Journal Title
XX(X):1–23

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Locomotion and Self-reconfiguration Autonomy for Spherical Freeform Modular Robots

Yuxiao Tu[1,2], Guanqi Liang[1,2], Di Wu[1,2], Xinzhuo Li[1,2], and Tin Lun Lam[1,2]

Abstract

Modular robotic systems are multi-robot systems comprising numerous repeated modules and can transform into different configurations. Matching system configurations to a library enables efficient automation of modular robotic systems that have high degrees of freedom and strict motion constraints. Many previous approaches have automated cube-oriented modular robots by mapping the predefined sequence of gaits in the library to the module controllers. However, they can hardly drive robust three-dimensional self-reconfigurations without external sensors due to limited gait control accuracy and docking misalignment tolerance. Freeform modular robots are a type of modular robot with no fixed-point connectors, typically featuring continuous spherical joint connections between modules. They exhibit higher docking misalignment tolerance and better environmental adaptability. However, existing library-driven systems are inapplicable to freeform robots due to their redundant degrees of freedom and incompatible self-reconfiguration approaches. This article first proposes an autonomy framework for the locomotion and self-reconfiguration of spherical freeform modular robots. We model module connections as either spherical joints or parallel robots, employing a unified approach for skeletal kinematics. The system achieves the target configuration through iterative inverse kinematics and command translation to module controllers. A library with interfaces for configuration design is proposed, defining behaviors and feasible kinematic transitions between configurations. The executable behavior can be efficiently retrieved from the library by combining the proposed configuration matching and mapping algorithm. The system is validated on the FreeSN system with up to 18 modules containing 48 joint motors, providing a foundation for high-level planning and control research in freeform modular robots.

Keywords

Self-reconfigurable, cellular and modular robots, configuration recognition, skeletal kinematics, graph isomorphism.

1 Introduction

Modular self-reconfigurable robot (MSRR) systems (Liang et al. 2024; S. Sankhar Reddy Chennareddy 2017; Yim et al. 2007; Seo et al. 2019; Brunete et al. 2017; Hayat 2020; Bray and Groß 2023; Dokuyucu and Özmen 2023) typically consist of numerous repeated modules that feature uniform docking interfaces. These robot modules can connect and rearrange themselves into various configurations based on environmental conditions and task requirements. In the past decades, numerous modular robotics systems (Belke et al. 2023; Spröwitz et al. 2010; Romanishin et al. 2015; Davey et al. 2012; Tosun et al. 2016; Neubert et al. 2014; Swissler and Rubenstein 2020; Garcia et al. 2011; Gregg et al. 2024; Veenstra et al. 2025) with different types of connectors have been proposed, and they have demonstrated the rich adaptability and versatility in several aspects. However, most previous MSRRs were designed with a fixed number of fixed-point connectors, which requires accurate alignment between connectors during docking. The accumulated perception and control errors of chain configurations (Swissler and Rubenstein 2020) and the docking accuracy requirements make robust self-reconfiguration of MSRRs in three-dimensional (3D) difficult.

The freeform modular self-reconfigurable robot (Liang et al. 2024; S. Sankhar Reddy Chennareddy 2017) is a type of MSRR with no fixed-point connector. Different from traditional MSRRs with limited connection points and docking poses, a freeform MSRR module can connect to another module and move freely on its surface, so freeform MSRRs generally have better adaptability and high docking misalignment tolerance (Brunete et al. 2017; Eckenstein and Yim 2014). In our previous works, we proposed several freeform modular robots (Liang et al. 2020; Zhao et al. 2024; Tu et al. 2022; Liang et al. 2023) with magnetic connectors and demonstrated (Swissler and Rubenstein 2022; Luo and Lam 2022; Malley et al. 2020) that the freeform modular robots can better adapt to unstructured environments and reconfigure in 3D. The freeform connection between modules can be modeled as a spherical joint or rolling contact joint with no physical zero point, which enables flexible connection but also poses challenges in motion planning and control.

[1]School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Guangdong, China.
[2]Shenzhen Institute of Artificial Intelligence and Robotics for Society (AIRS), Shenzhen, Guangdong, China.

Corresponding author:
Tin Lun Lam, School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Guangdong, China.
Email: tllam@cuhk.edu.cn

In a modular robotic system, each robot module has at least one degree of freedom, and the system usually has more than ten modules or even more. The high degree of freedom makes the motion planning and control of modular robotic systems difficult. The number of possible configurations of the system also increases rapidly as the number of modules increases, and it is important to find valuable configurations among them efficiently. An approach to automate a modular robotic system is library-driven. Rather than trying to create configurations and perform motion planning from scratch, the user designs a library of valuable configurations and behaviors and the designs can be retrieved from the library based on system configuration and task requirements. The modular robotic system can leverage predefined gaits from the library to coordinate module controllers, enabling the execution of locomotion behaviors and complex tasks. Some systems (Jing et al. 2018; Daudelin et al. 2018) demonstrated highly automated behaviors with good performance based on the library for cube-oriented modular robots, but the self-reconfiguration behaviors rely on external sensors due to limited alignment tolerance.

Configuration matching and mapping is the process of matching a new modular robotic configuration to an existing one in the library and finding the module mapping between them. The configuration of most modular robot systems can be represented as matrices or graphs. Many configuration matching and mapping algorithms have been proposed for specific types of modular robotic systems based on spectral decomposition (Park et al. 2008), linear algebra (Shiu et al. 2010), and heuristic graph search (Park et al. 2008; Zhu et al. 2012; Liu and Yim 2020). However, the previous algorithms are mainly designed for modular robots with fixed-point connectors, and they do not efficiently consider the optimality among the feasible isomorphisms during mapping.

