A Motion Generation Method for a Modular Robot

Eiichi Yoshida* † Satoshi Murata** Akiya Kamimura*

Kohji Tomita* Haruhisa Kurokawa* Shigeru Kokaji*

* Distributed System Design Research Group, Intelligent Systems Institute,

National Institute of Advanced Industrial Science and Technology (AIST)

1-2-1 Namiki, Tsukuba-shi, Ibaraki 305-8564 Japan

e-mail: e.yoshida@aist.go.jp

** Department of Computational Intelligence and Systems Science,

Interdisciplinary Graduate School of Science and Engineering,

Tokyo Institute of Technology,

4259 Nagatsuta-cho, Midori-ku, Yokohama, Kanagawa 226-8502 Japan

Abstract

This paper discusses motion generation of a homogeneous modular robot called Modular Transformer (MTRAN). The modules are designed to be self-reconfigurable so that a collection of the modules can transform themselves into a robotic structure. The motion generation of the self-reconfigurable robot is indeed a computationally difficult problem due to many combinatorial possibilities of module configuration, even though the module itself is simple with two degrees of freedom. This paper will describe a motion generation method for a class of multi-module structure, based on a motion planner and a motion scheduler. The motion planner has two-layer structure with global and local planners. The former is in charge of planning overall movement of the cluster whereas the latter decides locally coordinated module motions called motion schemes. After the motion is generated as a sequence of single motion schemes, the motion scheduler processes the output plan to allow parallel motions to improve the efficiency. The effectiveness of the motion generator is verified through a many-module simulation.

Key Words: Modular Robotic System, Self-reconfiguration, Motion Planning

*† Corresponding author

1 Introduction

Self-reconfigurable modular robots have been intensively investigated in recent years. Especially, homogeneous self-reconfigurable robots that can adapt themselves to the external environment by changing their configuration and are also self-reparable by using spare modules. This type of robot is useful in harsh environments where the adaptability and the self-maintainability becomes a significant factor. They can serve as planetary exploring vehicles, robots for searching survivors in rubble, or inspection robots in nuclear plants. Many interesting hardware design for three-dimensional (3D) modular robots have been proposed [1)−6)].

There have been a number of studies on distributed and centralized methods for generation of modular robots. We have also developed several distributed methods for two-dimensional and three-dimensional homogeneous modular robots [8, 9)]. These methods enabled them to self-assemble and self-repair in a distributed manner using local inter-module communication. In contrast, most of other methods are based on centralized planning. For instance, Kotay et. al [10)] developed a motion synthesis method for a class of module groups. Ünsal et. al [6)] reported multi-level motion planners for a bipartite module composed of cubes and links, based on heuristic graph search between module configurations. These methods are dedicated to modules that have sufficient degrees of freedom to move to every neighboring lattice position.

This paper seeks to build a methodology of generating motions of a modular robot called *Modular Transformer*, or *MTRAN*, developed in AIST [7)]. MTRAN is a new type of modular robot that can generate both static structure and dynamic robotic motion. This novel feature is realized by simplified design of a module and a special connecting mechanism using magnets. Although we have shown MTRAN can form various shapes such as a legged walking robot or a crawler-type robot, its motion generation is not straightforward because of restricted degrees of freedom and non-isotropic spatial property of movability of a module. The necessary motion combination should be correctly planned for each particular local configuration. Also, a huge search space should be explored to check the interchangeability between two arbitrary module configurations and the collision between modules in 3D space. For these reasons, it is currently difficult to find some generic laws of motion generation or to directly apply our distributed methods for MTRAN.

This paper therefore focuses on building a feasible motion generator for a particular class of module cluster by narrowing the search space of module motion as a first step to more general motion generation method. The module cluster we will investigate is serially connected cube-like “blocks,” each of which is composed of four modules. We will propose a motion generator that outputs a block-based motion.

The motion generator consists of a *motion planner* and a *motion scheduler* to generate module motions that allows the module clusters to move along a desired trajectory. The motion planner is two-layered, and includes global flow planner and local motion scheme selector. The former outputs possible module paths to realize the overall cluster motion. The latter selects valid paths and comprises them by collecting appropriate locally coordinated module motions based on a rule database. These rules take account of non-isotropic module movability by associating appropriate pre-planned motion schemes with various local configurations. The motion scheduler parallelizes the generated motion plan by allowing motion steps in multiple motion schemes to be executed in parallel. This improves the efficiency of module motion by reducing the steps of motions. The proposed method is classified as a centralized motion generator assuming that all the information of modules in the cluster is available.

After briefly introducing the hardware design and model description in Section 2, the structure of the motion generator is addressed in Section 3. Sections from 4 to 7 give detailed descriptions of the each component of the generator. Finally, Section 8 concludes the paper and discusses the future development of the motion generator.

2 Hardware Design and Model Description

2.1 Design of MTRAN

A module of MTRAN consists of two semi-cylindrical parts connected by a link (Fig. 1). Servomotors are embedded in the link part and each of the parts can rotate by 180°. Each module has six connecting surfaces (three for each part) that can actively connect and disconnect to other modules by using magnets and shape memory alloy (SMA) actuators. The connecting surfaces are equipped with electrodes for power supply and serial communication. All the connected modules can be supplied power if one of them is connected to the power source. In each module a microcomputer BasicStamp II (Parallax Inc.) using PIC16C57 processor (Microchip Technology Inc.) is mounted that receives commands from a host PC and drives servomotors and SMA actuators.

