Robotics: Science and Systems 2020
Corvalis, Oregon, USA, July 12-16, 2020

Motion Planning for Variable Topology Truss Modular Robot

Chao Liu
GRASP Laboratory
University of Pennsylvania
Email: chaoliu@seas.upenn.edu

Sencheng Yu
GRASP Laboratory
University of Pennsylvania
Email: schyu@seas.upenn.edu

Mark Yim
GRASP Laboratory
University of Pennsylvania
Email: yim@seas.upenn.edu

***Abstract*—Self-reconfigurable modular robots are composed of many modules that can be rearranged into various structures with respect to different activities and tasks. The variable topology truss (VTT) is a class of modular truss robot. These robots are able to change their shape by not only controlling joint positions which is similar to robots with fixed morphologies, but also reconfiguring the connections among modules in order to change their morphologies. Motion planning for VTT robots is difficult due to their non-fixed morphologies, high-dimensionality, potential for self-collision, and complex motion constraints. In this paper, a new motion planning algorithm to dramatically alter the structure of a VTT is presented, as well as some comparative tests to show its effectiveness.**

I. Introduction

Self-reconfigurable modular robots consist of repeated building blocks (modules) from a small set of types with uniform docking interfaces that allow the transfer of mechanical forces and moments throughout all modules [23]. These systems are capable of reconfiguring themselves in order to handle failures and adapt to different tasks.

Many self-reconfigurable modular robots have been developed, with the majority being lattice or chain type systems. In lattice modular robots, such as Telecubes [20] and Miche [5], modules are regularly positioned on a three-dimensional grid. Chain modular robots, such as PolyBot [22], M-Tran [16] and CKBot [24], consist of chains of modules. Their modules are connected in tree-like configurations and are more versatile as the serial chains of n modules can act like articulated robot arms. Some systems (SUPERBOT [17] and SMORES [11]) are hybrid and move like chain systems for articulated tasks but reconfigure using lattice-like actions.

Modular truss robots are different from lattice and chain type systems in that the systems are made up of beams that typically form parallel structures. The variable geometry truss (VGT) [15] is a modular robotic truss system composed of prismatic joints as truss members (modules), and examples include TETROBOT [6], Odin [14], and Linear Actuator Robots [21]. These truss members alter their length to perform locomotion or shape-morphing tasks. Another similar hardware is Intelligent Precision Jigging Robots [9]. The **variable topology truss (VTT)** is similar to variable geometry truss robots with additional capability to self-reconfigure the connection between members to alter the truss topology [18] [8]. One of the current hardware prototypes is shown in Fig. 1.

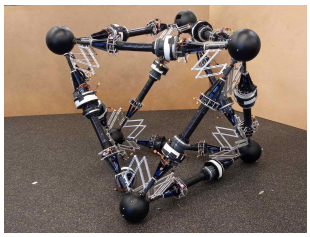


Fig. 1. The hardware prototype of a VTT in cubic configuration is composed of 12 members. Note that at least 18 members are required for topology reconfiguration [18].

A significant advantage for self-reconfigurable modular robots over other robots with fixed morphologies is their versatility, namely they are able to adapt themselves into different morphologies with respect to different requirements. For example, a VTT in which the members form a broad supported structure is well suited for shoring buildings or structures after disasters, while another truss with some members protruding to form an arm that has a large reachable workspace is good at manipulation tasks. However, a fundamental problem is performing collision-free (especially self-collision-free) motion planning for VTT systems.

A truss is composed of truss members (beams) and nodes (the connection points of multiple beams). A VTT is composed of *edge modules*. Each edge module has an active prismatic joint member and passive joint ends that can actively attach or detach from other edge module ends [18]. The *configuration* can be fully defined by the set of member lengths and their node assignments at which point the edge modules are joined. A node is constructed by multiple edge module ends using a linkage system with a passive rotational degree of freedom. The node assignments define the topology or how truss edge modules are connected, and the length of every member defines the shape of the resulting system [10].

Thus, there are two types of reconfiguration motion: *geometry reconfiguration* and *topology reconfiguration*. Geometry reconfiguration involves moving positions of nodes by changing length of corresponding members and topology reconfiguration involves changing the connectivity among members.

There are some physical constraints for a VTT to execute geometry reconfiguration and topology reconfiguration. A VTT has to be a rigid structure in order to maintain its shape

and be statically determinant. A node in a VTT must be of degree three, so has to be attached by at least three members to ensure its controllability. In addition, A VTT requires at least 18 members before topology reconfiguration is possible [18]. Thus, motion planning for VTT systems has to deal with at least 18 dimensions and typically more than 21 dimensions. These constraints complicate the motion planning problem.

As VTTs are inherently parallel robots, it is much easier to solve the inverse kinematics than the forward kinematics. So, for the geometry reconfiguration, rather than doing motion planning for the active degrees of freedom — the member lengths — we plan the motion of the *nodes* and then do the inverse kinematics to easily determine member lengths. However, motions of multiple nodes are strongly coupled, namely moving one node can significantly affect the configuration space of other nodes.

Hence, it is a challenge to do motion planning for multiple nodes at the same time. Motion planning for multiple nodes are also involved when executing topology reconfiguration. Some impossible motions can become possible with topology reconfiguration. For example, a single node controlled by enough edge modules can be split into a pair of nodes in order to go around some internal blocking members and then merge back to an individual node. This process requires the motion planning for two nodes at the same time.

In this work, we present a new framework for the motion planning of a VTT. This method dramatically reduces the search space when planning for multiple node motions greatly improving efficiency. In addition, a fast method to compute the configuration space which is often not fully connected can explicitly answer whether a topology reconfiguration action is required for a motion goal. An algorithm to compute a sequence of topology reconfiguration actions is then introduced, if required, for a motion goal. Software and videos are available on our lab website page.[1]

The rest of the paper is organized as follows. Section II reviews relevant and previous works and some necessary concepts are introduced in Section III as well as the motion planning problem statement. Section IV presents the geometry reconfiguration algorithm with motion of multiple nodes involved. Section V introduces the approach to verify whether a topology reconfiguration action is needed and the planning algorithm to do topology reconfiguration. The framework is demonstrated in Section VI. Finally, Section VII talks about the conclusion and some future work.

II. Related Work

In order to enable modular robotic systems to adapt themselves to different activities and tasks, many reconfiguration motion planning algorithms have been developed over several decades for a variety of modular robotic systems [3, 2, 7, 11]. The planning frameworks are for topology reconfiguration where undocking (disconnecting two attached modules) and docking (connecting two modules) actions are involved. There are also some approaches for shape morphing and manipulation tasks, including inverse kinematics for highly redundant chain using PolyBot [1], constrained optimization techniques with nonlinear constraints [4]. Here, there are no topology reconfiguration actions involved, but complicated kinematic structures and planning in high dimensional spaces needs to be considered. However, these methods are not applicable to variable topology truss systems which have a very different morphology and connection architecture. Indeed the physical constraints and collision models are significantly different from all of the previous lattice and chain type systems.

