2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Parallel Permutation for Linear Full-resolution Reconfiguration of Heterogeneous Sliding-only Cubic Modular Robots

Hiroshi Kawano

Abstract— **This paper presents a parallel permutation algorithm that achieves linear full-resolution reconfiguration of sliding-only cubic modular robots. We assume the use of a cubic module that can only slide across other modules' surfaces. The idea of a cubic modular robot with sliding-only motion primitive is a new concept that has advantages in simplifying the mechanisms of module hardware and space saving in its heterogeneous operations compared with previously studied cubic modules, such as those with sliding and convex motion primitives, or rotating motion primitives. However, because of its limited mobility, there are difficulties in managing the connectivity and scalability of the heterogeneous reconfiguration algorithm for it. To overcome these disadvantages, we introduce a parallel heterogeneous permutation method with linear operating time cost that can be incorporated into our previous full-resolution reconfiguration algorithm. We prove the correctness and completeness of the proposed algorithm. Simulation results show that the full-resolution reconfiguration algorithm that incorporates the proposed permutation algorithm reconfigures the robot structure with sliding-only cubic modules in linear operating-time cost.**

I. Introduction

A modular robot that can transform its shape by changing the configuration of identical modules with limited motion primitives is a promising platform for missions in drastically changing environments where transformation is needed to ensure the robot's mission is accomplished. It is clear that as the number of the modules increases, the robot's ability to adapt itself to such environments improves. However, in general, the costs of the planning and operation for reconfiguration increase exponentially as the number of modules increases. Therefore, developing a reconfiguration algorithm with "scalability" to control a large number of modules with reasonable algebraic calculation and operating-time costs still remains one of the most important challenges in this research field [11].

To design such a scalable reconfiguration algorithm, we must carefully consider the assumed robot module's identicalness and kinematical properties. The important kinematical properties of the modules to be considered in the design are their shapes, motion primitives allowed to them, and the usage of a meta-module-based robot structure. The meta-module-based structure (for example, a $2 \times 2 \times 2$ cubic meta-module with 8 cubic modules) makes it much easier to manage the connectivity of the robot structure; however, it restricts the variety of available robot configurations and makes the size of the robot and the operating time cost for reconfiguration much larger than those are in the case of no meta-module usage.

As for module identicalness, we must consider whether the modules are homogeneous or heterogeneous. All modules in a homogeneous modular robot are assumed to be identical, whereas some of the modules in a heterogeneous modular robot are assumed not to be identical (for example, all the modules have the same shape, but only a few modules have cameras). Therefore, heterogeneous modular robots offer more efficiency in designing modular robot hardware that is suitable for each application (for example, in the homogeneous case, all modules must have cameras, but a few modules with cameras will be sufficient in most applications). In the reconfiguration of homogeneous modular robots, each module does not have strict target positions in the start and goal configurations; however, strict target positions in the goal configuration must be assigned to some non-identical modules in the heterogeneous case. To navigate the non-identical modules to their destination, the heterogeneous reconfiguration often needs a permutation process, which is the repetition of position exchange among modules inside the robot structure. Because of the need for the permutation process, the heterogeneous reconfiguration is much more time-consuming than the homogeneous one is.

Even for cube-shaped lattice modules (hereafter cubic modules), we can see homogeneous and heterogeneous reconfiguration algorithms for several types of cubic modules with a variety of motion primitives in previous research. Buttler et al. proposed a crystal module with a compressive motion primitive and devised a distributed algorithm for its goal recognition and locomotion [1], and Rus et al. proposed reconfiguration algorithms for it that assume a meta-module-based structure [2]. Vassilvitskii et al. proposed a complete tunneling reconfiguration algorithm for homogenous compressible cubic modular robots with a $2\times2\times2$ meta-module-based structure [3]. For homogeneous $2 \times 2 \times 2$ meta-module-based compressive cubic modular robots, Aloupis et al. proposed a linear reconfiguration algorithm applicable within the union of boxing spaces of the start and goal configurations [4]. Romanishin et al. proposed a pivoting cubic modular robot [6] based on a mechanism proposed by Gajamohan et al. [5], and Sung et al. proposed a homogeneous reconfiguration algorithm for it [7]. Fitch et al. proposed a reconfiguration algorithm for heterogeneous cubic modular robots with sliding and convex motion primitives [8]. Their reconfiguration algorithm is quadratic in operating time cost. Fitch et al. researched a reconfiguration algorithm for cubic modular robots with sliding and convex motion in an environment with obstacles [9]. They also proposed a sublinear locomotion algorithm for homogeneous cubic

Hiroshi Kawano is with the NTT Corporation, NTT Communication Science Laboratories, Kanagawa, Japan (phone: +81-46-240-3185; fax: +81-46-240-4716; e-mail: hiroshi.kawano.yb@hco.ntt.co.jp).