2.2 Model description

Each semi-cylindrical part of a module is identified as p1 and p2 in the model description. The position and orientation of the module m are uniquely determined by specifying the position and orientation of one part, together with the rotation angles of the servomotors. Here, Σ_m is a local coordinate system fixed on p1 of module m where the axes z_m (rotation axis) and y_m (the symmetrical axis of the semi-cylindrical part) are defined as shown in Fig. 1. Let Σ_0 be the absolute coordinate system. The position and orientation of module m is determined by using:

- the central position $p_m(x, y, z)$ of p1 with respect to Σ_0,
- the orientation of basis vectors z_m and y_m of Σ_m with respect to Σ_0,
- the absolute rotation angles of each part (θ_1, θ_2).

Hereafter, we assume that both parts of modules move only on orthogonal-lattice grid and that the rotation angles (θ_1, θ_2) are limited to either 0° or ±90° for simplicity. A unit length of the lattice grid is defined as the length between the two rotational axes of a module. A module therefore occupies two adjacent points in the grid.

We also denote the connection faces as C_{iz+}, C_{iz-} and C_{iy} (i = 1, 2 for p1, p2) according to Σ_m. The state of a connecting face, $S(\text{face})$, takes either of the following:

`T(ID)`	Connecting to module ID
`T(*)`	Connecting to a module but ID not specified
`F`	No module connected

The connection state of a module is written as $[S(C_{1z+}), S(C_{1z-}), S(C_{1y})]$, $[S(C_{2z+}), S(C_{2z-}), S(C_{2y})]$. For example, Fig. 2 shows the initial configuration of two modules shown in Fig. 3, which is described as follows.

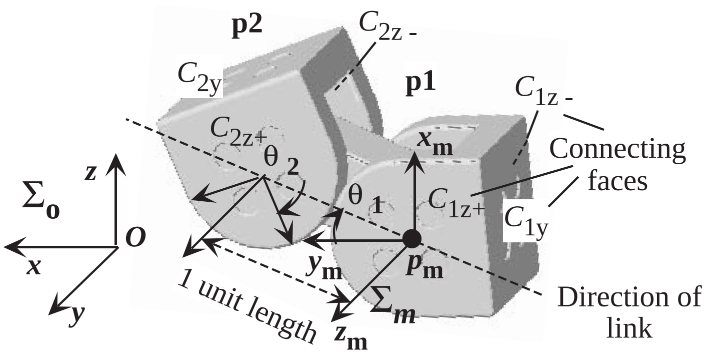


Fig. 1: A module of MTRAN.

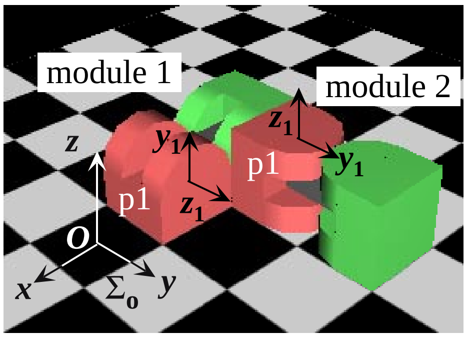


Fig. 2: An example of module configuration.

ID 1 $p_m(-1,0,0)$ $z_m(0,1,0)$ $y_m(0,0,1)$
$(\theta_1,\theta_2) = (-90°, 0°)$,
connection state: $[$`F`, `F`, `T(*)`$]$, $[$`T(2)`, `F`, `F`$]$

ID 2 $p_m(-2,1,0)$ $z_m(0,0,1)$ $y_m(0,1,0)$
$(\theta_1,\theta_2) = (0°, 0°)$
connection state: $[$`F`, `T(*)`, `T(1)`$]$, $[$`F`, `T(*)`, `F`$]$

2.3 Motion description

When a module makes a motion, one of the parts should be attached to another module to keep the connectivity. We call this fixed part a *base part*. A *motion step* is described using module IDs, base parts, rotation angles and the number of carried modules and their IDs if any.

A *motion sequence* is a collection of these motion steps. Figure 3 shows an example of two-module motion sequence in which module 1 reorients module 2 starting from the initial state in Fig. 2. In step 1, there are two moving modules, and the module ID 1 rotates by relative angles ($\Delta\theta_1$ = 90°, $\Delta\theta_2$ = 0°) with base part p1 and carries another module ID 2, which does not make rotation, and so on. This motion sequence consists of two steps and is described as follows.

```
step 1
 ID 1 base p1 rot(90, 0)    carry 1 ID 2
 ID 2 base p1 rot(0, 0)     carry 0

step 2
 ID 1 base p1 rot(-90, 90)  carry 1 ID 2
 ID 2 base p1 rot(0, 0)     carry 0
```