The variable topology truss hardware design and basic analysis were presented in [18]. Some approaches have been developed for VGT systems that are similar to VTT systems but without topology reconfiguration capability. Hamlin and Sanderson [6] presented a kinematic control but is limited to tetrahedrons or octahedrons. Usevitch et al. [21] introduced linear actuator robots (LARs) as well as a shape morphing algorithm. These systems are in mesh graph topology constructed by multiple convex hulls, and therefore self-collision can be avoided easily. However, this is not applicable to VTT systems because edge modules span the space in a very non-uniform manner. There has been some work on VTT motion planning. Retraction-based RRT algorithm was developed by Jeong et al. [8] in order to handle this high dimensional problem and narrow passage problems which is a well-known issue in sampling-based planning approaches; nevertheless this approach is not efficient because it samples the whole workspace for every node and collision checking needs to be done for every pair of members. Also sometime waypoints have to be assigned manually. Liu and Yim [10] presented a reconfiguration motion planning framework inspired by the DNA replication process — the topology of DNA can be changed by cutting and resealing strands as tanglements form. This work is based on a new method to discretize the workspace depending on the space density and an efficient way to check self-collision. Both topology reconfiguration actions and geometry reconfiguration actions are involved if needed. However, only a single node is involved in each step and the transition model is more complicated which make the algorithm limited in efficiency.

Liu et al. [13] presented a fast algorithm to compute the reachable configuration space of a given node in a VTT which is usually a non-convex space and this space can be then decomposed into multiple convex polyhedrons so that a simple graph search algorithm can be applied to plan a path for this node efficiently. However, multiple nodes are usually involved for shape morphing. In this paper, we first extend this approach to compute the obstacles for multiple nodes so that the search space can be decreased significantly. In addition, only collision among a small number of edge modules needs considered when moving multiple nodes at the same time. Hence RRT can be applied efficiently. The idea has been discussed briefly in [12]. For some motion tasks, topology reconfiguration is required. An updated algorithm

[1]Supplementary materials are available at https://www.modlabupenn.org/2020/06/03/motion-planning-for-variable-topology-truss-modular-robot/.

is developed to compute the whole not fully connected free space and required topology reconfiguration actions can then be generated which can achieve motions that are similar to the DNA replication process.

III. Preliminaries and Problem Statement

A VTT can be represented as an undirected graph $G = (V, E)$ where V is the set of vertices of G and E is the set of edges of G: each member can be regarded as an undirected labeled edge $e \in E$ of the graph and every intersection among members can be treated as a vertex $v \in V$ of the graph denoting a node. The Cartesian coordinates of a node $v \in V$ is its `Pos` property denoted as $v\,[\texttt{Pos}] = [v_x, v_y, v_z]^\intercal \in \mathbb{R}^3$. Let $q^v = v[\texttt{Pos}]$ and the *configuration space* of node v denoted as $\mathcal{C}^v$ is simply $\mathbb{R}^3$. In this way, the state of a member $e = (v_1, v_2) \in E$ where v_1 and v_2 are two vertices of edge e can be fully defined by q^{v_1} and q^{v_2}. The position of a given node $v \in V$ is controlled by changing the length of all attached members denoted as $E^v \subseteq E$.

The geometry reconfiguration motion planning of a VTT is achieved by planning the motion of the involved nodes then determining the required member length trajectories since it is easier to solve the inverse kinematics problem. Given a node $v \in V$ in $G = (V, E)$, the state of every member $e \in E^v$, denoted as $\mathcal{A}^v(q^v)$, can be altered by changing q^v. The *obstacle region* of this node which includes self-collision with other members as obstacles $\mathcal{C}^v_{obs} \subseteq \mathcal{C}^v = \mathbb{R}^3$ is defined as

$$\mathcal{C}^v_{obs} = \{q^v \in \mathbb{R}^3 | \mathcal{A}^v(q^v) \cap \mathcal{O}^v \neq \emptyset\} \tag{1}$$

in which $\mathcal{O}^v$ is the obstacle for E^v. This obstacle region is fully defined by the states of $\forall e \in E \setminus E^v$ [13] and composed of multiple polygons. For a simple VTT shown in Fig. 2a, the obstacle region $\mathcal{C}^{v_0}_{obs}$ is shown in Fig. 2b. The *free space* of node v is just the leftover configurations denoted as

$$\mathcal{C}^v_{free} = \mathbb{R}^3 \setminus \mathcal{C}^v_{obs} \tag{2}$$

However, $\mathcal{C}^v_{free}$ may not be fully connected and may be partitioned by $\mathcal{C}^v_{obs}$. Only the enclosed subspace containing q^v which is denoted as $\mathcal{C}^v_{free}(q^v)$ is free for node v to move. A fast algorithm to compute the boundary of this subspace is presented in [13]. For example, given the VTT in Fig. 2a, $\mathcal{C}^{v_0}_{free}$ — the free space of v_0 — is shown in Fig. 2b and the subspace $\mathcal{C}^{v_0}_{free}(q^{v_0})$ is shown in Fig. 3.

$\mathcal{C}^v_{free}$ is usually partitioned by $\mathcal{C}^v_{obs}$ into multiple enclosed subspaces, and it is impossible to move v from one enclosed subspace to another one without topology reconfiguration. The physical system constraints from [18] allow two atomic actions on nodes that enable topology reconfiguration: `Split` and `Merge`. Since the physical system must be statically determinate with all nodes of degree three, a node v must be composed of six or more edge modules to undock and split into two new nodes v' and v'', and both nodes should still have three or more members. This process is called `Split`. Two separate nodes are able to merge into an individual one in a `Merge` action. The simulation of these two actions is shown in Fig. 4. Hence, in topology reconfiguration process, the number of nodes can change, but the number of members which are physical elements remains constant.

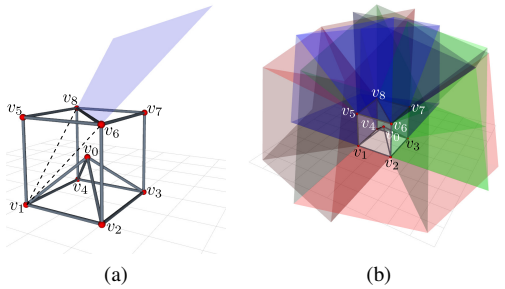


Fig. 2. (a) Given node v_0, one of its neighbors v_1 and a member (v_6, v_8) can define the blue polygon which is part of $\mathcal{C}^{v_0}_{obs}$. (b) The obstacle region $\mathcal{C}^{v_0}_{obs}$ is composed of polygons and the leftover space of $\mathbb{R}^3$ is $\mathcal{C}^v_{free}$.

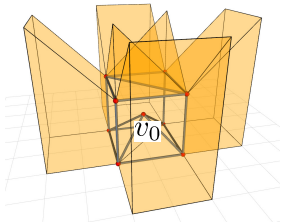


Fig. 3. $\mathcal{C}^{v_0}_{free}(q^{v_0})$ is bounded by polygons and workspace boundaries.

