of triangle by 15 units in Fig. 5 from initial straight line configuration. The parameters in the simulations are listed in Table 2.

Each unit repeats the calculation described in sections 3 and 4 at each step. In the simulations, only one unit is assumed to move at one step.

Figure 7 is the time-development of the average difference per unit until 2000 steps. The averages are calculated from 1000 simulation trials with both the improved formation algorithm and the case before improvement. The average difference becomes zero when the units accomplish the goal form.

Without improvement, the difference from the goal

Table 2: Parameters of simulations

diffusion constant	K	0.02
activation threshold ratio	G	1.25 (types "e", "o") 2.5 (type "ε")
leak constant	L	0.15 (types "e", "o") 0.02 (type "ε")
irritation increase threshold	x_{th}	0.01
irritation reset threshold	x_r	0.2
activation threshold for irritation	I_{th}	200

shape still remains at a relatively large value after 2000 steps because of deadlock, loop or slow convergence. On the other hand, the formation is almost completed as the difference is nearly equal to zero by the new algorithm. The statistics shows that the completion ratios of both the algorithms are 38.7% and 97.2% respectively. This result clearly demonstrates the effectiveness of the improved algorithm.

5.2 Self-repair

The newly added self-repair capacity is also verified. A simulation is conducted in the case of triangle of 10 units with a spare unit. Parameters are the same as Table 2. The difference and diffusion variable per unit in a typical trial case are plotted in Fig. 8.

By the goal description with priorities, the group of units converges to the configuration drawn at the bottom right of Fig. 8 at approximately 100 steps. The diffusion variable also becomes zero after the difference falls to zero owing to the leak coefficient.

To see the self-repair function, an arbitrary unit (except for type "s") is removed at around 200 steps. Even if the difference and diffusion variable rises to

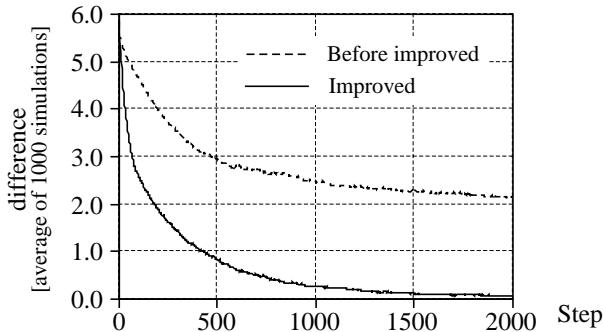


Fig. 7: Average difference by two algorithms

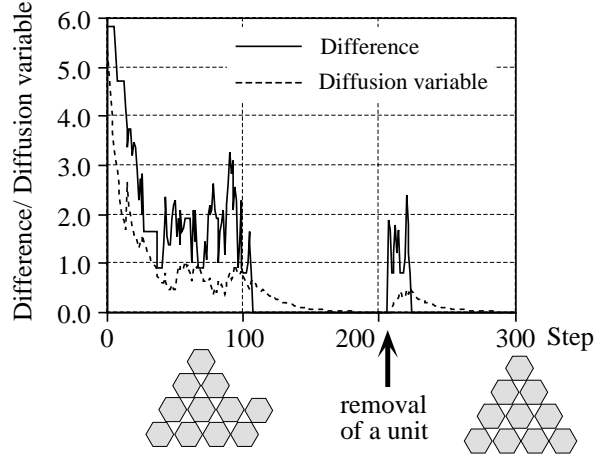


Fig. 8: Difference/diffusion variable during self-repair

non-zero for certain moment, they converge to zero and the goal triangle shape is formed again. This simulation result shows the validity of the self-repairing function.

6 Experiments

We integrated the algorithms in the previous sections and the low-level hardware control (see Appendix) into distributed formation control software, which is implemented in the hardware setup.

We carried out experiments of self-assembly control using 6 units. Each unit is programmed to achieve the goal shape of a triangle from any initial configuration. In the experiments, we start with the initial state of line configuration like the computer simulations.

Figure 9 is a series of snapshots taken from a successful formation process. We can observe that distributed formation control is brought into real motion through low-level algorithm. The goal configuration was completed without stopping in "e" type in Fig. 9 as planned in low-level control. The implementation of self-repair function is under development for the moment.