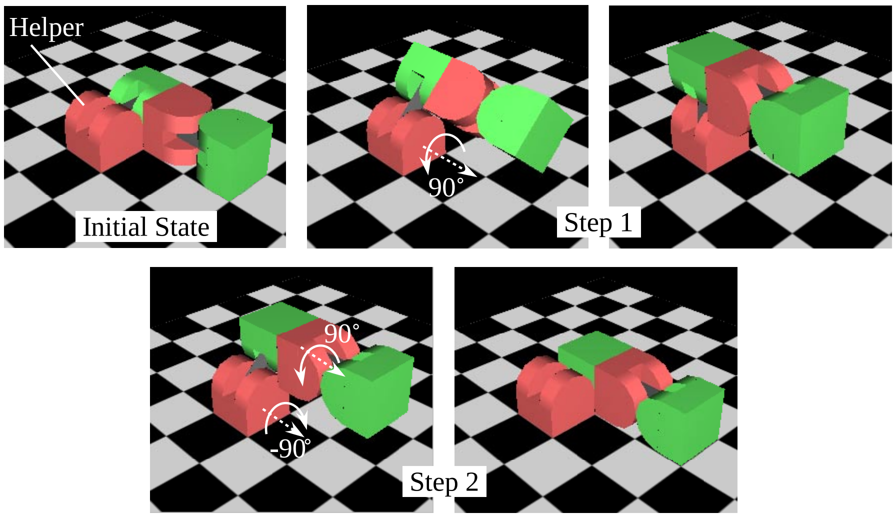


Fig. 3: Mode conversion from pivot to forward-roll.

3 Structure of Motion Generator

The goal of motion generator is to let a cluster of MTRAN modules move along a certain given three-dimensional trajectory in the lattice grid (Fig. 4). For instance, the trajectory corresponds to a path that a plant inspection robot or a planetary explorer should trace. This allows the module cluster to move into narrow space or to go over the obstacle. The motion generator should output appropriate motion sequence that realizes the cluster motion guided along the desired trajectory.

Nevertheless, the search space is too large to be explored to generate an arbitrary motion for an arbitrary configuration[1]. To develop a feasible motion generator, we consider a particular class of module clusters (Fig. 5) composed of four-module *blocks* that look like large cubes. All the rotation axes (z_m) of the modules in a block are oriented in the same direction whereas y_m axes of different layer are orthogonal. The reasons why we adopt this cube-like block among several possibilities are:

[1]For N modules, at each step there are $\sum_i^N {}_N C_i$ possibilities at maximum to choose which module to move. For each case, each module can take maximum 9 possible states for angles (θ_1, θ_2), even using discrete angle 0° and $\pm 90^\circ$. Base part selection has also 2 possibilities.

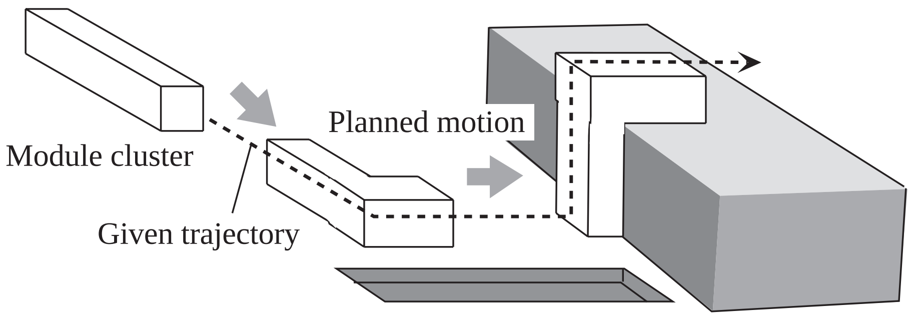
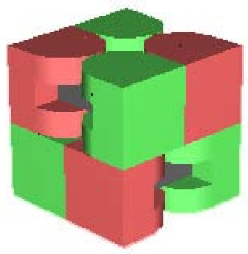


Fig. 4: Planning of cluster motion.



A block by 4 modules.

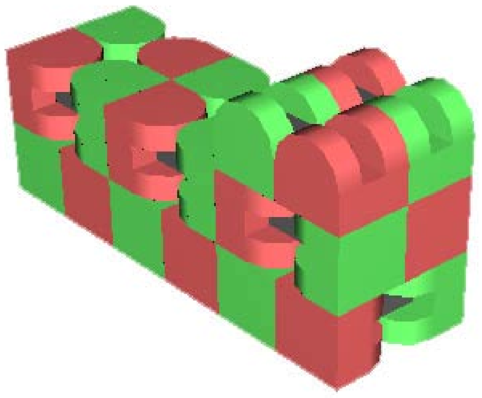


Fig. 5: A cluster composed of two layers with two converter modules.

(1) various sizes of 3D structures can be easily configured since the block has an isotropic shape,

(2) it is the smallest block that has such isotropic shape that can be connected at any of its faces,

(3) a global motion along 3D trajectory can be planned in a simple way on the basis of blocks, and

(4) the connectivity of all the modules is maintained in a cluster composed of these block.

A couple of modules that have different direction of rotation axes, called converters, are attached to on top of the cluster. The converter modules are used to change the direction of rotation axes modules in the chain cluster.

Given the trajectory as a block-base motion as an input, the motion should be planned for each module. However, the module's non-isotropic geometrical property makes it difficult to obtain the motion sequence straightforwardly. Since a module has only two parallel rotation axes, its three-dimensional motion usually requires a combined coordinated motion sequence of other surrounding modules. If this motion sequence is not carefully planned, the resultant sequence may not be possible for such reasons as inappropriate orientation of rotation axes, collision between modules, or loss of connectivity during the motion. Since generally applicable laws have not been found for planning these motion sequences, some database of rules to look up is necessary.

In this paper, we propose a motion generator composed of a *motion planner* and a *motion scheduler*. The motion planner has a two-layered architecture to cope with the complexity of the planning problem. The upper layer decomposes the planning problem into subproblems solvable by the lower layer. The lower layer is designed to solve simplified planning problems based on a database of rules for each local configuration. After the motion planner outputs a serial sequence of motion schemes, the motion scheduler processes it into a motion plan including motion steps that can be executed in parallel. This makes use of the concurrent feature of the modular robot and increases the efficiency be reducing the total time required for the plan.