In this work, given a VTT $G = (V, E)$, the motion planning problem can be stated in the following:

1) Geometry Reconfiguration: For a set of n nodes $\{v_t \in V | t = 1, 2, \cdots, n\}$, compute paths $\tau_t : [0, 1] \rightarrow \mathcal{C}^{v_t}_{free}$ such that $\tau_t(0) = q_i^{v_t}$ and $\tau_t(1) = q_g^{v_t}$ in which $t = 1, 2 \cdots, n$, $q_i^{v_t}$ is the initial position of v_t and $q_g^{v_t}$ is the goal position of v_t.

2) Topology Reconfiguration: Compute topology reconfiguration actions, including `Merge` and `Split`, and collision-free path(s) to move a node v from its initial position q_i^v to its goal position q_g^v.

IV. Geometry Reconfiguration

The overall shape of a VTT is altered by moving nodes around in the workspace. For an individual node, its configuration space is $\mathbb{R}^3$. Apparently, the configuration space for n nodes is $\mathbb{R}^{3n}$. When multiple nodes are involved, the motion planning problem will be in high-dimensional space. Our strategy to avoid this high-dimensionality is to divide the moving

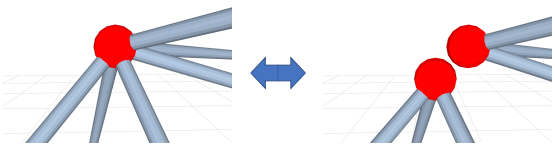


Fig. 4. A single node with six members can be split into a pair of nodes and two separate nodes can also merge into an individual node.

nodes into multiple groups and each group contains one or a pair of nodes. The motion planning space for each group is either in $\mathbb{R}^3$ or $\mathbb{R}^6$. Even with lower dimensional space, it is not efficient if searching the whole $\mathbb{R}^3$ or $\mathbb{R}^6$. Narrow passage is also a problem when applying rapidly-exploring random tree (RRT) algorithm. This issue is overcome by computing free space of the group in advance so that the sampling space is decreased significantly.

A. Obstacle Region and Free Space for a Group of Nodes

A group can contain either one node or a pair of nodes. Given a VTT $G = (V, E)$, for an individual node $v \in V$, an efficient algorithm to compute $\mathcal{C}^v_{obs}$ and $\mathcal{C}^v_{free}(q^v)$ with its boundary is introduced by Liu et al. [13]. If there are two nodes $v_i \in V$ and $v_j \in V$ in a group, then any collision among members in E^{v_i} and E^{v_j} is treated as self-collision, and all the members in $E \setminus (E^{v_i} \cup E^{v_j})$ define the obstacle region of this group denoted as $\widehat{\mathcal{C}}^{v_i}_{obs}$ (the obstacle region of v_i in the group) and $\widehat{\mathcal{C}}^{v_j}_{obs}$ (the obstacle region of v_j in the group) respectively, namely

$$\widehat{\mathcal{C}}^{v_i}_{obs} = \{q^{v_i} \in \mathbb{R}^3 | \mathcal{A}^{v_i}(q^{v_i}) \cap \mathcal{O}^{v_i,v_j} \neq \emptyset\}$$
$$\widehat{\mathcal{C}}^{v_j}_{obs} = \{q^{v_j} \in \mathbb{R}^3 | \mathcal{A}^{v_j}(q^{v_j}) \cap \mathcal{O}^{v_i,v_j} \neq \emptyset\}$$

in which $\mathcal{O}^{v_i,v_j}$ is formed by every $e \in E \setminus (E^{v_i} \cup E^{v_j})$.

Then the free space of v_i and v_j in the group can be derived as

$$\widehat{\mathcal{C}}^{v_i}_{free} = \mathbb{R}^3 \setminus \widehat{\mathcal{C}}^{v_i}_{obs} \tag{3}$$
$$\widehat{\mathcal{C}}^{v_j}_{free} = \mathbb{R}^3 \setminus \widehat{\mathcal{C}}^{v_j}_{obs} \tag{4}$$

Using the same boundary search approach in [13], the boundary of $\widehat{\mathcal{C}}^{v_i}_{free}(q^{v_i})$ — the enclosed subspace containing the current position of node v_i — can be obtained efficiently. Similarly, the boundary of $\widehat{\mathcal{C}}^{v_j}_{free}(q^{v_j})$ can be obtained. For example, given the VTT shown in Fig. 5, if node v_0 and v_1 form a group, then $\widehat{\mathcal{C}}^{v_0}_{free}(q^{v_0})$ and $\widehat{\mathcal{C}}^{v_1}_{free}(q^{v_1})$ can be computed and shown in Fig. 6. It is guaranteed that as long as v_0 is moving inside $\widehat{\mathcal{C}}^{v_0}_{free}(q^{v_0})$ (the space shown in Fig. 6a), there must be no collision between any member in E^{v_0} and any member in $E \setminus (E^{v_0} \cup E^{v_1})$. Similarly, no collision between any member in E^{v_1} and any member in $E \setminus (E^{v_0} \cup E^{v_1})$ can happen if v_1 is moving inside $\widehat{\mathcal{C}}^{v_1}_{free}(q^{v_1})$ (the space shown in Fig. 6b). In this way, when planning the motion of node v_0 and v_1 using RRT, the sample will only be generated inside $\widehat{\mathcal{C}}^{v_0}_{free}(q^{v_0})$ and $\widehat{\mathcal{C}}^{v_1}_{free}(q^{v_1})$, and we only need to consider self-collision in the group, namely the collision among members in $E^{v_0} \cup E^{v_1}$. There is a special case when these two nodes in the group are connected by a member. Both ends of the member are moving which is not considered by our obstacle model. This is an extra case when doing collision check.

B. Path Planning for a Group of Nodes

If there is only one node v in the group and the motion task is to move the node from its initial position q^v_i to its goal position q^v_g where $q^v_i \in \mathcal{C}^v_{free}(q^v_i)$ and $q^v_g \in \mathcal{C}^v_{free}(q^v_i)$, then it is straightforward to apply RRT approach in $\mathcal{C}^v_{free}(q^v_i)$ and

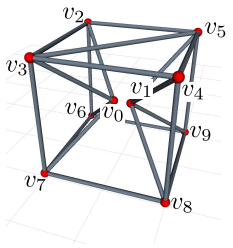


Fig. 5. A VTT is composed of 17 edge modules with 9 nodes among which v_0 and v_1 form a group.

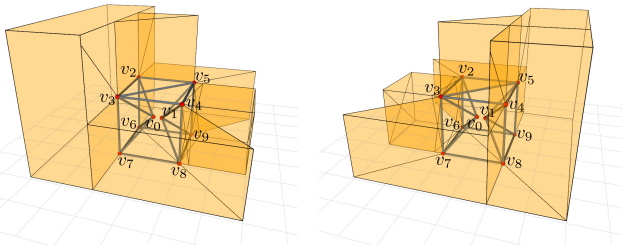


Fig. 6. (a) $\widehat{\mathcal{C}}^{v_0}_{free}(q^{v_0})$ is computed with all members controlling v_1 ignored. (b) $\widehat{\mathcal{C}}^{v_1}_{free}(q^{v_1})$ is computed with all members controlling v_0 ignored.

no collision can happen as long as the motion of each step is inside $\mathcal{C}^v_{free}(q^v_i)$ since this space is usually not convex.