modular robots with sliding and convex motion primitives [10]. While no hardware design of the cubic modules that use sliding and convex motions has been studied, Suzuki et al. proposed a hardware design for sliding-only cubic modules working in two-dimensional spaces and proposed algorithms for their reconfiguration and crawling motions [16]. Suzuki proposed a hardware design for sliding-only cubic modules working in three-dimensional spaces using helical magnetic actuators [17]. Kawano has proposed a full-resolution reconfiguration algorithm for heterogeneous cubic modular robots with only a sliding motion primitive [12]. The algorithm takes quadratic operation time cost in the reconfiguration process. Kawano has also proposed a $2 \times 2 \times 2$ meta-module based tunneling reconfiguration algorithm for heterogeneous sliding-only cubic modular robots [13][14][15]. Although the algorithms have limitations on the available robot configurations, they accomplished a linear reconfiguration using only the space provided by the start and goal configurations.

II. Motivation

Only from the analysis of the previous research, we do not have a clear answer to the question of which motion primitive for cubic modular robot is the best. This is because each of the motion primitives for cubic modules has both advantages and disadvantages. The crystal module with a compressive motion primitive has an advantage in hardware feasibility; however, there is difficulty in keeping the connectivity of the robot structure without adopting a meta-module-based structure. A heterogeneous solution for crystal modules has still not been studied. Adopting a rotating motion primitive easily accomplishes the management of the connectivity of the full-resolution robot structure with a feasible hardware design; however, there is a difficulty in space saving of heterogeneous reconfiguration because the rotating module needs a large hole to enter the robot structure during the permutation process. Cubic modules with sliding and convex motion primitives have strong advantages in saving space in their heterogeneous permutation (because it can enter the robot structure via the hole with one module size) and easing the management of connectivity of the full resolution robot structure; however, the feasibility of the hardware design is a serious issue. Cubic modules with only a sliding motion primitive have difficulty in keeping the connectivity of the full resolution robot structure; however, the hardware design is more feasible than that of cubic modules with sliding and convex motion primitives. The advantage of space saving in the heterogeneous permutation still remains in sliding-only cubic modules. Therefore, if fast (linear) heterogeneous reconfiguration for full-resolution sliding-only cubic modular robots is realized, we can enjoy almost all the advantages of cubic modules with a variety of motion primitives by only using sliding-only cubic modules. However, a linear heterogeneous solution for sliding cubic modular (in both with nor without convex motion cases) robots has been studied only in a meta-module-based structure case but not yet in the full-resolution case. Considering this situation, here we study a linear heterogeneous permutation algorithm that can be incorporated in the reconfiguration algorithm for sliding-only cubic modular robots with full-resolution configuration (full-resolution means that no meta-module based structure is not used to form the start and goal configurations of robots).

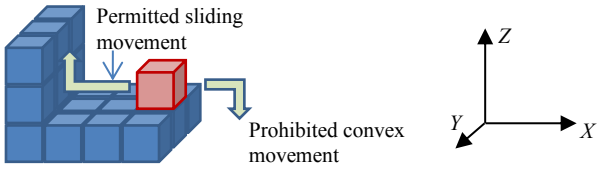

Figure 1. Limited Sliding Cube module that is allowed only sliding movement across other modules' surfaces.

We show that the proposed algorithm is correct and complete with linear operating-time cost for connected robot structures that can be homogeneously transformed from an arbitrary three-dimensional full-resolution configuration of robot structures.

III. Primitives

A. Limited Sliding Cube

We assume a self-reconfigurable robot model composed of multiple lattice modules. The geometrical properties of the assumed lattice module are very similar to that of *Sliding Cube* proposed by Fitch et al. [8]: the module is a three-dimensional cube that has a connector on each surface and can connect face-to-face to any other module. The module is also uncompressible. The difference between *Sliding Cube* and the assumed module is that the latter has only one motion primitive: the module is only allowed to slide across the surface of other modules; a convex transition around other modules is not allowed (Fig. 1). We call a module with such a property *Limited Sliding Cube*. A Limited Sliding Cube (LSC) module can travel reversibly on surfaces that intersect perpendicularly in a concave arrangement (Fig. 1). Because convex motion is forbidden, it needs another adjacent module for tunneling motion to go one step out of the robot structure. This makes it difficult to secure the existence of a mobile module, where a mobile module is defined as one that can move without disconnecting the robot structure. To guarantee module mobility, the intermediate configuration of the robot structure during the reconfiguration process must meet a certain condition.

We define R as a self-reconfigurable robot forming a connected three-dimensional structure with Limited Sliding Cube modules. We suppose that robot structure R has n modules. To describe the positions of modules i ($i = 1, 2,,, , n$) in R, we define a Cartesian coordinate system (Fig. 1). X and Y axes are in the horizontal plane, and the Z axis is perpendicular to the horizontal plane. X, Y, and Z coordinate values of the position of each module (x_i, y_i, z_i) are integers. Each module has an identified position in the start configuration S and in the goal configuration G. As is proved in [12], R cannot be reconfigured to any two- and one-dimensional structure. Therefore, the proposed algorithm only treats the three-dimensional reconfiguration of R.