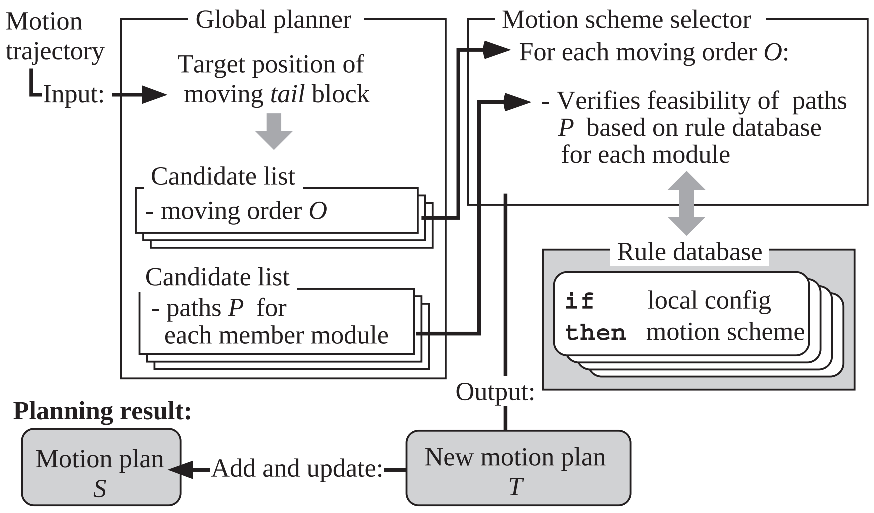


Fig. 6: Motion planner architecture.

4 Motion Planner Architecture

This section gives more detailed description of the motion planner. The upper and lower layers of the motion planner are called the *global flow planner* and the *local motion scheme selector* respectively. As shown in Fig. 6, the global flow planner searches possible module paths and motion orders to provide the global cluster movement, called *flow*, according to the desired trajectory. This is realized as a motion of a block such that the tail block is transferred toward the given heading direction. The local motion scheme selector verifies if the paths generated by the global planner are valid for each *member* module of the block based on rule database. If a given path from the global planner turns out to be valid, the selector updates the motion plan by adding a set of local reconfiguration motion sequences called *motion schemes*. Otherwise it tries another possible module path generated by the global planner. The selector copes with the non-isotropic property of module movability by associating the coordinated motion with the corresponding local configuration in the form of rules. Note that this is a centralized planning method assuming that all the information of modules in the cluster is available.

In the following planning method, we give the following assumptions:

(1) One module can lift only one other module.

(2) Only one motion scheme is allowed at a time.

(3) At least two converter modules are assumed in the whole cluster.

(4) The flow direction should go straight at least by two unit lengths.

The first assumption comes from the limited torque capacity of the hardware. The second one will be relaxed when the motion scheduler processes the plan. The remainders are introduced to simplify the planning problem.

5 Global Flow Planner

The input to the global planner is the desired trajectory of the cluster. The cluster *flow* is defined as the trace of block motion, where the *tail* block is removed and put at the other end as the new *head*, as shown in Fig. 7. By one block motion, the head of the cluster moves by two unit length on the lattice grid. Among several way of generating this kind of a block motion, we adopt simple motion schemes sending modules one by one towards the head. The modules move on the side of the cluster (Fig. 7).

The output of the global planner are the possible paths $\mathcal{P}_{mi}$ $(i = 0, 1, \ldots, N_P)$ for each member module m in the tail block and its motion orders $\mathcal{O}_i$ $(i = 0, 1, \ldots, N_o)$, where N_P and N_o are the numbers of candidate paths and orders respectively. An order $\mathcal{O}_i$ describes in what order the four member modules in a block moves along the corresponding path.

A path $\mathcal{P}_{mi}$ is derived by tracing lattice positions on the side of the cluster, starting from the initial tail position until the module reaches one of target positions next to the current head block (Fig. 8). A module may have multiple target positions and paths, and their number varies depending on the cluster configuration. After the tail block motion is completed, it becomes a new head block. Then the next tail will be sent to the head, and so forth.

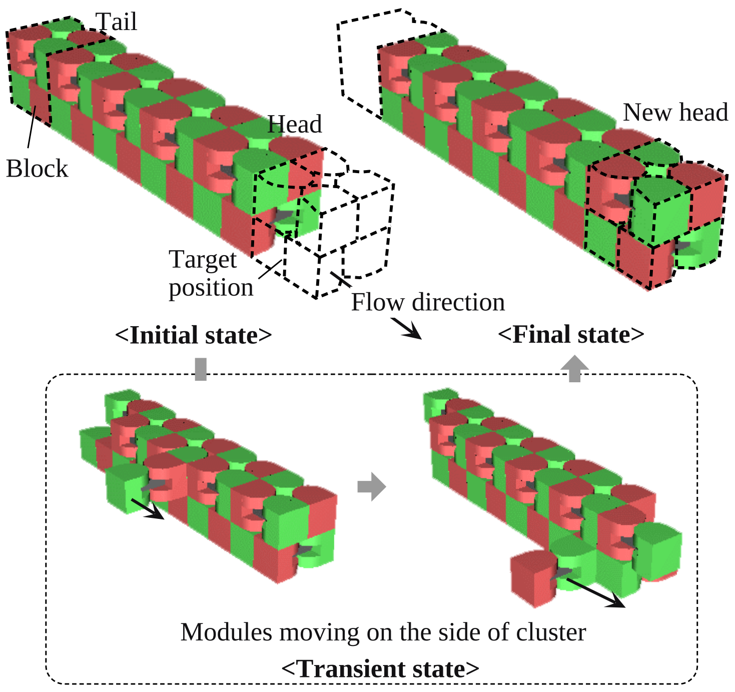


Fig. 7: Example of block motion.