When moving two nodes v_i and v_j in a group, sampling will only happen inside $\widehat{\mathcal{C}}^{v_i}_{free}(q^{v_i})$ and $\widehat{\mathcal{C}}^{v_j}_{free}(q^{v_j})$ for v_i and v_j respectively. If there is no edge module connecting v_i and v_j, then when applying RRT approach, the collision between moving members and fixed members can be ignored as long as the motion of both nodes in each step are inside $\widehat{\mathcal{C}}^{v_i}_{free}(q^{v_i})$ and $\widehat{\mathcal{C}}^{v_j}_{free}(q^{v_j})$ respectively. Only self-collision inside the group — the collision among members in $E^{v_i} \cup E^{v_j}$ — needs to be considered. If there is an edge module $e = (v_i, v_j)$ which connects v_i and v_j, since this case is not included in our obstacle model when computing the obstacle region, it is also necessary to check the collision between $e = (v_i, v_j)$ and every edge module in $E \setminus (E^{v_i} \cup E^{v_j})$.

In summary, when planning node v_i and v_j in a variable topology truss $G = (V, E)$, for each step, it is required to ensure the following

- The motion of both node v_i and v_j are inside $\widehat{\mathcal{C}}^{v_i}_{free}(q^{v_i})$ and $\widehat{\mathcal{C}}^{v_j}_{free}(q^{v_j})$ respectively;
- No collision happens among edge modules in $E^{v_i} \cup E^{v_j}$;
- No collision happens between edge module $e = (v_i, v_j)$ and every member in $E \setminus (E^{v_i} \cup E^{v_j})$ if $e = (v_i, v_j)$ exists.

It is difficult to check the second and third collision cases during the motion if both nodes are moving simultaneously. But since the step size for each node is limited and both of them are moving in straight lines, we can first check the collision during the motion of v_i while keeping v_j fixed, and

then check the motion of v_j. By doing so, the collision can be checked efficiently using the approach presented in [10]. Every edge module can be modeled as a line segment in space, thus, when moving node v, every $e \in E^v$ sweeps a triangle area, and if this member collides with another member $\bar{e} \in E$, then $\bar{e}$ must intersect with the triangle generated by e. Open Motion Planning Library (OMPL) [19] is used to implement RRT for this path planning problem.

C. Geometry Reconfiguration Planning

Assuming there are n nodes $\{v_t \in V | t = 1, 2 \cdots, n\}$ that should be moved from their initial positions $q_i^{v_1}, q_i^{v_2}, \cdots, q_i^{v_n}$ to their goal positions $q_g^{v_1}, q_g^{v_2}, \cdots, q_g^{v_n}$ respectively, we first divide these nodes into $\lceil n/2 \rceil$ groups. Each group contains at most two nodes. Then the motion task is achieved by moving nodes one group by one group. Then this geometry reconfiguration problem results in finding a sequence of groups that can achieve the task. If failed, try another grouping and find the corresponding sequence. Repeat this process until the task is finished, otherwise failed. With this approach, we can solve this geometry reconfiguration planning problem much faster than [13] and the detailed test results are in Section VI.

V. Topology Reconfiguration

Topology reconfiguration involves changing the connectivity among edge modules and there are two atomic actions: `Split` and `Merge`. The undocking and docking process is difficult for modular robotic systems, but sometimes necessary for some motion tasks. We need to verify whether the geometry reconfiguration process is enough or topology reconfiguration actions are needed. Recall that the free space of a node is usually not a single connected component, and if a motion task has initial and goal configurations in separated enclosed subspaces, topology reconfiguration actions are needed.

A. Enclosed Subspace in Free Space

If the free space $\mathcal{C}_{free}^v$ of a node v is separated by $\mathcal{C}_{obs}^v$ — the obstacle region of a node v, then each polygon in $\mathcal{C}_{obs}^v$ is connected to exactly two different enclosed subspaces. Hence after computing $\mathcal{C}_{free}^v$, we can compute all the enclosed subspaces by repeatedly applying the boundary search algorithm introduced in [13] until all polygons are included in exactly two enclosed subspaces. So we will first obtain the set of all obstacle polygons $\mathcal{P}_{obs}^v$. Then, as described in Algorithm 1, search for the enclosed subspace containing the current node configuration — $\mathcal{C}_{free}^v(q^v)$ — from a starting polygon P_s, which is the nearest one to the node [13]. Afterwards, we compute all other subspaces in $\mathcal{C}_{free}^v$.

In this algorithm, in the first loop, we build a counting map, C_P, to count how many enclosed subspaces a polygon has already been involved in. And in the second loop, we first check each polygon whether it is involved in two enclosed subspaces. If not, we will use this polygon as a starting polygon to apply the boundary search algorithm to obtain a new enclosed subspace and update the counts of all polygons involved in this subspace. The inner direction vector of the starting polygon [13] will be flipped to avoid obtaining the same subspace from different loops. Fig. 7 shows two enclosed subspaces of node v_0 in a simple cubic truss. In total, there are 33 enclosed subspaces in $\mathcal{C}_{free}^{v_0}$ above the ground.

Algorithm 1: Enclosed Subspace Search

Input: VTT $G = (V, E)$, node $v \in V$
Output: Set of all enclosed subspaces $\mathcal{C}_{free}^v$

Compute $\mathcal{P}_{obs}$;
$P_s \leftarrow$polygon closest to node v;
$\mathcal{C}_{free}^v(q^v) \leftarrow BoundarySearch\,(P_s)$;
$\mathcal{C}_{free}^v \leftarrow \left\{\mathcal{C}_{free}^v(q^v)\right\}$;
Define an empty counting map C_P;
for $P_i \in \mathcal{P}_{obs}^v$ **do**
  if P_i *is a boundary of* $\mathcal{C}_{free}^v(q^v)$ **then**
    $C_P\,[P_i] \leftarrow 1$;
  else
    $C_P\,[P_i] \leftarrow 0$;
for $P_i \in \mathcal{P}_{obs}^v$ **do**
  while $C_P\,[P_i] \neq 2$ **do**
    Flip inner direction vector of P_i;
    $\mathcal{C} \leftarrow BoundarySearch\,(P_i)$;
    $\mathcal{C}_{free}^v \leftarrow \mathcal{C}_{free}^v + \{\mathcal{C}\}$;
    for P_j *in boundaries of* $\mathcal{C}$ **do**
      $C_P\,[P_j] \leftarrow C_P\,[P_j] + 1$;
return $\mathcal{C}_{free}^v$

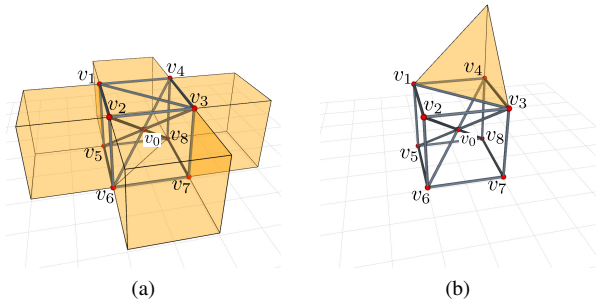


Fig. 7. (a) Enclosed subspace $\mathcal{C}_{free}^{v_0}(q^{v_0})$ contains the current position of v_0. (b) Another enclosed subspace is separated from $\mathcal{C}_{free}^{v_0}(q^{v_0})$ by obstacles.