IV. Reconfiguration

A. Overview of the reconfiguration method

We study the design of the linear reconfiguration algorithm for R by extending the quadratic reconfiguration algorithm previously proposed in [12]. The previous quadratic algorithm once transforms R in a homogeneous manner into an intermediate configuration (hereafter M_p) in which the

permutation process is executed. The condition for the shape of M_p is determined by considering the maintenance of mobility during the permutation. After the permutation in M_p, R in M_p is transformed to G in a homogeneous manner. The movement of the modules in these homogeneous transformation is considered in the permutation process. As the example of the reconfiguration results in [12] show, about 50% of the operating-time cost for the total reconfiguration is taken by the permutation process. In general, the permutation is the most time consuming process in the heterogeneous reconfiguration. The linearization of the transformation processes in [12] other than the permutation in M_p is easily accomplished by introducing parallel navigation of multiple modules. However, the linearization of the permutation in M_p presents some difficult issues in accomplishing completeness and correctness. Therefore, the main issue in the linearization of the previous quadratic algorithm in [12] is the linearization of the permutation process.

B. Requirement from linear permutation

It can be said that an algorithm is linear if it executes the permutation process within the linear operating-time cost without disconnection in R and without any deadlocking. Generally, the operating-time cost for one module's navigation to its target position is $O(n)$, and that for position exchange at the target position is constant bound. Therefore, in cases where non-parallel navigation in which only one module moves at the same time is adopted, the total operating-time cost for the n module's permutation is $n \times O(n) = O(n^2)$. Considering this, we introduce parallel navigation and position exchange of multiple modules to accomplish linear permutation. To make the operation time cost be linear, we need a method that makes at least n / n_c modules (n_c is a small constant value independent from n) navigate at the same time. The most serious risk for disconnecting R arises in the position exchange of the modules. This is because some void spaces are needed for the position exchange process in spite the limited mobility of LSC. Considering the maintenance of connectivity in position exchange, we introduce a new meta-module-based structure in M_p which is different from that in [12] and let the modules exchange positions inside the meta-module space. To avoid deadlocking of the permutation process, the most important issue is to prevent the modules navigating in parallel from colliding with each other. Because n / n_c modules must navigate at the same time to make the algorithm linear, the free space adjacent to the surface of R that can be used for avoiding collisions among navigating modules is seriously limited (only a few voids adjacent to n_c modules can be used for collision avoidance.). In general, the calculation of a collision-free path for many modules navigating in such a severe situation is itself a very difficult problem. Considering this, we set a condition on the shape of M_p so that there is one closed unicursal path p_{uni} on the surface of M_p in which all access points for the target positions of the navigating multiple modules are included. Collisions among the navigating modules are avoided by letting all navigating modules track the closed unicursal path that leads each one to its target position.

C. Requirement from correct permutation

As preparation for the discussion of the correctness of the permutation algorithm, suppose the target position of module m_{i_j} (m_{i_j} is the ith module in the jth set C_j that is defined later) is the position of m_{i+1_j}, the target position of module m_{i+1_j} is the position of m_{i+2_j}, ..., and the target position of module $m_{i+nchainj\_j}$ ($n_{chainj} < n$) is the position of m_{i_j} when the permutation starts. Here we refer to the jth set of modules that contains only m_{i_j}, m_{i+1_j}, m_{i+2_j}, m_{i+3_j}, ... , $m_{i+nchainj\_j}$ as C_j ($0 < 2j < n$). R taking M_p has at least one C_j. To guarantee the correctness of the permutation algorithm, the algorithm must meet the following conditions:

Conditions_For_Correctness:

1. At least one member module of each C_j starts its navigation to its target position.

2. After the position exchange between navigating module m_{i_j} and module m (usually m is m_{i+1_j}) occupying the target position of m_{i_j} is carried out, m starts the navigation to its target position (if m is m_{i+1_j}, m goes to the position of m_{i+2_j}).

It is easily proved inductively that the first condition guarantees the start of the permutation process and that the second condition guarantees that the navigation of all module m_{i_j} in C_j is invoked by the arrival of module m_{i-1_j} at the position of m_{i_j}. The solution for meeting these conditions is obtained simply by letting each meta-module in M_p containing at least one module that has not started its navigation to its target eject a module every time the position exchange between the navigating module and module stored in the meta-module is carried out.