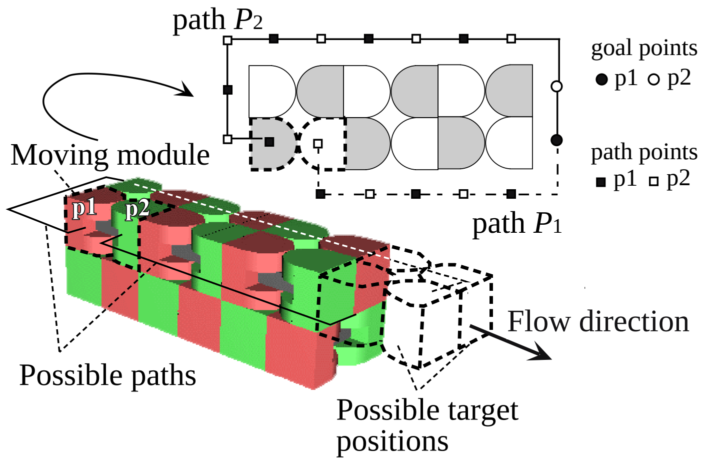


Fig. 8: Path of a module for block motion.

The order of applying motion steps $\mathcal{O}_i$ should be decided in such a way that the connectivity of whole cluster is maintained. For instance, consecutive transportation of the two modules in the upper layer of the tail block in Fig. 8 are not allowed because the connectivity condition of two lower modules is violated when the two upper modules move.

6 Motion Scheme Selector

Based on the output of the global planner, appropriate *motion schemes* should be selected to achieve the planned block motion, considering connectivity and collision avoidance. The *motion scheme selector* play this part using a database of rules. In the following, after outlining the selection procedure, we will detail rule description and matching, and validity check of module paths.

6.1 Selection procedure

According to the motion order $\mathcal{O}_i$ $(i = 0, 1, \ldots, N_o)$ given from the global planner, the selector verifies the validity of possible paths $\mathcal{P}_{mj}$ $(j = 0, 1, \ldots, N_P)$ of each member module m in the block, in increasing order of traveling distance. Namely, the path with the shortest length is first tried, next the second shortest, and so on. Each rule includes a motion scheme associated with an initial configuration that is described as a connectivity graph (Fig. 9a). Among the rules that matches the current local configuration, a motion scheme that gives the largest forward movement is selected. The motion scheme of the selected rule is stored in the temporary motion sequence $\mathcal{T}$. If all the motion steps of the member modules are correctly determined, the planner updates the motion plan $\mathcal{S}$ by appending the output sequence $\mathcal{T}$ to it. Otherwise, the selector tries next possibilities of $\mathcal{P}$ or $\mathcal{O}$.

6.2 Rule description and matching

A rule R_k $(k = 1, 2, \ldots, N_R)$ in the database is composed of a `if`-condition part and a `then`-action part, where N_R is the total number of rules. The former is a connectivity graph G_k that describes a local connection state to be matched to the current local configuration of the moving module. The latter corresponds to a motion scheme M_k written in the form of motion sequence.

Figure 9b illustrates the graph description of local configuration. In the connectivity graph G_k, a node is assigned to each module. The node includes such data as a temporary ID number, rotation angles and the states of the six connecting faces. To make the rules applicable to various cases, we introduce a wild card state "*" (don't care) that matches all the states.

An arc in the connectivity graph denotes the connection to other modules and specifies the relative direction of z and y axes of connecting module m, such as $[(z(m), y(m)]$. Every module configuration