This article introduces a novel locomotion and self-reconfiguration framework for spherical freeform modular robots. The framework is validated on the FreeSN (Tu et al. 2022; Tu and Lam 2023) robotic system, which autonomously performs locomotion and self-reconfiguration without external sensors. The system configuration of spherical freeform modular robots is generally redundant and can be simplified as skeletal configuration by considering the symmetry of module shape, which can be represented as a directed graph with module ID and connection position vectors as edge attributes. The module spherical joint of different freeform modular robots or a group of modules can be modeled as a kinematic node. The kinematic nodes of a robotic system can form a kinematic tree. The forward kinematics and inverse kinematics of the skeletal configuration are derived based on the kinematic tree. Based on the skeletal kinematics, we present a locomotion and self-reconfiguration control strategy by moving to a set of predefined configurations in order. Then, an isomorphism tree searching algorithm is proposed to efficiently search the best module mapping between two configurations with the estimated sub-configuration distance as heuristic. Given a set of predefined skeletal configurations and a new skeletal configuration, the configuration that is isomorphic and most similar to the new configuration, along with a sub-optimal module mapping between them, can be efficiently searched. A behavior can be a configuration sequence or mapping between system and module motion parameters to produce specific locomotion or self-reconfiguration. We design a tool to design and record kinematic trees, configurations, and behaviors. A library can be automatically generated to store feasible kinematic transitions between the configurations and their labeled behaviors. For a newly constructed robotic system targeting reconfiguring to a configuration or executing a behavior, the configuration sequence with identical module mapping can be retrieved from the library by combining configuration matching, configuration mapping, and shortest path searching, which can be executed with the locomotion and self-reconfiguration control strategy. We demonstrate autonomous locomotion and self-reconfiguration experiments in 3D with at most 18 modules containing 48 motors for joint motion control. The main contributions of the article are the following:

- A skeletal kinematics modeling for spherical freeform modular robots.
- A configuration mapping and matching algorithm for freeform modular robots that considers the optimality among possible isomorphisms.
- A locomotion and self-reconfiguration framework that comprises library design, behavior retrieval, and behavior execution.

The rest of this article is organized as follows. Section 2 reviews and summarizes the relevant work and our previous work. Section 3 proposes a framework to model the skeletal kinematics of freeform modular robots and introduces an implementation of the module modeling. Section 4 presents a locomotion and self-reconfiguration control strategy with configurations of identical module mapping as input, and the detailed implementation of FreeSN. Section 5 proposes our configuration matching and mapping algorithm considering the optimal mapping among feasible isomorphisms. Section 6 introduces the overall autonomy framework. Section 7 demonstrates the proposed system with a real robotic platform. Finally, Section 8 concludes this article.

2 Related Work

2.1 Freeform Modular Robots

Freeform modular robots are a type of modular robots focusing on connection capabilities, enabling alignment-free connections among modules, significantly enhancing docking misalignment tolerance and adaptability (Liang et al. 2024). The s-bot (Gross et al. 2006) and Slimebot (Shimizu et al. 2006) stand out as the earliest freeform modular robots. An s-bot module contains a gripper and a ring, enabling the gripper to connect to any position on the ring of another module. Slimebot utilizes circular Velcro connectors and supports module connection at any planar position and orientation. Subsequently, several freeform modular robots were developed (Campbell et al. 2005; Kirby et al. 2007; Malley et al. 2020; Saintyves et al. 2024), incorporating diverse docking mechanisms and enhanced connector performance. Recently, a series of freeform modular robots (Swissler and Rubenstein 2020; Liang et al. 2020; Zhao and Lam 2022; Tu et al. 2022; Swissler and

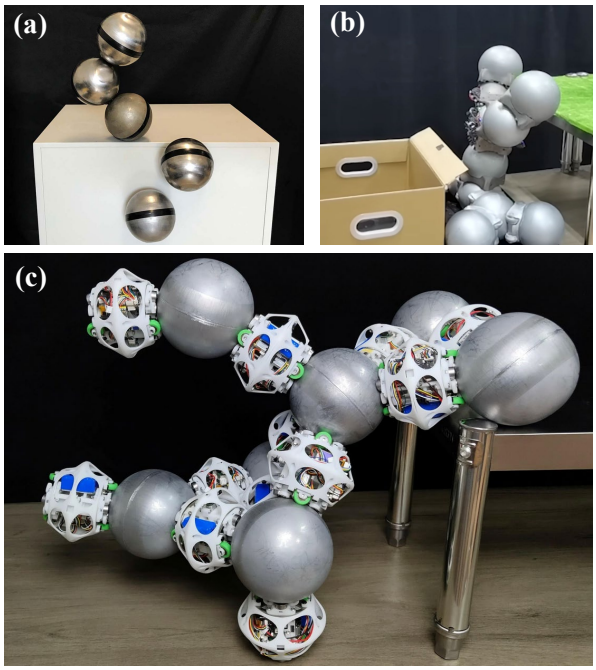

Figure 1. Freeform modular robots in our previous works. (a) FreeBOT. (b) Snailbot. (c) FreeSN.

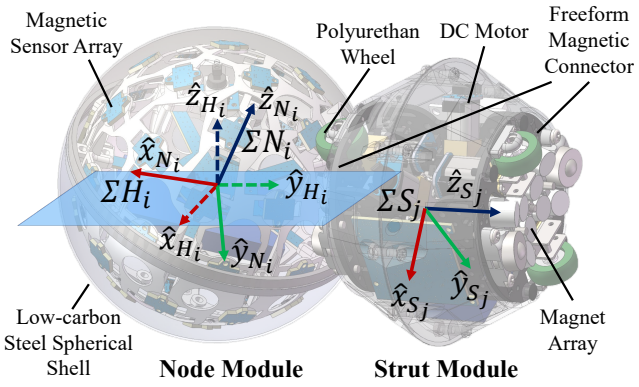

Figure 2. FreeSN hardware design and coordinate definition (Tu and Lam 2023).

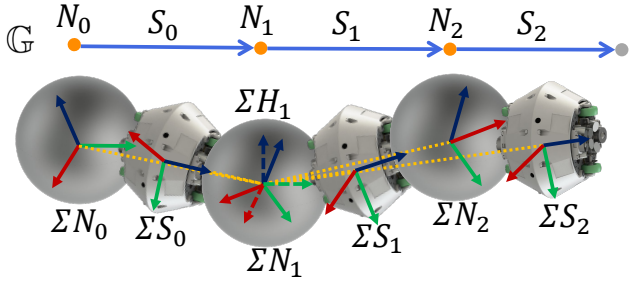

Figure 3. Example of FreeSN configuration representation. ΣH_1 is chosen as the root horizontal frame.

Rubenstein 2023) that can reconfigure in 3D emerged. The systems have demonstrated (Swissler and Rubenstein 2022; Luo and Lam 2022; Malley et al. 2020) that the freeform modular robots can better adapt to unstructured environments and perform robust self-reconfigurations.

In our previous works, we proposed several freeform MSRRs with magnetic connectors, which are introduced in the following sections.