D. Condition for M_p

We introduce a meta-module for position exchange (MMPE) to form a new M_p composed of 2×2 square + one or two modules adjacent to the 2×2 square in the negative direction of the Z axis from it [hereafter, leg modules (LMs)] [Fig. 2 (a)]. The 2×2 square in the MMPE retains connectivity in the position exchange, and LMs are used to eject a navigating module from the MMPE. It is clear that one void per 2×2 square in the MMPE forming M_p can exist without disconnecting R. Therefore, parallel position exchanges executed in all MMPEs at the same time do not disconnect R, if each position exchange process generates only one void. The proposed algorithm uses only one void in the position exchange. By virtue of the existence of at least one LM in each MMPE, each MMPE can eject one module on its top surface without disconnecting R, and after the ejection, there is no void in the 2×2 square of each MMPE. Therefore, no module navigating on the top surface of M_p causes any disconnection in R. To guarantee the existence of the closed unicursal path p_{uni}, we set the condition for the shape of M_p composed of MMPEs as:

Condition_for_M_p: [for example, see Fig. 2 (b)]

(1) The 2×2 squares of the MMPEs in M_p form a horizontal rectangular plane.

(2) The number of the MMPEs contained on one side of the rectangle is an even number and that on the perpendicular side is larger than 1.

Lemma 1: *R with $24 \geq n \geq 20$, $36 \geq n \geq 30$, $48 \geq n \geq 40$, or $n \geq 50$ can be reconfigured into an arbitrary configurations meeting* ***Condition_for_M_p.***

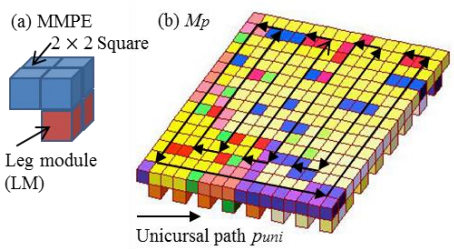

Figure 2. Examples of (a) MMPE and (b) the unicursal path p_{uni} on the table top of R that takes M_p.

Proof: It is clear that MMPEs arranged in 2 × 2, 2 × 3, 2 × 4, or 2 × L (L ≥ 5) rectangle meet ***Condition_for_Mp*** *(24 ≥ n ≥ 20, 36 ≥ n ≥ 30, 48 ≥ n ≥ 40, and 12L ≥ n ≥ 10L in each reactangular arrangement respectively). It is clear that 12L ≥ 10(L + 1) > 10L ≥ 50. Let's refer to a configuration that can be obtained by compressing all LMs in R taking M_p in positive directions of X and Y axes by the method in [12] as $M_{p'}$. As is proved in [12], arbitrary R with n ≥ 20 can be reconfigured to $M_{p'}$. After the reconfiguration from R to $M_{p'}$, all compressed LMs in $M_{p'}$ can go to any places under 2 × 2 squares of MMPEs without disconnecting R.* ■

E. Position exchange in MMPE

After the module has ended its navigation and position exchange, it is preferable that it does not move anymore. In cases where there are such modules only in the 2 × 2 square of the MMPE, any LM can be used to fill the void generated by the ejection of the new navigating module in the 2 × 2 square. However, if all LMs in the MMPE have already ended their navigation, one of the LMs in it must be used to fill the void. We refer to such a module that fills the void after it has ended its navigation as a plug module (PM). The motion of the PM must be carefully managed until it returns to its target position again. Considering the management of PMs, we propose the following method for the module ejection at the start of the permutation process and a method for the position exchange control in each MMPE.

Module_Ejection_At_Start(*M*): (Executed in MMPE *M*)

If the 2 × 2 square part of *M* has a module *m* that is not at its target position, **then** {// CASE A
_Eject *m* and fill the void generated by the ejection of *m*
_with one of the LMs (*m'*). If *m*' is at its target position,
_*m'* becomes PM. }
Else if there is an LM *m* in *M* that is not at its target position, **then** {//CASE_B Eject *m*. }

Position_Exchange(*M*): (*m* is the module on *M*.)

If module *m'* in *M* is at the target position of *m*, **then** {1
_**If** *m'* is not a PM, **then** {2
__**If** *m'* is not an LM, **then** {3 // CASE 1
___**Exchange1(*m*, *m'*)** (Fig. 3 (a)) 3}
__**Else**{4 // CASE 2
___**Exchange2(*m*, *m'*)** (Fig. 3 (b)). 4} 2}
_**Else**{5 (*m'* is a PM.)
__**If** module *m''* in *M* that has not started its navigation
__exists, **then** {6
___**If** *M* has an LM, **then** {7
____**If** *m'* is an LM, **then** {8 // CASE 3 (never happens) 8}
____**If** *m'* is not an LM and *m''* is LM, **then** {9// CASE 4
_____**Exchange1(*m*, *m'*)** (Fig. 3 (a)), (*m'* goes home.)
_____**Exchange3(*m'*, *m''*)** (Fig. 3 (c)). 9}
____**If** *m'* and *m''* are not LMs, **then** {10 //CASE 5
_____**Exchange1(*m*, *m'*)** (Fig. 3 (a)),
_____**Exchange1(*m'*, *m''*)** (Fig. 3 (a)). 10} 7}
___**Else** {11 (*m'* is a PM.) // CASE 6
____**Exchange4(*m*, *m'*, *m''*)** (Fig. 3 (d)). 11} 6}
__**Else** {12 (All modules in *M* are after their navigation.)
___**If** *M* has an LM, **then** {13
____**If** *m'* is an LM, **then** {14 // CASE 7 (never happens) 14}
____**Else**, {15 (*m'* is a PM, and *m'* goes home.) // CASE 8
_____**Exchange1(*m*, *m'*)** (Fig. 3 (a)),
_____**Exchange6(*m'*)** (Fig. 3 (f)). 15} 13}
___**Else** {16 (*m'* is a PM, and *m'* goes home.) // CASE 9
____**Exchange_5(*m* , *m'*)** (Fig. 3 (e)). 16} 12} 5} 1}
Else {17 (The target of *m* is a void under the 2 × 2 square.)
_**If** *m''* that has not started its navigation exists, **then** {18
__**If** *m''* is an LM, **then** {19 // CASE 10
___**Exchange3(*m*, *m''*)** (Fig. 3 (c)). 19}
__**Else**{20 // CASE 11 (*m* becomes a PM.)
___**Exchange1(*m*, *m''*)** (Fig. 3 (a)). 20} 18}
_**Else** {21 // CASE 12
__**Exchange6(*m*)** (Fig. 3 (f)). 21} 17}