B. Topology Reconfiguration Planning

Given a VTT $G = (V, E)$ and the motion task that is to move node v from q_i^v to q_g^v, if q_i^v and q_g^v belong to the same enclosed subspace, then geometry reconfiguration planning is able to handle this problem by either the approach in [13] or the approach introduced in Section IV-B. Otherwise, topology reconfiguration is needed.

There are multiple ways for a node v to split into nodes v' and v'' as there are multiple ways to take the members into two groups. However it is straightforward to compute all possible ways, denoted as action set A. Given q^v, the current

position of node v, and $\mathcal{C}^v_{free} = \{{}^t\mathcal{C}^v_{free} | t = 1, 2, \cdots, T\}$ that contains T enclosed subspaces in the free space of node v, $\forall a \in A$, two new nodes v' and v'' can be generated. $\mathcal{C}^{v'}_{free}(q^{v'})$ and $\mathcal{C}^{v''}_{free}(q^{v''})$ can be computed accordingly. If there is a position $q \in {}^t\mathcal{C}^v_{free}$ that is contained by both $\mathcal{C}^{v'}_{free}(q^{v'})$ and $\mathcal{C}^{v''}_{free}(q^{v''})$, namely both nodes can reach this position q, then ${}^t\mathcal{C}^v_{free}$ and $\mathcal{C}^v_{free}(q^v)$ can be connected under this action. With this transition model, a graph search algorithm can be applied to compute a sequence of topology reconfiguration actions to move this node v from q^v_i to q^v_g where q^v_i and q^v_g belong to different enclosed subspaces in $\mathcal{C}^v_{free}$. Here, the graph has enclosed subspaces as nodes. An edge in this graph connecting two enclosed subspaces denotes that node v can move from one enclosed subspace to the other. The graph is built from $\mathcal{C}^v_{free}(q^v)$, and grows as valid actions apply and stops when the enclosed subspace containing q^v_g is visited. A graph search algorithm designed based on Dijkstra's framework is shown in Algorithm 2.

Line 1 — 8: Compute all enclosed subspaces in $\mathcal{C}^v_{free}$ for node v using Algorithm 1 and, if q^v_i and q^v_g are in the same enclosed subspace, then no topology reconfiguration is needed. Otherwise, we uniformly and randomly select a location in every enclosed subspace except $\mathcal{C}^v_{free}(q^v_i)$ and $\mathcal{C}^v_{free}(q^v_g)$, and make two sets $\mathcal{Q}$ and $\overline{\mathcal{Q}}$ where $\mathcal{Q}$ contains all newly checked or non-visited enclosed subspaces and $\overline{\mathcal{Q}}$ contains all visited enclosed subspaces. The size of these two sets will change as the algorithm explores $\mathcal{C}^v_{free}$. Initially, only the enclosed subspace containing q^v_i that is $\mathcal{C}^v_{free}(q^v_i)$ and the enclosed subspace containing q^v_g that is $\mathcal{C}^v_{free}(q^v_g)$ are in $\mathcal{Q}$ and the algorithm starts with $\mathcal{C}^v_{free}(q^v_i)$. The value $g(\mathcal{C})$ is the cost of the path from q^g_i to the enclosed subspace $\mathcal{C}$, so $g(\mathcal{C}^v_{free}(q^v_i)) = 0$ and $g(\mathcal{C}^v_{free}(q^v_g)) = \infty$ in the beginning.

Line 10 — 12: Every iteration starts with the enclosed subspace that has the lowest cost $g(\mathcal{C})$ in $\mathcal{Q}$. At the beginning, $\mathcal{C}^v_{free}(q^v_i)$ has the lowest cost. After selecting an enclosed subspace, update the position of v to be ${}^{\overline{\mathcal{C}}}q$, and update $\mathcal{Q}$ and $\overline{\mathcal{Q}}$. The connection information among all enclosed subspaces is not known so we have to try all possible topology reconfiguration actions (all possible ways to split a node).

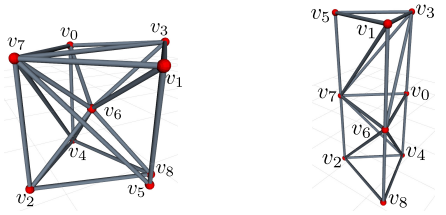
Line 13 — 26: After applying every valid topology reconfiguration action on node v which is currently in $\overline{\mathcal{C}}$, we can obtain a new VTT with two new generated nodes v' and v''. The enclosed subspace containing the current positions of these two new generated nodes can be computed using the boundary search algorithm in [13]. Then iterate every enclosed subspace $\mathcal{C}$ except $\overline{\mathcal{C}}$ in $\mathcal{C}^v_{free}$ and check if it is already visited. If so, then this potential connection is not a new connection. Otherwise, check if ${}^{\mathcal{C}}q$ is inside both $\mathcal{C}^{v'}_{free}(q^{v'})$ and $\mathcal{C}^{v''}_{free}(q^{v''})$. If this is true, then there are two cases: this subspace is not checked for the first time namely that there is already a connection between this enclosed subspace and another enclosed subspace, or this subspace has never been checked which means it has no connection before. For the first case, we need to check whether its cost needs to be updated. $c(a)$ is the cost of the current action a. In our case, a is always `Split` and different ways to split a node can have the same cost. If its cost is updated, then its parent $p(\mathcal{C})$ should also be updated accordingly. For the second case, initialize the cost and parent of this newly checked enclosed subspace, and update set $\mathcal{Q}$. Since ${}^{\overline{\mathcal{C}}}q$ is randomly selected, it is possible that the condition in *Line 18* is failed although there does exist a location in $\overline{\mathcal{C}}$ that can pass this condition.

Once $\mathcal{C}^v_{free}(q^v_g)$ is visited, the algorithm ends. With p, a tree with visited enclosed subspace as vertices, it is straightforward to find the path connecting $\mathcal{C}^v_{free}(q^v_i)$ and $\mathcal{C}^v_{free}(q^v_g)$ as well as the optimal topology reconfiguration action sequence. When moving from one enclosed subspace to the next enclosed subspace after splitting the node, apply the approach in Section IV-B to move them to any positions in the next enclosed subspace that are close enough to merge and merge them there.