FreeBOT FreeBOT (Liang et al. 2020) is a spherical modular robot that can connect to any point on the surface of other modules by magnetic connection. A FreeBOT module mainly contains a low-carbon steel spherical shell, internal magnet, and driving mechanism. The magnet inside FreeBOT can connect to the surface of other modules and produce rolling contact joint motion with an internal driving mechanism. Since a FreeBOT module contains only one connector, the topology connection of the FreeBOT system is a graph with at most one cycle. The point contact connection also weakens the torsional strength of the connector.

The joint modeling of spherical rolling contact (SRC) joint (Zong et al. 2022) is first proposed, where one body rolls without slipping over the surface of the other. The forward and inverse kinematics of the serial-chain SRC manipulator are well modeled and demonstrated using FreeBOT realization. The modeling can be extended to other freeform modular systems that use rolling contact joints and have tree-based connection topologies. However, it does not apply to systems with different joint types or non-tree connection topologies.

Snailbot To address the limitations of point connections of FreeBOT, we proposed a series of versions of the Snailbot (Zhao and Lam 2022; Zhao et al. 2024). The Snailbot can move on other modules with a large contact surface as a spherical joint and connect and disconnect freely using magnet-embedded rocker-bogie suspension or tracks. The Snailbot can also strengthen the connection strength with a vacuum sucker with directional polymer stalks. In this state, the module can only rotate around the sucker with one degree of freedom.

A connection planning method (Luo and Lam 2023) was proposed for Multiple In-degree Single Out-degree (MISO) modules such as FreeBOT and Snailbot, achieving computational efficiency through polynomial-time heuristics while preserving optimality via subsequent exponential-time searching. The algorithm focuses solely on the connection topology changes for MISO modules, while the poses of modules must be considered for actual robot self-reconfigurations.

FreeSN FreeSN is a freeform modular robot system (Tu et al. 2022) consisting of strut and node modules. As shown in Figure 2, the strut module contains two symmetric magnetic connectors. The surface of the node module is a low-carbon steel spherical shell. Each connector contains a magnet array and two polyurethane wheels driven by DC motors with wheel encoders. The magnet array can connect to any position on the node module, and the magnetic attraction force can be controlled by changing the position of the magnet array. Two wheels and motors form a differential driver so the connector can move on the surface of node module freely, enabling spherical joint motion. A strut module can connect with two node modules simultaneously, allowing for truss-like system topologies. We use FreeSN to illustrate the examples of the proposed system and conduct the system autonomy experiments in this article.

To enable the closed loop control of FreeSN system, we proposed a configuration identification system (Tu and Lam 2023). A magnetic sensor array is installed inside

the node module to locate the position of the connected magnetic connectors. We define a reference horizontal frame ΣH_i at each node module, where its z-axis $\hat{\boldsymbol{z}}_{H_i}$ is parallel with gravity, as shown in Figure 2. By fusing the magnetic localization result, inertial measurement unit, and wheel odometry, the modules can estimate their orientations relative to the horizontal frame of the adjacent node modules, such as ${}^{H_i}_{N_i}\hat{\boldsymbol{q}}$ and ${}^{H_i}_{S_j}\hat{\boldsymbol{q}}$. For a group of inter-connected modules, any node module can be selected as the root node module for reference. The configuration identification system can estimate the connection topology graph $\mathbb{G}$ and the real-time poses of modules relative to the horizontal frame ΣH_r of the root node module, denoted as ${}^{H_r}_{N_i}\hat{\boldsymbol{q}}, {}^{H_r}\boldsymbol{p}_{N_i}, {}^{H_r}_{S_j}\hat{\boldsymbol{q}}, {}^{H_r}\boldsymbol{p}_{S_j}$. An example of configuration identification results is shown in Figure 3, where the node module N_1 is selected as the root node module. The poses of modules relative to the root horizontal frame ΣH_1 can be estimated by the configuration identification system in real time.

The FreeSN modules and a centralized computer communicate through a Wi-Fi router. The module identification and local orientation filter are executed on the microcontrollers of modules distributively while the computer centrally estimates the system topology and configuration. We implemented low-level controllers on the strut module. Each strut module can control its connection position and orientation relative to the adjacent node modules, such as ${}^{H_i}_{S_j}\hat{\boldsymbol{q}}, {}^{H_i}\boldsymbol{p}_{S_j}, {}^{N_i}_{S_j}\hat{\boldsymbol{q}}, {}^{N_i}\boldsymbol{p}_{S_j}$, where modules N_i and S_j are adjacent. The centralized computer can send the control commands to the controllers of strut modules, and we demonstrated system behaviors (Tu and Lam 2023) by predefining a set of control commands and synchronizing the controllers. However, designing and synchronizing commands for these behaviors can be very complex for users. The success rate of these behaviors is also limited by the slippage between modules, which can lead to rapid offsets between node horizontal frames.

2.2 Configuration Matching and Mapping

Configuration matching and mapping is an algorithm that matches and maps a given new configuration with the existing configurations in a library and maps the modules with the matched one, which enables an important approach for planning and controlling modular robotic systems.

Chen and Burdick (Chen and Burdick 1993) used an assembly incidence matrix (AIM) to represent the module configuration and define the AIM equivalence relation based on geometric symmetry and graph isomorphism, which is applied to enumerate nonisomorphic configurations. Park et al. (Park et al. 2008) proposed a method called 3DDL, where the configuration is represented by a three-dimensional linked list of module objects (3DLL). The heuristics graph search approach is designed to find the mapping and achieve impressive gains in speed, especially for large configurations. However, different heuristics may need to be designed for different robots, and the 3DLL is specific to cube-oriented modules. Shiu et al. (Shiu et al. 2010) propose to recognize isomorphisms between two configurations by linear algebra. All isomorphisms can be found by operating the configuration matrix, but the complexity of the method is high. Liu and Yim (Liu and Yim 2017) propose a new approach to solve the matching and mapping problem simultaneously in polynomial time and demonstrate the system using SMORES. Zhu et al. (Zhu et al. 2012) combine the Matlab *graphisomorphism* function with a 3D linked list of modules, which uses the global pose information. However, the algorithm can only work for the system with discrete states of pose information and cannot efficiently solve the redundant isomorphisms.

These methods do not consider the optimality among the possible multiple isomorphisms or match the geometric information with all isomorphisms. Existing methods primarily demonstrate effectiveness on individual modular robots or specific types of modular robots, with a notable absence of configuration matching and mapping methods tailored for freeform modular robots.