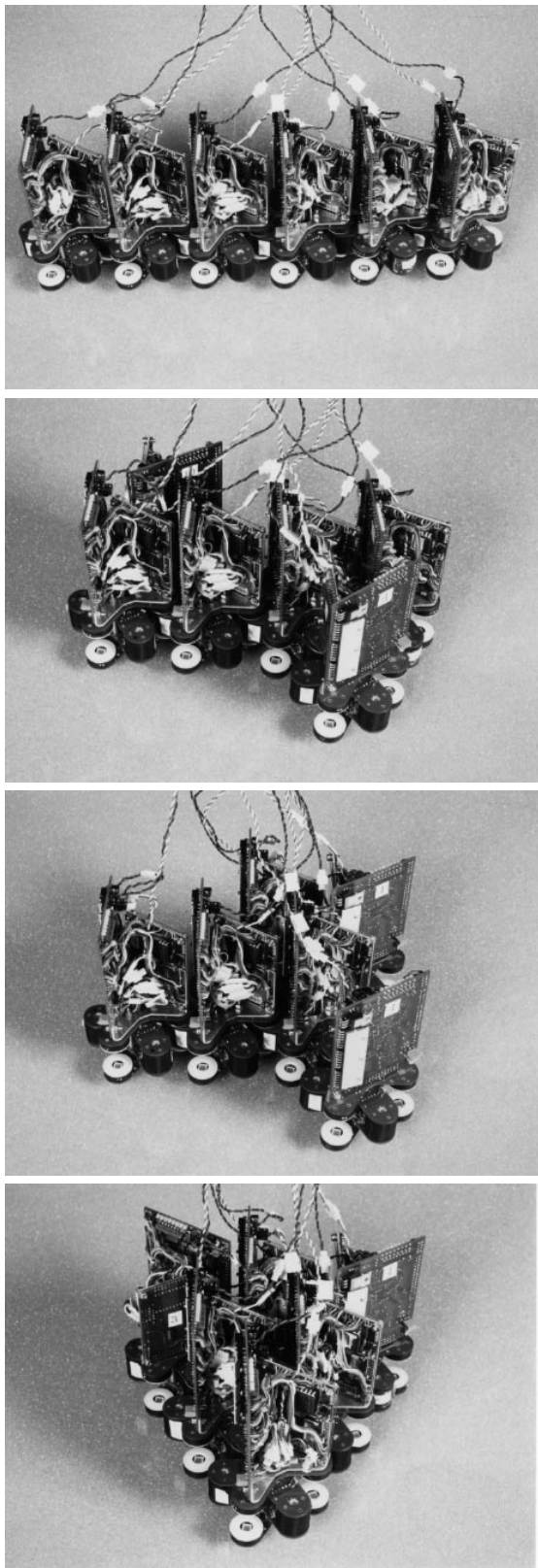


Fig. 9: Self-assembly process by 6 units

7 Conclusion

We developed a distributed formation control algorithm for a modular mechanical system and verified its effectiveness by self-assembly experiments. A robust distributed formation method is proposed based on previously developed algorithm in order to solve deadlock and to realize self-repair as well as self-assembling capacity. Computer simulations revealed that the proposed method made significant performance improvement of formation process. A low-level hardware control is also elaborated to put the formation algorithm to hardware units.

We demonstrated the sufficient feasibility of our control algorithm in real hardware systems by self-assembly experiments.

As future work, we intend to realize self-repair operation with the hardware setup and to build three-dimensional hardware units.

References

- [1] M. Yim: "New Locomotion Gaits," Proc. of Int. Conf. on Robotics and Automation, pp.2508-1524, 1994.
- [2] G. J. Hamlin, et al.: "Tetrobot Modular Robotics: Prototype and Experiments," Proc. of Int. Conf. on Intelligent Robots and Systems, pp.390-395, 1996.
- [3] T. Fukuda, et al.: "Dynamically Reconfigurable Robotic Systems," Proc. of Int. Conf. on Robotics and Automation, pp.1581-1586, 1988.
- [4] G. Chirikjian, et al.: "Evaluating Efficiency of Self-Reconfiguration in a Class of Modular Robots," J. of Robotic Systems, Vol.12, No.5, pp.317-338, 1996.
- [5] S. Murata, et al.: "Self-assembling Machine," Proc. IEEE Int. Conf. on Robotics and Automation, pp.441-448, 1994.
- [6] C. Reynolds: "Flocks, herds and schools: A distributed behavioral model," Computer Graphics, Vol.21, No.4, pp.25-34, 1987.
- [7] P. Wang, "Navigation strategies form multiple autonomous robots moving in formation," J. of Robotic Systems, Vol.8, No.2, pp.177-195, 1991.
- [8] Q. Chen, J.Y.S. Lus, "Coordination and control of a group of small mobile robots," Proc. IEEE Int. Conf. on Robotics and Automation, pp.2315-2320, 1994.
- [9] K. Tomita, et al.: "Reconfiguration Method for a Distributed Mechanical System." Distributed Autonomous Robotic System 2, H. Asama, et al., eds., pp.17-25, Springer, 1996.
- [10] S. Kokaaji, et al.: "Clock Synchronization algorithm for a Distributed Autonomous System," J. of Robotics and Mechatronics, Vol.8, No.5, pp427-434. 1996.