Algorithm 2: Topology Reconfiguration Planning

Input: VTT $G = (V, E)$, initial q^v_i, goal q^v_g
Output: Tree of enclosed subspaces for node v

1 $\mathcal{C}^v_{free} \leftarrow EnclosedSubspaceSearch(G, v)$;
2 **if** $\mathcal{C}^v_{free}(q^v_i) = \mathcal{C}^v_{free}(q^v_g)$ **then**
3 **return** `Null`;
4 $\forall \mathcal{C} \in \mathcal{C}^v_{free} \setminus \left\{\mathcal{C}^v_{free}(q^v_i), \mathcal{C}^v_{free}(q^v_g)\right\}$, uniformly and randomly select a position ${}^{\mathcal{C}}q \in \mathcal{C}$;
5 ${}^{\mathcal{C}^v_{free}(q^v_i)}q \leftarrow q^v_i$, ${}^{\mathcal{C}^v_{free}(q^v_g)}q \leftarrow q^v_g$;
6 $\mathcal{Q} \leftarrow \left\{\mathcal{C}^v_{free}(q^v_i), \mathcal{C}^v_{free}(q^v_g)\right\}$, $\overline{\mathcal{Q}} \leftarrow \emptyset$;
7 $g(\mathcal{C}^v_{free}(q^v_i)) \leftarrow 0$, $g(\mathcal{C}^v_{free}(q^v_g)) \leftarrow \infty$;
8 Compute action set A;
9 **while** $\mathcal{C}^v_{free}(q^v_g) \in \mathcal{Q}$ **do**
10 $\overline{\mathcal{C}} \leftarrow \arg\min_{\mathcal{C} \in \mathcal{Q}} g(\mathcal{C})$, $q^v \leftarrow {}^{\overline{\mathcal{C}}}q$;
11 $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{\overline{\mathcal{C}}\}$;
12 $\overline{\mathcal{Q}} \leftarrow \overline{\mathcal{Q}} + \{\overline{\mathcal{C}}\}$;
13 **for** $a \in A$ **do**
14 Apply a on v to generate v' and v'';
15 Compute $\mathcal{C}^{v'}_{free}(q^{v'})$ and $\mathcal{C}^{v''}_{free}(q^{v''})$;
16 **for** $\mathcal{C} \in \mathcal{C}^v_{free} \setminus \{\overline{\mathcal{C}}\}$ **do**
17 **if** $\mathcal{C} \notin \overline{\mathcal{Q}}$ **then**
18 **if** ${}^{\mathcal{C}}q \in \mathcal{C}^{v'}_{free}(q^{v'}) \bigwedge {}^{\mathcal{C}}q \in \mathcal{C}^{v''}_{free}(q^{v''})$ **then**
19 **if** $\mathcal{C} \in \mathcal{Q}$ **then**
20 **if** $g(\overline{\mathcal{C}}) + c(a) < g(\mathcal{C})$ **then**
21 $g(\mathcal{C}) \leftarrow g(\overline{\mathcal{C}}) + c(a)$;
22 $p(\mathcal{C}) \leftarrow \overline{\mathcal{C}}$;
23 **else**
24 $\mathcal{Q} \leftarrow \mathcal{Q} + \{\mathcal{C}\}$;
25 $g(\mathcal{C}) \leftarrow g(\overline{\mathcal{C}}) + c(a)$;
26 $p(\mathcal{C}) \leftarrow \overline{\mathcal{C}}$;
27 **return** p



(a) (b)

Fig. 8. The motion task is to change the shape of a VTT from (a) a cubic truss for rolling locomotion to (b) a tower truss for shoring.

VI. Test Scenarios

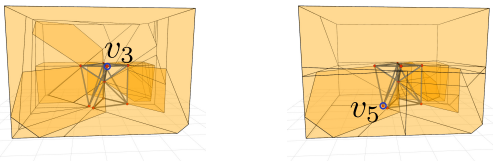
The motion planning framework is implemented in C++. Three example scenarios were conducted to measure the effectiveness of our approach. The performance of the framework is compared with the approach in [8], [13] and [10]. The first one tests geometry reconfiguration and the other two topology reconfiguration. All tests run on an Intel i7 processor at 2.20 GHz and the workspace is a cuboid above the ground.

A. Geometry Reconfiguration

The geometry reconfiguration planning test changes the cube shape of a VTT, Fig. 8a, to a tower shape, Fig. 8b. This VTT is composed of 21 members with initial positions of nodes listed in the following

$$q^{v_0} = [-1.60, -0.77, 2.08]^\intercal \quad q^{v_1} = [0.78, -0.76, 2.08]^\intercal$$
$$q^{v_2} = [-0.48, -2.02, 0.08]^\intercal \quad q^{v_3} = [-0.41, 0.42, 2.18]^\intercal$$
$$q^{v_4} = [-1.61, -0.77, 0.08]^\intercal \quad q^{v_5} = [0.38, -0.37, 0.13]^\intercal$$
$$q^{v_6} = [-0.43, -0.96, 1.23]^\intercal \quad q^{v_7} = [-0.48, -2.02, 2.08]^\intercal$$
$$q^{v_8} = [0.18, -0.17, 0.08]^\intercal$$

Four nodes v_1, v_3, v_5 and v_6 are involved in this motion task and their goal positions in the tower VTT are $q_g^{v_1} = [0.18, -0.17, 4.13]^\intercal$, $q_g^{v_3} = [-1.61, -0.77, 4.08]^\intercal$, $q_g^{v_5} = [-0.48, -2.02, 4.08]^\intercal$ and $q_g^{v_6} = [0.18, -0.17, 2.13]^\intercal$. These four nodes are separated into two groups $\{v_3, v_5\}$ and $\{v_1, v_6\}$ which is randomly selected. We first compute $\widehat{\mathcal{C}}_{free}^{v_3}(q_i^{v_3})$ (Fig. 9a) and $\widehat{\mathcal{C}}_{free}^{v_5}(q_i^{v_5})$ (Fig. 9b), and then do planning for these two nodes. The motion of node v_3 and v_5 is shown in Fig. 10. Most of the obstacle region in this step is surrounded by the subspace $\widehat{\mathcal{C}}_{free}^{v_3}(q_i^{v_3})$ and $\widehat{\mathcal{C}}_{free}^{v_5}(q_i^{v_5})$, hence it is easier for them to extend outwards first in order to navigate to the goal positions. After planning for v_3 and v_5, the truss is updated, and $\widehat{\mathcal{C}}_{free}^{v_1}(q_i^{v_1})$ and $\widehat{\mathcal{C}}_{free}^{v_6}(q_i^{v_6})$ are computed accordingly. Finally the planning for v_1 and v_6 finishes this motion task with the result shown in Fig. 11. For v_1 and v_6 in this updated truss, the enclosed subspace $\widehat{\mathcal{C}}_{free}^{v_1}(q_i^{v_1})$ and $\widehat{\mathcal{C}}_{free}^{v_6}(q_i^{v_6})$ almost covers the whole workspace so it is also easy to navigate to the goal positions. This motion task is also demonstrated in [8] and [13]. With the retraction-based RRT algorithm in [8], a waypoint needs to be added manually to mitigate the narrow passage problem. Using the approach in [13], it takes 4.004 s to finish the planning. However, we can solve this motion planning problem much faster on the same hardware. We did 1000 trails and the mean running time is 0.860 s with standard deviation of 0.669 s and the success rate is 100%.



(a) (b)

Fig. 9. (a) The enclosed subspace of v_3 in group $\{v_3, v_5\}$ containing $q_i^{v_3}$. (b) The enclosed subspace of v_5 in group $\{v_3, v_5\}$ containing $q_i^{v_5}$ in the initial VTT.