3 Skeletal Kinematics

In this section, the general configuration representations of spherical freeform modular robots are first explained, including the definition of skeletal configuration. In order to control the skeletal configuration of robots with different kinematics, the definitions of kinematic node and kinematic tree are introduced, and the implementations of kinematic nodes for FreeSN in several modes are illustrated as examples. Finally, the forward and inverse kinematics of the skeletal configuration are introduced, forming the basis for its configuration matching and control.

3.1 Configuration Representation

The proposed freeform modular robots contain freeform connectors that can connect to other modules, and the connection produces spherical or rolling contact joint motion. We use a unified way to represent the configurations of the spherical freeform modular robots. Generally, the configuration of a group of inter-connected modular robots can be estimated by a system similar to the configuration identification system of FreeSN (Tu and Lam 2023), containing a directed connection topology graph $\mathbb{G}$ and the poses of modules relative to a reference frame. As shown in Figure 4, the topology graph of the example freeform modular robot systems are plotted. For FreeBOT and Snailbot, the nodes of the topology graph represent the modules and the edges represent the connection relationships. For FreeSN, the nodes of the topology graph represent the node modules and the edges represent that the strut modules connecting the nodes. If a module connector is not connected with other modules and the direction of the connector is essential, a virtual node is added to the graph. The orange circles and the blue arrows represent the nodes and edges of the topology graph $\mathbb{G}$, and the gray circle is the virtual node.

The above configuration representation contains redundant information about the module poses since the position and orientation of modules are related. Supposing that the connection direction of the module connector is always perpendicular to the spherical tangent plane of the connection point, the position of module is decided by the connection position vector $\hat{\boldsymbol{z}}_{M_j}$, and the orientation of module M_i can be represented with the connection position vector $\hat{\boldsymbol{z}}_{M_j}$ and the forward direction vector $\hat{\boldsymbol{x}}_{M_j}$. For FreeSN, the orientation of

Table 1. Nomenclature.

Notation	Description
$\mathbb{G}$	Topology connection graph
$\mathbb{M}$	Morphology configuration
$\mathbb{S}$	Skeletal configuration
$\mathbb{K}$	Kinematic Tree
$\mathbb{L}$	Behavior Library
S_j	Strut module of FreeSN
N_i	Node module of FreeSN
M_i	Freeform robot module
H_i	Horizontal frame of module
${}^{*}_{**}\boldsymbol{R}$	Rotation matrix of body ** relative to body *
${}^{*}_{**}\hat{\boldsymbol{q}}$	Unit quaternion of body ** relative to body *
${}^{*}\boldsymbol{p}_{**}$	Position of body ** relative to body *
${}^{*}\hat{\boldsymbol{z}}_{**}$	Connection direction vector of the connector of ** expressed in the frame *
${}^{*}\hat{\boldsymbol{x}}_{**}$	Forward direction vector of the connector of ** expressed in the frame *
${}^{H_r}Z^{*}$	Connection position vector matrix of configuration *
$\boldsymbol{f}_{node} : N^{*} \rightarrow N^{**}$	Node mapping from graph * to graph **
$\boldsymbol{f}_{edge} : E^{*} \rightarrow E^{**}, inv$	Edge mapping from graph * to graph **
$\boldsymbol{f}^{\mathbb{S}^{*}\rightarrow\mathbb{S}^{**}}_{node,edge}$	Module mapping from configuration * to configuration **
$\boldsymbol{f}^{I}_{node,edge}$	Identical module mapping between configurations
${}^{\mathbb{G}^{**}}\mathbb{S}^{*}$	Skeletal configuration $\mathbb{S}^{*}$ represented with configuration topology $\mathbb{G}^{**}$
$\boldsymbol{I}$	Identity matrix
$*^{\wedge}$	Hat operation that convert unit vector * to skew-symmetric matrix
$*(i)$	i-th element of vector $*$
$*(i,j)$	i-th row and j-th column of matrix $*$
$\circ$	Hadamard product of two vectors or matrixes

node module does not change the system properties and can be ignored since it is spherical. Generally, the configuration can be represented by the connection topology graph with $\hat{\boldsymbol{z}}_{M_j}$ and $\hat{\boldsymbol{x}}_{M_j}$ of the connections as edge attributes, denoted as morphology configuration $\mathbb{M}$. The blue arrows in Figure 4 also represent the vectors $\hat{\boldsymbol{z}}_{M_j}$ as edge attributes, and the red arrows are $\hat{\boldsymbol{x}}_{M_j}$.

In most locomotion and reconfiguration actions in morphology shaping applications, we aim to control only the shape of the system, and the configuration representation can be simplified based on the symmetry of the modules. We define this type of configuration representation as skeletal configuration, denoted as $\mathbb{S}$. For example, the FreeBOT module and the strut module of FreeSN are nearly axisymmetric around its z-axis, the self-rotation around it does not change the shape of the module. The skeletal configuration $\mathbb{S}$ can be represented by the topology graph $\mathbb{G}$ with $\hat{\boldsymbol{z}}_{M_j}$ as edge attributes.

3.2 General Kinematic Node and Kinematic Tree

The kinematics of the above modular robotic systems are different. We can model one or multiple interconnected modules as kinematic nodes for each type of robot and build a kinematic tree connecting the kinematic nodes to uniformly model the system kinematics. This section first proposes an abstraction of kinematic nodes, which summarizes the common properties of a general kinematic node.

The connection topology between the modules in a kinematic node k can be represented by a sub-topology graph $\mathbb{G}_k$, and the modules are arranged as an ordered set, denoted as $\mathbf{M}_k$. The connection position vectors and the forward direction vectors of the modules can be concatenated in order, denoted as $\boldsymbol{Z}_k$ and $\boldsymbol{X}_k$, where $\boldsymbol{Z}_k = \begin{bmatrix} ... & \hat{\boldsymbol{z}}_{M_j} & ... \end{bmatrix}, \boldsymbol{X}_k = \begin{bmatrix} ... & \hat{\boldsymbol{x}}_{M_j} & ... \end{bmatrix}, M_j \in \mathbf{M}_k$. Then, the following properties of the kinematic node should be defined.