Exchange1–6: Exchange positions as shown in Fig. 3.

Lemma 2: *Every time* ***Position_Exchange*** *is executed, M ejects one of the modules that has not started its navigation to the MMPE containing its target position.*

Proof: Conditional branches by ***If then*** *rules in* ***Position_Exchange*** *cover all cases of whether the target position is occupied by module m' in M or not; m' is one of the leg modules of M or not; m' has started its navigation to its target or not; there is a module m'' (≠ m') that has not started its navigation to its target or not (in case where m' is PM); and m'' is one of the leg modules of M or not. It is clear that CASEs 3 and 7 of* ***Position_Exchange*** *never happens. (If m' is a leg module and at its target position, it is not PM. Therefore, the target position of no module is at the position of m'.)*

PM generation is only in CASE 11 of ***Position_Exchange*** *and CASE A of* ***Module_Ejection_At_Start****. In these cases, only one module in M is appointed to a PM. In CASE 11 of* ***Position_Exchange****, the module originally located at the void target position of m is not a PM in M because m goes to the void position. Therefore, in this case, there is no PM in M before the execution of* ***Exchange1****. In only CASEs 5 and 6 of* ***Position_Exchange****, the position of the PM is changed, but no additional module is appointed to another PM. Therefore, M has at most only one PM during the permutation. The PM is released in CASEs 4, 8, and 9 of* ***Position_Exchange****. In CASE 4, there is no need for a PM because one of the LMs is ejected. In CASEs 8 and 9, M has no module that needs navigation to its target position. In all cases where M does not need a PM, the module used as the PM is released and sent back to its target position. In CASEs 4, 8, and 9, one PM is released. Therefore, no PM remains after their executions.* ■

F. Linear permutation algorithm

Now let's look at the whole procedure in the proposed permutation algorithm. First, by executing **Module_Ejection_At_Start**, the algorithm ejects one module per MMPE in cases where there is at least one module in the

MMPE that is not at its target position. By this ejection of modules, there are at most $n/5$ modules (because each MMPE has at least five modules) on the table top of M_p composed of 2 × 2 squares of MMPEs. Next, each ejected module checks whether it is on the MMPE containing its target position or not. When some of the ejected modules are on the MMPEs containing their target positions, **Position_Exchange** is executed by the MMPEs containing their target positions**.** When the execution of **Position_Exchange** is completed in all MMPEs**,** each module on the table top of M_p starts its navigation to the MMPE that contains its target position. Each navigating module tracks unicursal path p_{uni}, which is discussed in section IV. B. All navigating modules move simultaneously. If some of the navigating modules arrive at the MMPE with their target position, all modules pause their navigation and **Position_Exchange** is executed by the MMPE containing the target positions of the navigating modules on it**.** After completion of the executions of **Position_Exchange**, new navigating modules are on the MMPEs that executed **Position_Exchange**. Until all modules arrives at their target positions, these operations are repeated.

The navigation strategy that makes modules on the table top of M_p track the unicursal path is appropriate at the beginning of the permutation process. This is because there is not enough space on the crowded surface of M_p for navigating modules to take the more efficient short-cut paths to their destination. However, as the permutation process progresses, the number of modules navigating in parallel decreases and more efficient short-cut paths for the navigation of modules become feasible without collisions. Therefore, we also propose an additional method that makes navigating modules leave the unicursal path and take a more efficient path to their destination. In the proposed method, the navigating module goes to the neighboring vacant 2 × 2 square if it can arrive at its target position earlier by tracking the unicursal path from the vacant neighboring 2 × 2 square than by tracking the unicursal path from the 2 × 2 square where the module currently exists. The proposed permutation algorithm is as follows:

Linear_Permutation_Algorithm:
(M_k is the kth MMPE in R)
1: **For** all k, **do** at the same time {[1]
  _Module_Ejection_At_Start(M_k). [1]}
2: **For** all k, **do** at the same time {[2]
  _If M_k has the target position of the module on it, **then**
  __Position_Exchange(M_k). [2]}
3: **For** all k, **do** at the same time {[3]
  _If there is a neighboring MMPE M' that will be vacant two time steps from now, the navigating module on M' does not come to M_k, and the module on M_k can arrive at its target earlier by tracking p_{uni} from M' than by tracking p_{uni} from M_k, **then** {[4] Let the module on M_k move two steps toward M'. [4]}
  _Else, {[5] Let the module on M_k move two steps toward the next MMPE in p_{uni}. [5]} [3]}
4: **If** there is a module on the table top of M_p, **then** Go to 2.
  Else, end.

Lemma 3: ***Linear_Permutation_Algorithm*** *correctly navigates each module in R meeting* ***Condition_for_M_p*** *to its unique destination.*

Proof: Because each MMPE in R has at least one LM, each MMPE can eject one navigating module on its top surface without disconnection if the MMPE has a module that is not at its target position. Suppose at least one member module of C_j is ejected as the navigating module and no member module of $C_{j'}$ is ejected at the beginning of the permutation by ***Module_Ejection_At_Start****. From lemma 2, all member modules of C_j go to their target position by repeating* ***Position_Exchange****. The member modules of $C_{j'}$ are ejected as the new navigating module if the MMPE executing* ***Position_Exchange*** *has no other modules that has not started their navigation to their target position (see CASEs 4, 5, 6, 10, and 11 of* ***Position_Exchange****). If no navigating module arrives at the MMPE containing the member modules of $C_{j'}$ during the permutation, the MMPE must not have a member module of C_j that is not at its target position at the beginning of the permutation; therefore, this situation is contradictory. Once the member modules of $C_{j'}$ are ejected, from lemma 2, all member modules of $C_{j'}$ go to their target position by repeatedly executing* ***Position_Exchange****.*

Even if there is a PM in the 2 × 2 square part of M when CASEs 1, 2, 10, and 12 of ***Position_Exchange*** *are executed, it is guaranteed that there is a module in R (that is not m) whose target position is the position of the current PM. This is because, the PM is at the original position of the module that has started its navigation to its target position (See CASE A of* ***Module_Ejection_At_Start****, and CASEs 5, 6 and 11 of* ***Position_Exchange****). As proved here, it is guaranteed that all modules in R that are not at their positions at the beginning of the permutation start their navigation to their target positions. Therefore, the PM remaining in M when the position exchanges of CASEs 1, 2, 10, and 12 are executed could be sent back to its target position by the execution of CASEs 4, 8, or 9 of* ***Position_Exchange*** *when the module whose target position is the PM's position arrives at M (this is also a possibility after several executions of CASE 5 or 6 of* ***Position_Exchange****).* ■

Lemma 4: ***Linear_Permutation_Algorithm*** *executes the permutation of R meeting* ***Condition_for_M_p*** *in linear operating-time cost without disconnecting R.*

Proof: Suppose N_{no_eject} MMPEs do not eject any navigating module by ***Module_Ejection_At_Start*** *at the beginning of the permutation. In such a case, the number of the module that need navigation to their target position is at most $n - 5\ N_{no\_eject}$ (hereafter, $n_{navi} = n - 5\ N_{no\_eject}$), and at least $n_{navi}/6$ modules are ejected as navigating modules by* ***Module_Ejection_At_Start****. Even if no additional navigating module is ejected until all these $n_{navi}/6$ navigating modules arrive at their target and exchange positions, the time cost for them reaching their target positions is O(n). By simply repeating this at most six times (because each MMPE has at most six modules), all n_{navi} modules can reach their target positions. Therefore, the total cost for the navigation of n_{navi} modules is $6 \times O(n)$. The time cost for each position exchange is constant bound; therefore, the total operation time cost for position exchanges of n_{navi} modules is $O(n_{navi}) < O(n)$. It is clear from Fig. 3 that there are no voids in the table top of M_p when the navigating modules move, and the execution of* ***Position_Exchange*** *only generates one void per MMPE.* ■