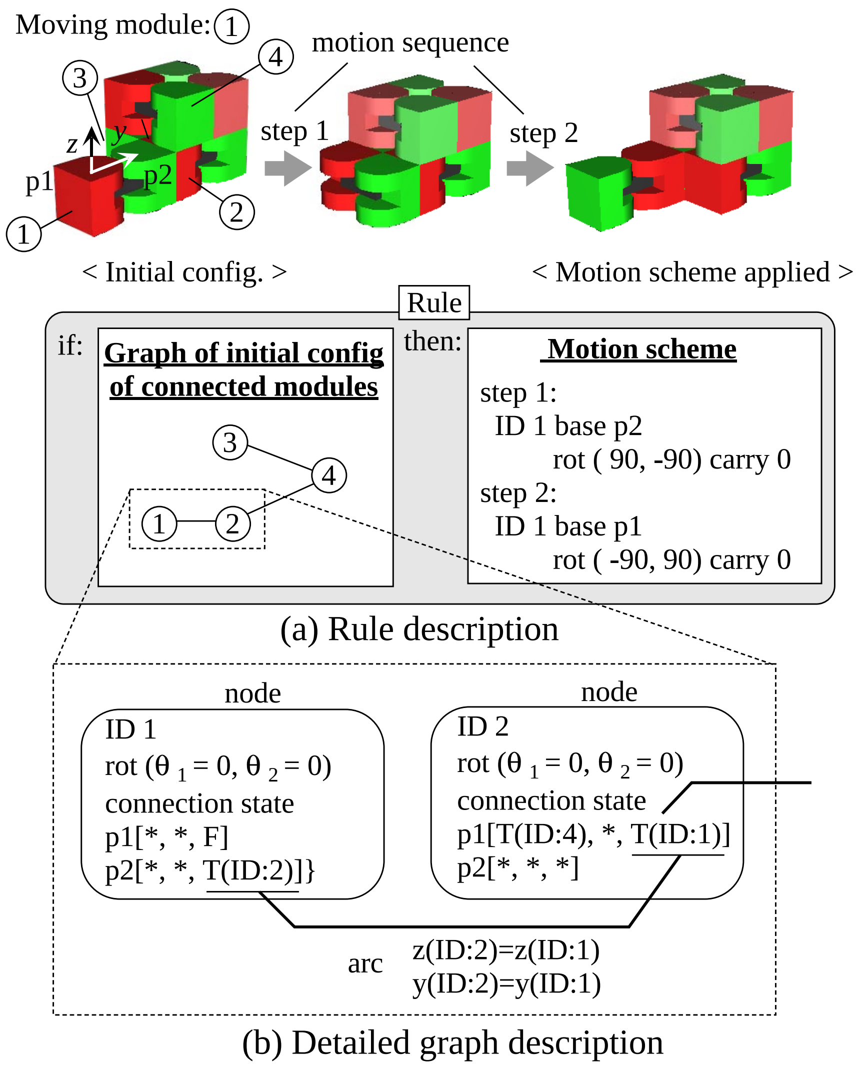


Fig. 9: Example of a rule for a rolling motion scheme

can be described in the this graph form starting from one top node. These rules are currently manually coded.

To find a motion scheme of a module m for the given path $\mathcal{P}_{mj}$, the selector searches rules that matches the local configuration of m. Let G_m be the current connectivity graph of module m. Matching between G_m and rule templates G_k ($k = 1, 2, \ldots, N_R$) proceeds from the top node down to the connecting nodes. During the matching process, the connection states and rotation angles are compared for corresponding nodes, as well as the connecting directions in each arc. The graph matching succeeds if all the nodes and arcs turned out to be compatible. All the matching possibilities are tested for each rule, such as mirrored configurations and configurations where the parts p1 and p2 are swapped. The selector makes a list of all the matched rule R_k to check the validity as described next.

In order to implement the motion scheme selector, we extracted several fundamental motion schemes as follows.

(1) rolling on a side of a straight cluster (Fig. 9)

(2) carrying a module along a trajectory whose heading direction changes by right-angle on a plane (no direction change of rotation axes).

(3) converting the direction of rotational axes of a module using converter modules.

(4) moving the converter modules to appropriate positions.

For those basic motion schemes, we have extracted approximately 30 basic rules, which are currently hand-coded.

6.3 Validity check of a module path

For each rule R_k ($k = 1, 2, \ldots, N_f$) that matched to the current configuration G_m, the validity of associated motion scheme M_k is checked. If there exist valid motion schemes, then the selector chooses the one that gives the largest forward movement for the path $\mathcal{P}_{mj}$.

The validity check is performed from two aspects, collision avoidance and connectivity of total cluster. By applying the motion scheme M_k to the module m, collision can be detected by calculating the sweeping area of its motion steps. Similarly, the connectivity is examined during the motion by tracing the connected modules from module m in the cluster.

When more than one rules are found valid, one of the motion schemes is selected based on some additional criteria, such as the maximum traveling distance along the path.

7 Motion Scheduler

The output of the planner described so far is a sequence of motion schemes $\mathcal{S}$ to achieve the desired trajectory. However, as shown in the assumption (2) in Section 4, only one motion scheme is allowed at a time in this plan. This severely limits the concurrent feature of the modular robot because other modules do not move even if they can.

The motion scheduler is devised to improve the efficiency through parallel execution of multiple motion schemes.

7.1 Parallelizing a motion plan

The plan $\mathcal{S}$ can be decomposed into a series of motion sequences M_i ($i = 1, 2, \ldots, 4 \times N_B$), each of which corresponds to a single path $\mathcal{P}$ for a i th moving module m_i, where N_B is the total number of moving blocks. This M_i is a unit element of scheduling. Initially, the current pointer t_i is set to 1 for each M_i.

Figure 10 shows how the motion scheduler works. The parallelization process of a plan proceeds by incrementing t_i in the sequences M_i, starting from M_1. Suppose the first non-finished motion sequence is M_i during scheduling. For every motion step at t_i in M_i, the next sequence M_{i+1} is tested