- Joint space definition. The feasible motion of modules inside the kinematic node can be projected into a joint space with bounds. The modules should have no motion at the zero point of the joint space.
- Connection ports definition. The modules inside the kinematic node that can connect with other modules are defined as the connection ports of the kinematic node. The relative orientations between the ports change as the modules move.
- Kinematic node forward kinematics. The joint forward kinematics takes ordered morphology configuration vectors and joint space values as input, denoted as $\boldsymbol{Z}_k^{init}$, $\boldsymbol{X}_k^{init}$, and $\boldsymbol{\sigma}_k$. Moreover, it outputs the transformed vectors and the motion costs of modules, denoted as $\boldsymbol{Z}_k^{joint}$, $\boldsymbol{X}_k^{joint}$, and $\boldsymbol{C}_k$. The transformed vectors of the kinematic node also decide the control targets of the controller. During forward kinematics calculation, the relative orientation changes between the ports are cached.
- Controller implementation. Each kinematic node should have a corresponding controller implementation in the robot modules, where the controller can ideally execute the desired motion of the kinematic node.

The kinematic nodes can be classified into three types based on their properties:

- Fixed kinematic node. Any subgraph of the topology graph can be modeled as fixed modules, where the joint space dimension is zero, and the relative orientations between the ports are identity matrices.

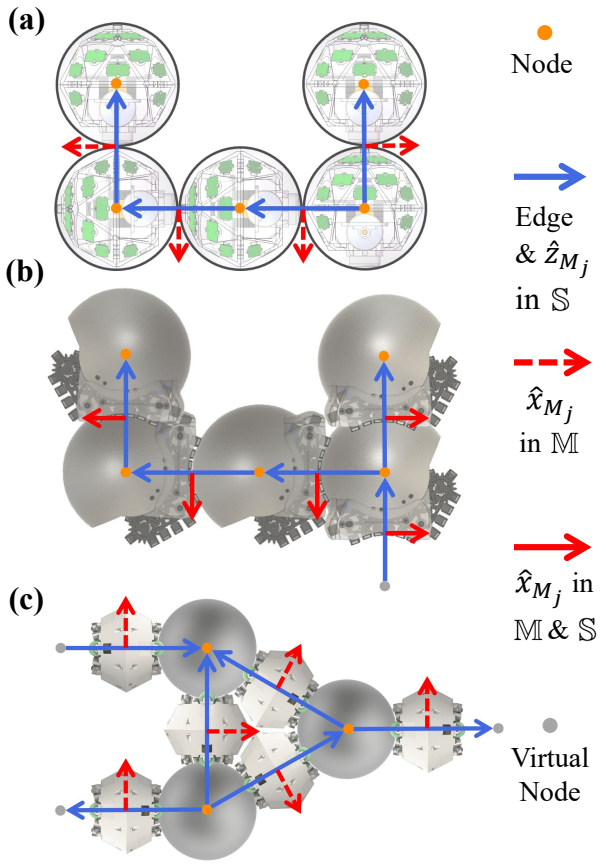

Figure 4. Configuration representations of spherical freeform modular robot systems. The orange circles represent the nodes of the topology graph, and the gray circle is the virtual node. The blue arrows represent the edges of the topology graph and the connection position vectors, while the red arrows represent the forward direction vectors. (a) Configuration representation of FreeBOT. (b) Configuration representation of Snailbot. (c) Configuration representation of FreeSN.

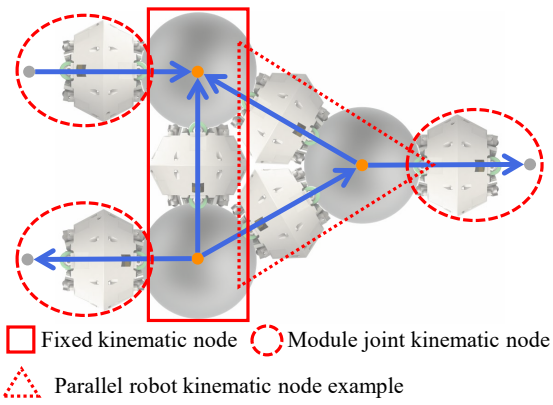

Figure 5. Three types of kinematic nodes and kinematic tree examples. The modules within a rectangle, triangle, and ellipse form a fixed kinematic node, a parallel robot kinematic node, and a module joint kinematic node, respectively.

- Module joint kinematic node. The module joint kinematic node models the connection of a single module. A module can work in different modes to better accomplish different types of tasks so that a robot module can have multiple kinematic node definitions. Different freeform robotic systems can share the same kinematic nodes.
- Parallel robot kinematic node. Parallel robot kinematic node models the subgraph of the topology graph containing cycles. For chain-type freeform robotics, the system topology graph has at most one loop, and the system can only include ring-type parallel robots. In FreeSN, the modules can form various types of parallel robots since the topology graph is diverse.

For a freeform robotics system with a tree-structure connection topology graph, the root node at the center of the topology graph can be modeled as a fixed kinematic node, and each pair of nodes and edges can be modeled as a module joint kinematic node. The kinematic nodes are then connected based on system topology with the port information as edge attributes, and we have the kinematic tree $\mathbb{K}$. The forward and inverse kinematics can be inferred based on the kinematic tree.

For the connection topology graph containing cycles, we assume that any cycle in the topology graph should be modeled inside a fixed kinematic node or parallel robot kinematic node, and we can model the system kinematics as a kinematic tree.

As shown in Figure 5, an example topology connection graph containing a triangular cycle is plotted. The cycle of the graph can be modeled as a fixed kinematic node and a parallel robot kinematic node. The other three subgraphs are modeled as module joint kinematic nodes.

Example Kinematic Node Modeling of FreeSN A strut module contains two differential driving connectors, and the connectors can produce spherical joint motion. If both sides of a strut module connect with node modules, a strut module can be modeled as two spherical joints in series. However, the two joints are parallel and have a much redundant degree of freedom. The strut can be simplified and modeled as different kinematic nodes by limiting the joint motion. The strut joint kinematic node modelings are summarized in Figure 6.