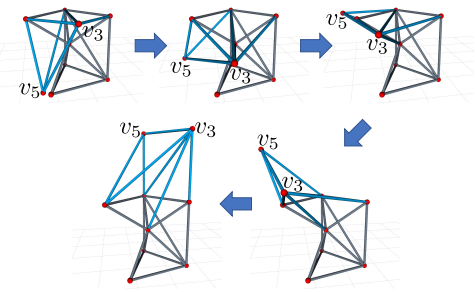


Fig. 10. v_3 and v_5 firstly extend outwards, and then move upward to their goal positions.

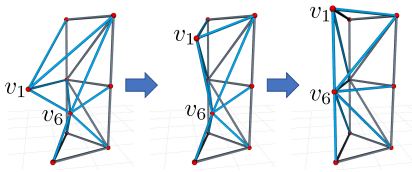


Fig. 11. v_1 and v_6 can navigate to their goal positions easily since $\widehat{\mathcal{C}}_{free}^{v_1}(q_i^{v_1})$ and $\widehat{\mathcal{C}}_{free}^{v_6}(q_i^{v_6})$ almost cover the whole workspace.

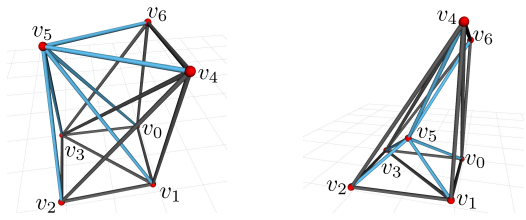
B. Topology Reconfiguration

1) Scenario 1: The VTT configuration used for this topology reconfiguration example is shown in Fig. 12a with the following nodes' positions

$$q^{v_0} = [0.05, 0, 0]^\intercal \quad q^{v_1} = [0.1, 1.8, 0]^\intercal$$
$$q^{v_2} = [2.1, 1.9, 0]^\intercal \quad q^{v_3} = [2.1, 0, 0]^\intercal$$
$$q^{v_4} = [0, 2.1, 3.1]^\intercal \quad q^{v_5} = [1.95, 0.9, 3]^\intercal$$
$$q^{v_6} = [0, 0, 2.9]^\intercal$$

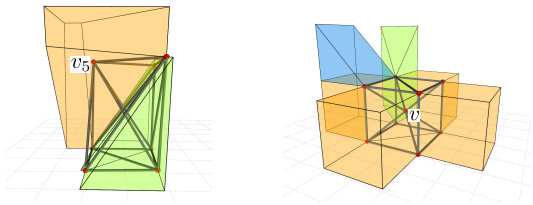
The motion task is to move v_5 from its initial position $q_i^{v_5} = [1.95, 0.9, 3]^\intercal$ to a goal position $q_g^{v_5} = [1, 1.2, 0.9]^\intercal$ (Fig. 12b). This motion cannot be executed with only geometry reconfiguration because $\mathcal{C}_{free}^{v_5}(q_i^{v_5})$ and $\mathcal{C}_{free}^{v_5}(q_g^{v_5})$ shown in Fig. 13a are separated by the obstacle region generated from edge module (v_3, v_4).

With our topology reconfiguration planning algorithm (Algorithm 2), one pair of `Split` and `Merge` actions is sufficient. v_5 is split into a pair of nodes v_5' and v_5'' so that both $\mathcal{C}_{free}^{v_5'}(q^{v_5'})$ and $\mathcal{C}_{free}^{v_5''}(q^{v_5''})$ contain $q_g^{v_5}$. Then the geometry motion planning is used to plan the motion of v_5' and v_5'' and



(a) (b)

Fig. 12. (a) A VTT is constructed from 18 edge modules with 6 nodes. (b) The goal is to move node v_5 from its initial position to a position inside the truss.



(a) (b)

Fig. 13. (a) $\mathcal{C}^{v_5}_{free}(q_i^{v_5})$ is the yellow space on the upper left and $\mathcal{C}^{v_5}_{free}(q_g^{v_5})$ is the green space on the lower right which are not connected and separated by the obstacle region generated from edge (v_3, v_4). (b) v has to move from $\mathcal{C}^{v}_{free}(q_i^{v})$ that is the yellow enclosed subspace to the green enclosed subspace, and then $\mathcal{C}^{v}_{free}(q_g^{v})$ that is the blue enclosed subspace.

control them to positions which are close enough to $q_g^{v_5}$, then merge them back to an individual node at $q_g^{v_5}$. The detailed process is shown in Fig. 14.

This motion task has been solved in [10] with the graph search algorithm exploring 8146 VTT configurations with a more complex action model in order to find a valid sequence of motion actions. With the proposed framework, the free space of v_5 is partitioned into 53 enclosed subspaces and it takes on average only $0.390\,\mathrm{s}$ to solve this motion task with standard deviation of $0.019\,\mathrm{s}$ in 1000 trails, including the enclosed subspace computation, topology reconfiguration planning and two-node geometry reconfiguration planning, and the success rate is 100%.

2) Scenario 2: Another motion task in which topology reconfiguration actions are involved is shown in Fig. 15 that is to move v from a position q_i^v inside the cubic truss (Fig. 15a) to a new position q_g^v (Fig. 15b). For this motion task, topology reconfiguration actions have to be applied twice to traverse three enclosed subspaces in $\mathcal{C}^{v}_{free}$ shown in Fig. 13b.

The detailed planning result is shown in Fig. 16. We first split the node v into two and do geometry reconfiguration planning to control them to an intermediate enclosed subspace and merge them back. Then split the node in a different way in order to navigate these two nodes to $\mathcal{C}^{v}_{free}(q_g^v)$ and merge them at q_g^v. The average planning time is $2.343\,\mathrm{s}$ with standard deviation being $0.531\,\mathrm{s}$ in 1000 trails, and the success rate is 95.8%. The search space is larger and more actions have to be applied in order to find the sequence of topology reconfiguration actions which consumes more time.

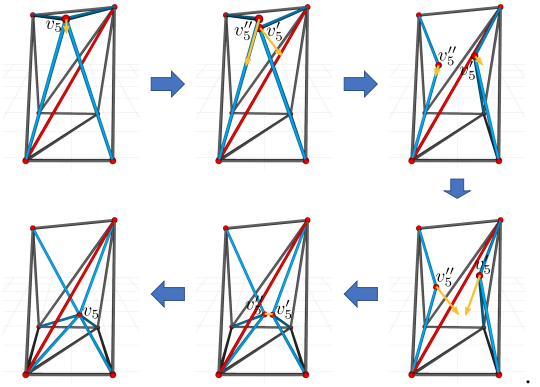
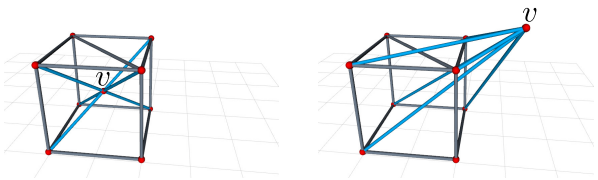


Fig. 14. The sequence to move v_5 from $q_i^{v_5}$ to $q_g^{v_5}$ and the motion is denoted as $\rightarrow$. First split v_5 into v_5' and v_5'', and then move these two newly generated nodes in different directions to go around the red edge module (v_3, v_4). Finally, merge them into an individual node at $q_g^{v_5}$.