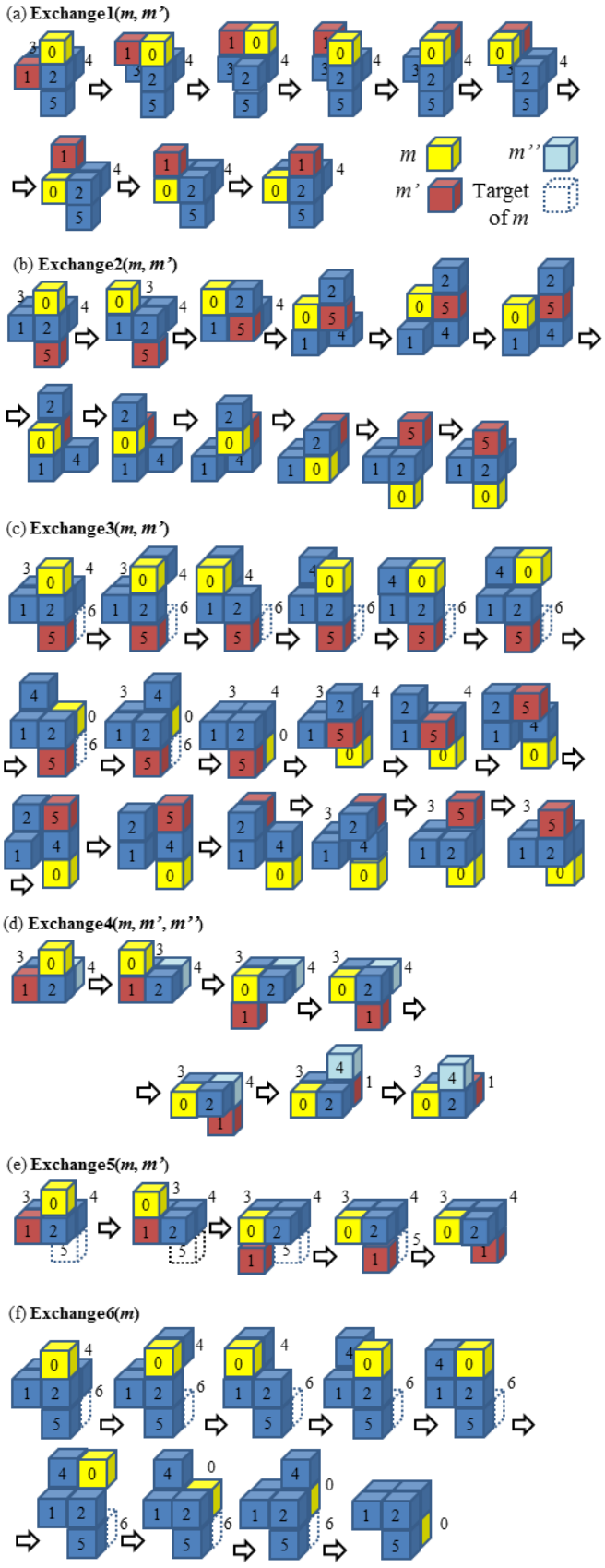

Figure 3. Position exchange process by **Exchange 1–6** [shown by (a) –(f)**]**.

V. Discussion

By setting the strict condition for the arrangement of MMPEs in M_p, the proposed permutation algorithm need not adopt any ordinary search algorithm for deciding the unicursal path for the navigating modules. However, if the condition for the shape of the M_p is released, NP-complete problem for searching Hamiltonian cycle path must be solved to obtain the closed unicursal path p_{uni} for navigating modules. In that case, the arranging the MMPEs in a Hamiltonian graph structure would be useful.

Finally, we prove the correctness and linear completeness of the reconfiguration algorithm for R.

Theorem 1: *Arbitrary heterogeneous R with $n \geq 50$ can be reconfigured into an arbitrary three-dimensional connected structure in linear operating-time cost.*

Proof: From lemma 1, arbitrary R with $n \geq 50$ can be reconfigured to M_p in linear operating-time cost by adopting the parallel navigation of multiple modules in the homogeneous transformation method described in [12]. From lemmas 3 and 4, ***Linear_Permutation_Algorithm*** *correctly permutates R in M_p with linear operating-time cost without disconnecting R.* ■

[18] provides the path planning method for multiple robots that fully fills grid environments with $O(n^2)$ planning cost. However, in the proposed method, there may be cases where some of the simulutaneously navigating modules have the same target MMPE. [18] does not consider such cases. While the reconfiguration algorithm described in [15] for R in a $2 \times 2 \times 2$ meta-module based structure can be executed with limited spaces inside $S + G$, the reconfiguration by the proposed algorithm still needs free space. Therefore, next challenge in this research is to reduce of the space needed in the full-resolution reconfiguration of R so that the algorithm can be applied to environments with obstacles. Together with this improvement, we are now working on implementating the proposed algorithm in distributed form.

VI. Result

The video demonstration shows a simulation reconfiguration when the proposed permutation algorithm is incorporated into the full-resolution reconfiguration algorithm in [12]. In this simulation, R consisting of 322 LSCs is reconfigured from a beetle-shaped starting configuration S to a butterfly-shaped goal configuration G. The table top of M_p is an 16×16 rectangle composed of 64 MMPEs. During the time steps from 699 to 2,142, the proposed permutation algorithm is applied (it takes 1,443 time steps). After about 1,000 time steps from the start of the permutation, we can see several navigating modules take short-cut paths. It takes 2,731 time steps ($< 18n + 6n = 5{,}796 + 1{,}932 = 7{,}728$, where 18 is the number of time steps needed in **Exchange3**) to complete the whole reconfiguration process. We can see that the permutation by the proposed algorithm is more than three times faster than the permutation algorithm based on single navigation in [12] (with the previous quadratic algorithm, it takes 10144 and 4921 time steps for the full reconfiguration and permutation, respectively).