As shown in Figure 6(a), the module joint of FreeSN can work as a revolute joint. We define the joint space dimension as one, representing the expected rotation angle around the connection position vector $\hat{\boldsymbol{z}}_{M_j}^{init}$. Other properties of the kinematic node can be defined as follows:

$$\begin{aligned} \boldsymbol{R} &= rodrigues(\hat{\boldsymbol{k}} = \hat{\boldsymbol{z}}_{M_j}^{init}, \theta = \sigma_K) \\ \hat{\boldsymbol{z}}_{M_j}^{joint} &= \hat{\boldsymbol{z}}_{M_j}^{init} \\ \hat{\boldsymbol{x}}_{M_j}^{joint} &= R\, \hat{\boldsymbol{x}}_{M_j}^{init} \\ C &= |\sigma_K| \end{aligned} \tag{1}$$

where *rodrigues* outputs the rotation matrix that transforms $\hat{\boldsymbol{z}}_{M_j}^{init}$ to $\hat{\boldsymbol{z}}_{M_j}^{joint}$ based on the Rodrigues' Rotation Formula, as:

$$\begin{aligned} \hat{\boldsymbol{k}} &= \frac{\hat{\boldsymbol{v}} \times \hat{\boldsymbol{v}}'}{\|\hat{\boldsymbol{v}} \times \hat{\boldsymbol{v}}'\|}, \theta = arccos(\hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{v}}') \\ \boldsymbol{R} &= cos(\theta)\, \boldsymbol{I}_3 + (1 - cos(\theta))\hat{\boldsymbol{k}}\, \hat{\boldsymbol{k}}^T + sin(\theta)\, \hat{\boldsymbol{k}}^{\wedge} \end{aligned} \tag{2}$$

The joint produces differential driving spherical motion if only one differential driver is activated. As shown in

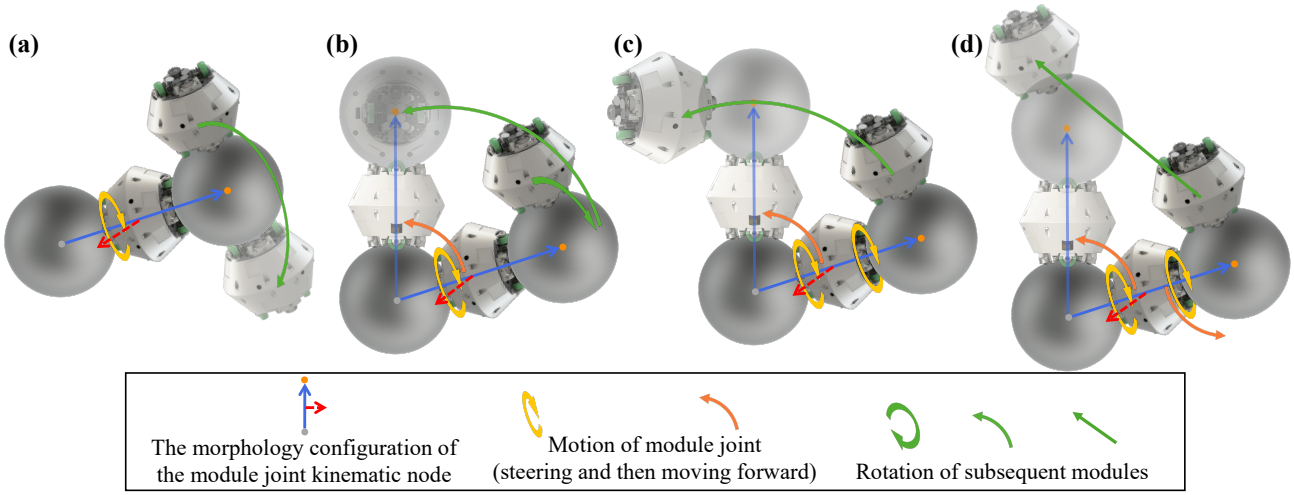

Figure 6. Module joint kinematic node examples of FreeSN. For each kinematic node, the motion of the module joint and the rotation of subsequent modules are labeled, and the poses of the modules after rotation are shown with transparency.

Figure 6(b), the module motion is modeled by first steering to the target position vector, then moving forward. Any joint motion can change the poses of subsequent kinematic nodes. We define the joint space dimension as three, and the joint space values are ideally the difference of the initial connection position vector $\boldsymbol{z}_{M_j}^{init}$ and the target one $\boldsymbol{z}_{M_j}^{joint}$. Then we have:

$$
\begin{aligned}
\hat{\boldsymbol{z}}_{M_j}^{joint} &= \frac{\boldsymbol{z}_{M_j}^{joint}}{\left\|\boldsymbol{z}_{M_j}^{joint}\right\|}, \boldsymbol{z}_{M_j}^{joint} = \hat{\boldsymbol{z}}_{M_j}^{init} + \boldsymbol{\sigma}_k \\
\hat{\boldsymbol{x}}_{M_j}^{steer} &= \frac{\boldsymbol{x}_{M_j}^{steer}}{\left\|\boldsymbol{x}_{M_j}^{steer}\right\|}, \boldsymbol{x}_{M_j}^{steer} = \boldsymbol{\sigma}_k - (\boldsymbol{\sigma}_k \cdot \hat{\boldsymbol{z}}_{M_j}^{init})\hat{\boldsymbol{z}}_{M_j}^{init} \\
C &= arccos(\hat{\boldsymbol{z}}_{M_j}^{init} \cdot \hat{\boldsymbol{z}}_{M_j}^{joint}) + \alpha\, arccos(\hat{\boldsymbol{x}}_{M_j}^{init} \cdot \hat{\boldsymbol{x}}_{M_j}^{steer}) \\
\boldsymbol{R} &= rodrigues\left(\hat{\boldsymbol{v}} = \hat{\boldsymbol{z}}_{M_j}^{init}, \hat{\boldsymbol{v}}^{'} = \hat{\boldsymbol{z}}_{M_j}^{joint}\right) rodrigues\,(\\
&\qquad \hat{\boldsymbol{v}} = \hat{\boldsymbol{x}}_{M_j}^{init}, \hat{\boldsymbol{v}}^{'} = \hat{\boldsymbol{x}}_{M_j}^{steer}\Big) \\
\hat{\boldsymbol{x}}_{M_j}^{joint} &= R\, \hat{\boldsymbol{x}}_{M_j}^{init}
\end{aligned}
\tag{3}
$$

The other differential driver can be activated to produce the opposite motion of the base one. If only the opposite steering motion is produced, we can assume that the steering motion does not change the poses of subsequent kinematic nodes, as shown in Figure 6(c). The forward kinematic property of the joint kinematic node becomes:

$$
\begin{aligned}
C &= arccos(\hat{\boldsymbol{z}}_{M_j}^{init} \cdot \hat{\boldsymbol{z}}_{M_j}^{joint}) + 2\alpha\, arccos(\hat{\boldsymbol{x}}_{M_j}^{init} \cdot \hat{\boldsymbol{x}}_{M_j}^{steer}) \\
\boldsymbol{R} &= rodrigues\left(\hat{\boldsymbol{v}} = \hat{\boldsymbol{z}}_{M_j}^{init}, \hat{\boldsymbol{v}}^{'} = \hat{\boldsymbol{z}}_{M_j}^{joint}\right)
\end{aligned}
\tag{4}
$$

Suppose all opposite motions shown in Figure 6(d) are produced. In that case, the motion cost is twice that of the pure differential spherical joint mode, and the transformation matrix of subsequent kinematic nodes ideally becomes the identity matrix.