(a) (b)

Fig. 15. (a) A VTT is constructed from 19 members with 9 nodes. (b) The task is to move v from its initial position to a position outside the cubic truss.

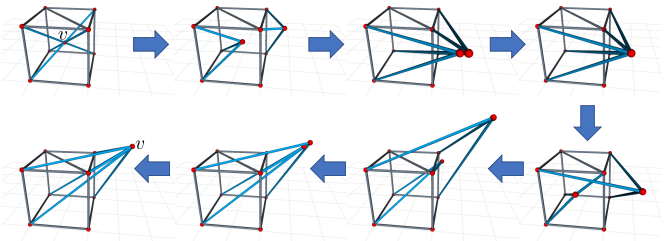


Fig. 16. The sequence to move v from q_i^v to q_g^v by traversing three enclosed subspaces in $\mathcal{C}^{v}_{free}$.

VII. Conclusion

A new efficient motion planning framework for variable topology trusses is presented that includes geometry reconfiguration and topology reconfiguration. The motion of multiple nodes are divided into the motion of multiple groups which contains either one or two nodes. Obstacle regions and free space for each group can be computed efficiently so that the geometry reconfiguration planning is fast using RRT technique. A fast algorithm to compute all enclose subspaces in the free space of a node is presented so that we can verify whether topology reconfiguration actions are needed. With our topology reconfiguration planning algorithm based on Dijkstra's algorithm, topology reconfiguration actions can be computed with geometry reconfiguration planning for a group of nodes, and the motion tasks requiring topology reconfiguration can then be solved efficiently.

References

[1] S. K. Agrawal, L. Kissner, and M. Yim. Joint solutions of many degrees-of-freedom systems using dextrous workspaces. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 3, pages 2480–2485 vol.3, May 2001. doi: 10.1109/ROBOT.2001.932995.

[2] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *The International Journal of Robotics Research*, 23(9):919–937, 2004. doi: 10.1177/0278364904044409.

[3] A. Casal and M. Yim. Self-reconfiguration planning for a class of modular robots. In Gerard T. McKee and Paul S. Schenker, editors, *Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, pages 246 – 257. International Society for Optics and Photonics, SPIE, 1999. doi: 10.1117/12.360345.

[4] M. Fromherz, T. Hogg, Y. Shang, and W. Jackson. Modular robot control and continuous constraint satisfaction. In *Proceedings of IJCAI Workshop on Modelling and Solving Problems with Contraints*, pages 47–56, Seatle, WA, 2001.

[5] Kyle Gilpin, Keith Kotay, Daniela Rus, and Iuliu Vasilescu. Miche: Modular shape formation by self-disassembly. *The International Journal of Robotics Research*, 27(3-4):345–372, 2008. doi: 10.1177/0278364907085557.

[6] G. J. Hamlin and A. C. Sanderson. TETROBOT: A modular approach to parallel robotics. *IEEE Robotics Automation Magazine*, 4(1):42–50, March 1997. ISSN 1558-223X. doi: 10.1109/100.580984.

[7] F. Hou and W.-M. Shen. Graph-based optimal reconfiguration planning for self-reconfigurable robots. *Robotics and Autonomous Systems*, 62(7):1047 – 1059, 2014. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2013.06.014.

[8] S. Jeong, B. Kim, S. Park, E. Park, A. Spinos, D. Carroll, T. Tsabedze, Y. Weng, T. Seo, M. Yim, F. C. Park, and J. Kim. Variable topology truss: Hardware overview, reconfiguration planning and locomotion. In *2018 15th International Conference on Ubiquitous Robots (UR)*, pages 610–615, June 2018. doi: 10.1109/URAI.2018.8441880.

[9] E. Komendera and N. Correll. Precise assembly of 3D truss structures using MLE-based error prediction and correction. *The International Journal of Robotics Research*, 34(13):1622–1644, 2015. doi: 10.1177/0278364915596588.

[10] C. Liu and M. Yim. Reconfiguration motion planning for variable topology truss. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1941–1948, Nov 2019. doi: 10.1109/IROS40897.2019.8967640.

[11] C. Liu, M. Whitzer, and M. Yim. A distributed reconfiguration planning algorithm for modular robots. *IEEE Robotics and Automation Letters*, 4(4):4231–4238, Oct 2019. ISSN 2377-3766. doi: 10.1109/LRA.2019.2930432.

[12] C. Liu, S. Yu, and M. Yim. Shape morphing for variable topology truss. In *2019 16th International Conference on Ubiquitous Robots (UR)*, Jeju, Korea, June 2019.

[13] C. Liu, S. Yu, and M. Yim. A fast configuration space algorithm for variable topology truss modular robots. In *2020 International Conference on Robotics and Automation (ICRA)*, Paris, France, May 2020.

[14] A. Lyder, R. F. M. Garcia, and K. Stoy. Mechanical design of odin, an extendable heterogeneous deformable modular robot. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 883–888, Sep. 2008. doi: 10.1109/IROS.2008.4650888.

[15] K. Miura. Design and operation of a deployable truss structure. In *NASA. Goddard Space Flight Center The 18th Aerospace Mech. Symp.*, pages 49–63, Greenbelt, Maryland, may 1984. doi: 19840017014.

[16] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441, Dec 2002. ISSN 1941-014X. doi: 10.1109/TMECH.2002.806220.

[17] B. Salemi, M. Moll, and W. Shen. SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3636–3641, Oct 2006. doi: 10.1109/IROS.2006.281719.

[18] A. Spinos, D. Carroll, T. Kientz, and M. Yim. Variable topology truss: Design and analysis. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2717–2722, Sep. 2017. doi: 10.1109/IROS.2017.8206098.

[19] I. A. Sucan, M. Moll, and L. E. Kavraki. The open motion planning library. *IEEE Robotics Automation Magazine*, 19(4):72–82, Dec 2012. ISSN 1558-223X. doi: 10.1109/MRA.2012.2205651.

[20] J. W. Suh, S. B. Homans, and M. Yim. Telecubes: Mechanical design of a module for self-reconfigurable robotics. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 4, pages 4095–4101 vol.4, May 2002. doi: 10.1109/ROBOT.2002.1014385.

[21] N. Usevitch, Z. Hammond, S. Follmer, and M. Schwager. Linear actuator robots: Differential kinematics, controllability, and algorithms for locomotion and shape morphing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5361–5367, Sep. 2017. doi: 10.1109/IROS.2017.8206431.

[22] M. Yim, D. G. Duff, and K. D. Roufas. PolyBot: A modular reconfigurable robot. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 514–520

vol.1, April 2000. doi: 10.1109/ROBOT.2000.844106.

[23] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics Automation Magazine*, 14(1): 43–52, March 2007. ISSN 1558-223X. doi: 10.1109/MRA.2007.339623.

[24] M. Yim, P. White, M. Park, and J. Sastra. Modular self-reconfigurable robots. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 5618–5631. Springer New York, New York, NY, 2009. ISBN 978-0-387-30440-3. doi: 10.1007/978-0-387-30440-3_334.