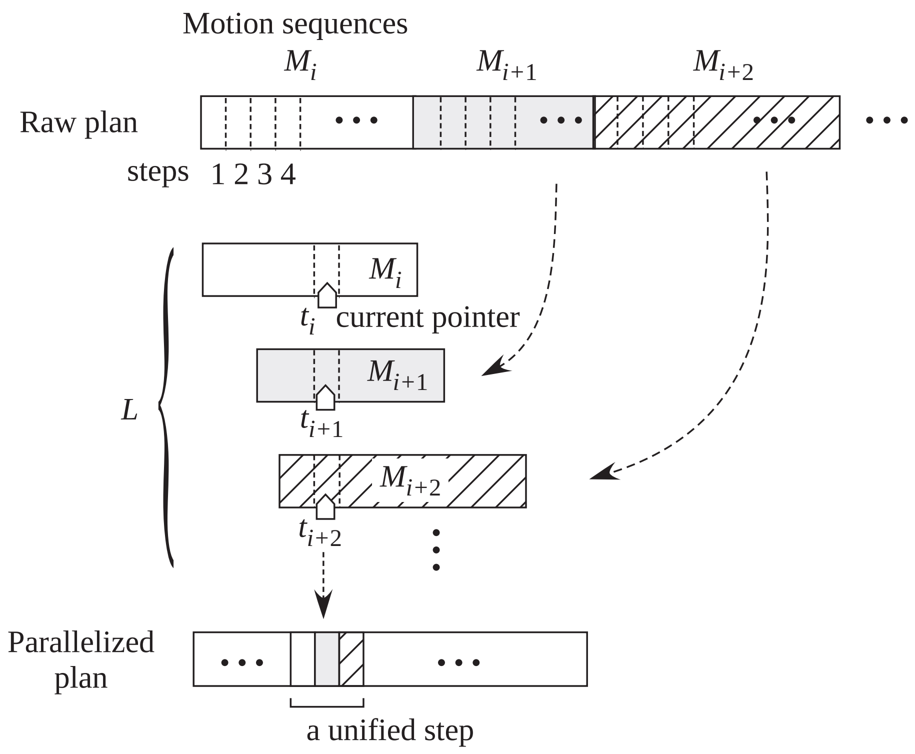


Fig. 10: Parallelization of plan by motion scheduler.

if the motion step at their current pointer t_{i+1} can be executed in parallel. The scheduler verifies:

- whether the ${t_i}^{\text{th}}$ motion step of M_i and ${t_{i+1}}^{\text{th}}$ motion step of M_{i+1} can be executed at the same time, avoiding collision and keeping the connectivity of the cluster, and
- whether the M_i can correctly terminate when the ${t_{i+1}}^{\text{th}}$ motion step of M_{i+1} is made in parallel with ${t_i}^{\text{th}}$ motion step of M_i.

The first verification is done basically by the same validity checker described in 6.3. The second verification is necessary because it may happen that M_i cannot be completed by inserting other motion steps during its execution. For this verification, the whole sequence of M_i should be simulated from the parallelized motion steps.

If the above parallel motion steps turn out to be feasible, the next motion sequence M_{i+2} is tested. In this case, after checking collision and connectivity, the correct termination of both M_i and M_{i+1} should be confirmed in the second verification. In this way, up to L in total motion sequences are tested until no more motion steps are found executable in parallel. Then those motion steps executable in parallel are unified into one motion step. After these motion steps are parallelized, the motion scheduler increments the current pointers t_i, $t_{i+1}, \ldots$ for motion sequences M_i, $M_{i+1}, \ldots$ The same procedures are repeated throughout the plan $\mathcal{S}$ generated by the motion planner. As a result, module motion $\mathcal{S}'$ using parallel motion sequence are derived.

7.2 Generated motion

The motion generation framework described so far is applied to a module cluster composed of 22 modules. The desired trajectory includes horizontal and vertical direction changes of cluster flow as shown in Fig. 11. At the initial state, the flow direction is y direction. First, the cluster first changes the flow in $-x$ direction and advances by three blocks. It then moves by one block in z direction after the vertical direction change. Figure 12 shows some snapshots taken from the generated motion. At the final step 199, the converters are at the positions where they are involved in transferring the final