Most simplified joint kinematic nodes can satisfy the control demand in applications where the control target is the skeletal configuration. We choose the kinematic node shown in Figure 6(c) as the default joint kinematic node of FreeSN in the rest of this article, which decouple the motion of modules with little additional cost. Other joint kinematic nodes can be selected if extra control demands exist.

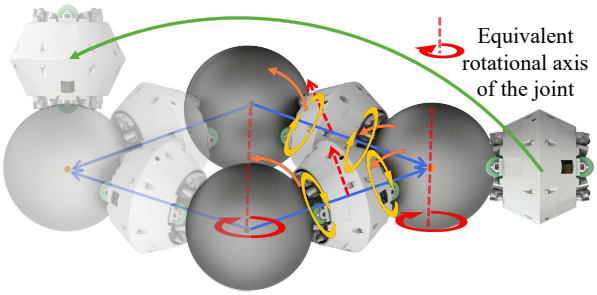

Figure 7. Parallel strut kinematic node example, where the parallel motions of two strut modules are equivalent to two parallel revolute joints.

Various parallel robot sub-structures can exist in the FreeSN system. A parallel strut kinematic node is introduced as example, as shown in Figure 7. The kinematic node includes two strut modules and one node module. The central two nodes belong to the parent of this kinematic node. The two connectors connected with the central two nodes can synchronously follow the circular trajectories on the nodes and produce revolute joint motion with superimposed torque. The other two connectors can also produce revolute joint motion similarly, while the two connectors of a strut module need to adjust the forward direction by producing parallel steering motion. So, the parallel strut kinematic node can be treated as two parallel revolute joints, and the joint space dimension is two, then we have:

$$\boldsymbol{Z}^{init} = \begin{bmatrix} \hat{\boldsymbol{z}}^{init}_{M_1} & \hat{\boldsymbol{z}}^{init}_{M_2} \end{bmatrix}$$
$$\hat{\boldsymbol{x}}^{joint} = \frac{\boldsymbol{x}^{joint}}{\|\boldsymbol{x}^{joint}\|}, \boldsymbol{x}^{joint} = \hat{\boldsymbol{z}}^{init}_{M_1} - \hat{\boldsymbol{z}}^{init}_{M_2}$$
$$\boldsymbol{R}_1 = rodrigues(\hat{\boldsymbol{k}} = \hat{\boldsymbol{x}}^{joint}, \theta = \boldsymbol{\sigma}_{k(1)})$$
$$\boldsymbol{Z}^{joint} = \boldsymbol{R}_1 \boldsymbol{Z}^{init}$$
$$\boldsymbol{R} = rodrigues(\hat{\boldsymbol{x}}^{joint}, \boldsymbol{\sigma}_{k(2)})\boldsymbol{R}_1$$
$$\hat{\boldsymbol{x}}^{steer} = \frac{\boldsymbol{x}^{steer}}{\|\boldsymbol{x}^{steer}\|}, \boldsymbol{x}^{steer} = \hat{\boldsymbol{z}}^{init}_{M_1} \times \hat{\boldsymbol{z}}^{init}_{M_2}$$
$$\boldsymbol{C} = \begin{bmatrix} 2\alpha\, arccos(\hat{\boldsymbol{x}}^{init}_{M_1} \cdot \hat{\boldsymbol{x}}^{steer}) + |\boldsymbol{\sigma}_{k(1)}| + \beta|\boldsymbol{\sigma}_{k(2)}| \\ 2\alpha\, arccos(\hat{\boldsymbol{x}}^{init}_{M_2} \cdot \hat{\boldsymbol{x}}^{steer}) + |\boldsymbol{\sigma}_{k(1)}| + \beta|\boldsymbol{\sigma}_{k(2)}| \end{bmatrix} \quad (5)$$

where β is the weight parameter.

3.3 Skeletal Forward Kinematics

The forward kinematic can be defined at any morphology configuration as the zero point of the joint space. The connection position vectors and the forward direction vectors can be formatted in breadth-first search (BFS) order of kinematic nodes as matrixes, denoted as $\boldsymbol{Z}^{init}$ and $\boldsymbol{X}^{init}$, where $\boldsymbol{Z} = \begin{bmatrix} ... & \boldsymbol{Z}_k & ... \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} ... & \boldsymbol{X}_k & ... \end{bmatrix}, k \in \mathbb{K}$. The joint space of the system is the union of the joint space of all kinematic nodes in the same order, and we denote the joint space vector as $\boldsymbol{\sigma} = \begin{bmatrix} ... & {\boldsymbol{\sigma}_k}^T & ... \end{bmatrix}^T, k \in \mathbb{K}$.

We have the skeletal forward kinematic algorithm shown in Alg. 1, which outputs the transformed configuration vectors $\boldsymbol{Z}^{rot}, \boldsymbol{X}^{rot}$ and the estimated motion costs $\boldsymbol{C}$.

BFS outputs the parent and child kinematic node pairs and the port connection via breadth-first search, except that an empty kinematic node is set as the parent of the root kinematic node for algorithm pseudocode simplification. *vecDim* outputs the dimension of the configuration vectors of the kinematic node, and *jointDim* outputs the dimension of the joint space of the kinematic node. *cacheRotMat* caches the rotation matrix from the root to the ports after joint motion, and $rotFromRoot$ outputs the rotation matrix from the root module of the root kinematic node to the input port of the input kinematic node. *forward* calls the forward kinematics of the kinematic node, which outputs the transformed configuration vectors and the motion costs with initial configuration vectors and joint space values. *ports* outputs the set of ports of the kinematic node, and *portRot* outputs the rotation matrix from one port to another based on the forward kinematics results.