Appendix: Low-level hardware control

The formation algorithm explained in this paper provides the decision whether the units should move or not. To implement this algorithm into the hardware, we need low-level control such as solenoid driving and communication synchronization. This appendix outlines how hardware units are actually controlled.

We assume the followings for simplicity:

- All units have a common synchronized clock.
- Unit moves so that it avoids resulting in “e” type.

The first assumption can be satisfied using a distributed synchronization algorithm [10]. The second is introduced since a unit often has difficulty in starting movement once it stops in “e” type, which lies in an equilibrium point of overlapping magnetic fields of the neighboring magnets.

If a unit determines to move by the algorithm, the following low-level control procedure that consists of three phases is executed in a synchronous fashion.

- Local map generation:**
Each unit generates a local map of its neighbors within r connection distance.
- Negotiation:**
We should guarantee that at most one unit moves in each r connection distance to prevent a conflict caused by simultaneous movement of units. For example, this can be realized using mutual voting process.
- Movement by local cooperation:**
When a unit moves, it requires other units’ cooperation. Figure 10 (a)~(c) illustrates units possibly involved in the counterclockwise movement of unit M. First, unit L must repulse one of its solenoids and unit A, B and C should pull in turn.

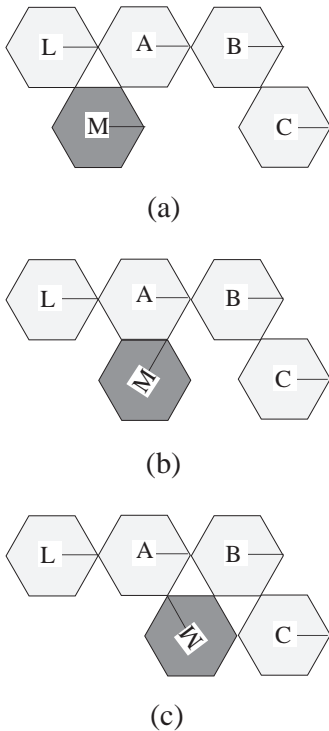


Fig. 10: Cooperation for unit M’s motion

As we noted earlier, units avoid resulting in “e” type as shown in Fig. 11. If a unit moves clockwise for instance, it continues until the resultant type is *not* “e” type. How many steps are needed is computed as j in the procedure in Fig. 12. A lookup table is used to see which solenoid is pulled or repulsed by which unit.

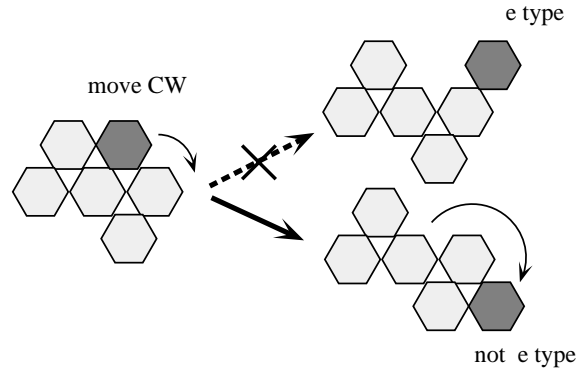


Fig. 11: Movement avoiding type “e”

```

procedure:difference_after_move()
  for  $j < 5$ 
    if [ type != ‘e’ after movement ]
      calculate_difference();
      detect_loop();
      if [ type of remainder unit  $r$ 
        after movement == ‘e’]
        difference_after_move() for unit  $r$ ;
      break;
    else  $j++$ ;
  end;
  Compare the sum of difference
  decide direction to move
  (Not move to direction
  where there will be a loop)

```

Fig. 12: The procedure of avoiding “e” type