REFERENCES

[1] Z. Buttler, R. Fitch, D. Rus, "Distributed Control for Unit-Compressible Robots: Goal-Recognition, Locomotion, and Splitting," *IEEE/ASME Trans. Mechatronics*, vol. 7, no. 4, Dec. 2002, pp. 418–430.

[2] D. Rus, M. Vona, "A Basis for Self-reconfiguring Robots using Crystal Modules," in *Proc.2000 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pp. 2194-2202, Takamatsu, Japan, Oct., 2000.

[3] S. Vassilvitskii, M. Yim, J. Suh, "A Complete, Local and Parallel Reconfiguration Algorithm for Cube Style Modular Robots," in *Proc.2002 IEEE Int. Conf. Robotics and Automation*, pp. 117-122, Washington DC, May, 2002.

[4] G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, S. Ramaswami, V. Sacristán, S. Wuhrer, "Linear Reconfiguration of Cube-Style Modular Robots", *Computational Geometry – Theory and Applications*, vol. 42, no. 6-7, pp. 652–663, 2009.

[5] M. Gajamohan, M. Merz, I. Thommen, and R. D'Andrea, "The Cubli: A Cube that can Jump Up and Balance", in *Proc. 2012 IEEE/RSJ International Conference of Intelligent Robots and Systems*, pp. 3722-3727, Algarve, Portugal, October, 2012.

[6] J. W. Romanishin, K. Gilpin, S. Claici, D. Rus, "3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions", in *Proc. 2015 IEEE International Conference on Robotics and Automation*, pp.1925-1932, Seattle, WA, May, 2015.

[7] C. Sung, J. Bern, J. Romanishin, and D. Rus, "Reconfiguration Planning for Pivoting Cube Modular Robots", in *Proc.2015 IEEE Int. Conf. Robotics and Automation*, pp. 1933-1940, Seattle, Washinton, May, 2015.

[8] R. Fitch, Z. Butler, D. Rus, "Reconfiguration Planning for Heterogeneous Self-Reconfiguring Robots," in *Proc.2003 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pp. 2460-2467, Las Vegas, NV, Oct., 2003.

[9] R. Fitch, Z. Butler, D. Rus, "Reconfiguration Planning Among Obstacles for Heterogeneous Self-Reconfiguring Robots," in *Proc.2005 IEEE Int. Conf. Robotics and Automation*, pp. 117-124, Barcelona, Spain, April, 2005

[10] R. Fitch, Z. Butler, "Million Module March: Scalable Locomotion for Large Self-Reconfiguring Robots," in *The Int. Journal of Robotics Research*, Vol. 27, no. 3-4. pp. 331-343, Mar.-Apr., 2008.

[11] K. Stoy, D. Brandt, and D. J. Christensen, *Self-Reconfigurable Robots An introduction*, MIT Press, 2010.

[12] H. Kawano, "Full-resolution Reconfiguration Planning for Heterogeneous Cube-shaped Modular Robots with only Sliding Motion Primitive", in *Proc.2016 IEEE Int. Conf. Robotics and Automation*, pp.5222-5229, Stockholm, Sweden, May, 2016.

[13] H. Kawano, "Tunneling-Based Self-Reconfiguration of Heterogeneous Sliding Cube-Shaped Modular Robots in Environments with Obstacles", in *Proc.2017 IEEE Int. Conf. Robotics and Automation*, pp.825-832, Singapore, May, 2017.

[14] H. Kawano, "Distributed Tunneling Reconfiguration of Sliding Cubic Modular Robots in Severe Space Requirements", in *Proc.14th Int. Symposium on Distributed Autonomous Robotic Systems, paper no. 1*, Boulder, CO, Oct., 2018.

[15] H. Kawano, "Linear Heterogeneous Reconfiguration of Cubic Modular Robots via Simultaneous Tunneling and Permutation", in *Proc. 2019 IEEE Int. Conf. Robotics and Automation*, pp. 332-338, Montreal, Canada, May 2019.

[16] Y. Suzuki, N. Inou, H. Kimura, M. Koseki, "Reconfigurable group robots adaptively transforming a mechanical structure - Crawl motion and adaptive transformation with new algorithms-", *Proc. 2006 IEEE Int. Conf. on Intelligent Robots and Systems*, pp. 2200-2205, Beijing, China, October, 2006.

[17] Y. Suzuki, "Modular robot using helical magnet for bonding and transformation", in *Proc.2017 IEEE Int. Conf. Robotics and Automation*, pp.2131-2137, Singapore, May, 2017.

[18] J. Yu, "Constant Factor Time Optimal Multi-Robot Routing on High-Dimensional Grids", in *Proc.2018 Robotics: Science and Systems*, pp.2131-2137, Pittsburgh, PA, June, 2018.