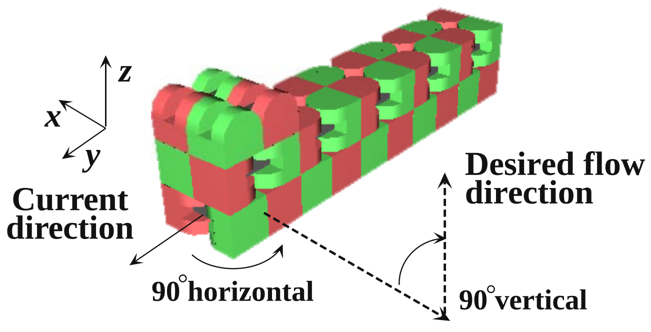


Fig. 11: Desired trajectory of module cluster

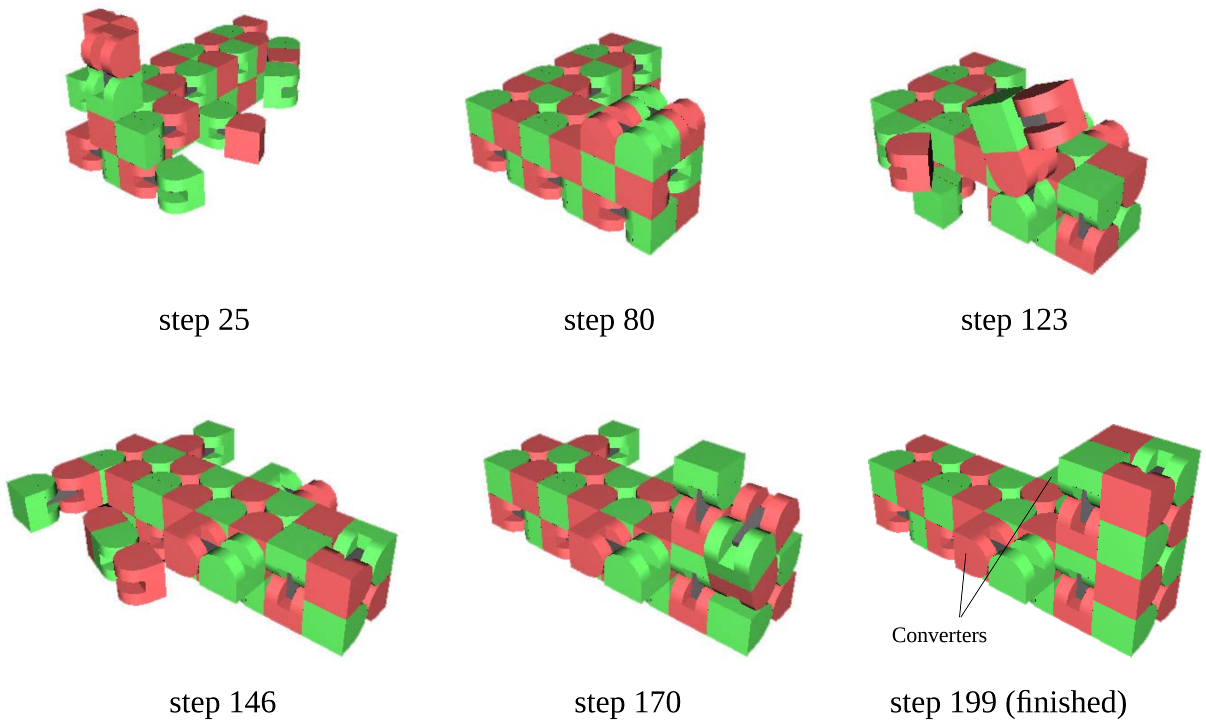


Fig. 12: Generated plan for desired motion in Fig. 11.

block.

The raw plan generated by the motion planner takes 354 motion steps, where only one motion scheme is allowed at one step. After processing the plan by the motion scheduler, the length was reduced down to 199 steps. The motion scheduler tries to put motion steps of maximum three motion sequences in parallel ($L = 3$). In this example, the length of the plan was shortened by 44%, which means that the efficiency cluster motion was greatly improved. The further increase of L did not affect the result in this case because after this level of parallelization nearly no more motion steps can be interleaved throughout the plan. In this way, the parallelism of the modular robot can be fully exploited so that the efficiency can be close to the maximum.

8 Conclusions and Discussions

This paper discussed motion generation of a self-reconfigurable modular robot MTRAN designed to generate both static structure and dynamic robotic motions. We proposed a motion generator composed of motion planner and motion scheduler. The motion planner has a two-layered architecture, global flow planner and local motion scheme selector. The former part provides the possible paths and motion orders to realize the flow of the cluster. The latter combines a series of motion schemes based on a rule database to make the flow. The motion scheduler interleaves the motion sequences in the plan generated by the planner to improve the efficiency of cluster motion. The simulation result showed that the total motion steps of motion plan was greatly reduced by the motion scheduler.

In spite of the limited class of applicable structures in this paper, we believe our approach will be effective for other classes. To extend this framework, we intend to extract necessary basic rules sets that are valid for various classes of module clusters. Depending on the problems, “meta-rules” for how to use these rules should also be investigated. Evolutionary method will be tested for acquisition of these basic rules and meta-rules. The motion scheduler is considered to be widely applicable to improve the efficiency of the module motion through parallelism. Future work also includes the evaluation of this efficiency by investigating theoretically optimal motion. On hardware side, we are also aiming to implement the motion planner to the hardware modules. The usage of sensors will be indispensable to adaptation in real world. By integrating motion generator framework with a sensor system, we are aiming to realize a modular robot that can move around in environments with bumps or walls, adapting its shape to the outside world.

References

1) S. Murata, et al.: “A 3-D self-reconfigurable structure,” *Proc. IEEE Int. Conf. on Robotics and Automation*, 432–439, 1998.

2) K. Kotay, et al.: “The self-reconfiguring robotic molecule,” *Proc. IEEE Int. Conf. on Robotics and Automation*, 424–431, 1998.

3) P. Will, et al. : “Robot modularity for self-reconfiguration,” *Proc. SPIE, Sensor Fusion and Decentralized Control in Robotic Systems II*, 236–245, 1999.

4) A. Casal and M. Yim: “Self-reconfiguration planning for a class of modular robots,” *Proc. SPIE, Sensor Fusion and Decentralized Control in Robotic Systems II*, 246–257, 1999.

5) A. Castano, et al. “Autonomous and self-sufficient CONRO modules for reconfigurable robots,” *Distributed Autonomous Robotic Sytems 4*, Parker L E, et al. eds., Springer, 155–164.

6) C. Ünsal, et al. : “A modular self-reconfigurable bipartite robotic system: implementation and motion planning,” *Autonomous Robots*, **10**-1, 23–40, 2001.

7) S. Murata, et al. : “Hardware Design of Modular Robotic System,” *Proc. 2000 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, F-AIII-3-5, 2000.

8) E. Yoshida, et al. : “A distributed method for reconfiguration of 3-D homogeneous structure,” *Advanced Robotics*, **13**-4, 363–380, 1999.

9) K. Tomita, et al. : “Self-assembly and self-repair method for distributed mechanical system,” *IEEE Trans. on Robotics and Automation*, **15**-6, 1035–1045, 1999.

10) K. Kotay and D. Rus: “Motion synthesis for the self-reconfigurable molecule,” *Proc. 1998 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 843–851, 1998.