3.4 Full Body Skeletal Inverse Kinematics

The skeletal forward kinematics estimate the system skeletal configuration with an initial morphology configuration and the joint space values. Assume that the estimated system configuration is represented under a horizontal reference frame of the root port of the root kinematic node, such as the root horizontal frame of FreeSN configuration identification system (Tu and Lam 2023), denoted as $H_{k_{root},p_{root}} = H_r$. And the configuration vectors are denoted as ${}^{H_r}\boldsymbol{Z}^{init}$ and ${}^{H_r}\boldsymbol{X}^{init}$. We have a target skeletal configuration $\mathbb{S}^{target}$ with the same topology graph, but the vectors are represented

Algorithm 1 Skeletal Forward kinematics

Input: $\mathbb{K}, \boldsymbol{Z}^{init}, \boldsymbol{X}^{init}, \boldsymbol{\sigma}$
Output: $\boldsymbol{Z}^{joint}, \boldsymbol{X}^{joint}, \boldsymbol{Z}^{rot}, \boldsymbol{X}^{rot}, \boldsymbol{C}$

$n_{vec} = 0, n_{joint} = 0, \boldsymbol{C} = [\ ]$
$\boldsymbol{Z}^{joint} = [\ ], \boldsymbol{X}^{joint} = [\ ]$
$\boldsymbol{Z}^{rot} = [\ ], \boldsymbol{X}^{rot} = [\ ]$
for $k_{parent}, k_{child}, p_{parent}, p_{child}$ in $BFS(\mathbb{K})$ **do**
  $\boldsymbol{Z}^{init}_{k_{child}} = (\boldsymbol{Z}^{init}_{ij})_{1\le i\le 3, n_{vec}\le j< n_{vec}+vecDim(k_{child})}$
  $\boldsymbol{X}^{init}_{k_{child}} = (\boldsymbol{X}^{init}_{ij})_{1\le i\le 3, n_{vec}\le j< n_{vec}+vecDim(k_{child})}$
  $\boldsymbol{\sigma}^{init}_{k_{child}} = (\boldsymbol{\sigma}_i)_{n_{joint}\le i< n_{joint}+jointDim(k_{child})}$
  $n_{vec} = n_{vec} + vecDim(k_{child})$
  $n_{joint} = n_{joint} + jointDim(k_{child})$
  ${}^{p_{root}}_{p_{parent}}\boldsymbol{R} = rotFromRoot(p_{parent})$
  $\boldsymbol{Z}^{joint}_{k_{child}}, \boldsymbol{X}^{joint}_{k_{child}}, \boldsymbol{C}_{k_{child}} = forward(k_{child}, Z^{init}_{k_{child}}, X^{init}_{k_{child}}, \sigma^{init}_{k_{child}})$
  for $p \in ports(k_{child})$ **do**
    ${}^{p_{root}}_{p}R = {}^{p_{root}}_{p_{parent}}R\, {}^{p_{parent}}_{p}R$
    ${}^{p_{parent}}_{p}R = portRot(p)$
    $cacheRotMat({}^{p_{root}}_{p}R)$
  end for
  $\boldsymbol{Z}^{joint} = \begin{bmatrix} \boldsymbol{Z}^{joint} & \boldsymbol{Z}^{joint}_{k_{child}} \end{bmatrix}$
  $\boldsymbol{X}^{joint} = \begin{bmatrix} \boldsymbol{X}^{joint} & \boldsymbol{X}^{joint}_{k_{child}} \end{bmatrix}$
  $\boldsymbol{Z}^{rot} = \begin{bmatrix} \boldsymbol{Z}^{rot} & {}^{p_{root}}_{p_{parent}}\boldsymbol{R}\, \boldsymbol{Z}^{joint}_{k_{child}} \end{bmatrix}$
  $\boldsymbol{X}^{rot} = \begin{bmatrix} \boldsymbol{X}^{rot} & {}^{p_{root}}_{p_{parent}}\boldsymbol{R}\, \boldsymbol{X}^{joint}_{k_{child}} \end{bmatrix}$
  $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{C}_{k_{child}} \end{bmatrix}$
end for

under some horizontal frame H_w. This section proposes the full-body skeletal inverse kinematics, assuming the ground is a horizontal plane. The two horizontal frames have a yaw angle difference θ. The proposed inverse kinematic algorithm estimates the yaw angle difference and the optimum joint space variables that align the system skeletal configuration with the target one with minimum weighted motion cost by solving the following non-linear optimizing problem.

$$\min_{\theta,\boldsymbol{\sigma}} \sum_{j\in M} \boldsymbol{w}(j) \left[\zeta \boldsymbol{C}(j)^2 + \gamma\, arccos\left(\sum_{k=1}^{3}\left({}^{H_r}\boldsymbol{X}^{rot} \circ (Rot_Z(\theta)\, {}^{H_w}\boldsymbol{X}^{target})\right)(i,j)\right)^2 + arccos\left(\sum_{i=1}^{3}({}^{H_r}\boldsymbol{Z}^{rot} \circ (Rot_Z(\theta)\, {}^{H_w}\boldsymbol{Z}^{target}))(i,j)\right)^2\right] \quad (6)$$

where $\boldsymbol{w}$, γ and ζ are weight parameters. $\boldsymbol{w}$ can be set based on the weight distribution of the system. If the forward direction vectors are not part of the skeletal configuration, γ can be simply set to zero. ${}^{H_r}Z^{rot}$ and ${}^{H_r}X^{rot}$ are the transformed connection position matrix and forward direction matrix, and C is a vector of motion costs. These are estimated by the skeletal forward kinematics as:

$${}^{H_r}\boldsymbol{Z}^{rot}, {}^{H_r}\boldsymbol{X}^{rot}, \boldsymbol{C} = SFK(\mathbb{K}, {}^{H_r}\boldsymbol{Z}^{init}, {}^{H_r}\boldsymbol{X}^{init}, \boldsymbol{\sigma}) \quad (7)$$

Ground Model For non-tree configurations with a floating base, the raw output of the forward kinematics algorithm may not have stable contact with the ground. The module inside the fixed kinematic node and parallel robot kinematic node may not have enough degree of freedom to align with the target one. A three-point contact model is applied to estimate the contact points between the ground and the modules. The matrixes are transformed again so that the modules are in contact with the ground at three points, and the center of mass lies in the triangle formed by the contact points.

Optimization Solving The proposed optimization problem can be solved by modern non-linear optimization algorithms, such as L-BFGS-B (Byrd et al. 1995) and Trust Region Reflective (TRF). Good initial variable selection of the optimization can significantly improve the optimization efficiency and prevent the local minimum. The output configuration of forward kinematics is the same as the input if the joint space variables are zeros. Zeros are suitable initial joint space variables if the initial configuration is not far from the target. The yaw angle difference can be estimated by solving the following linear optimization problem.

$$\min_{\theta} \sum_{j \in M} \boldsymbol{w}(j) \, [\, \gamma \, arccos \left(\sum_{k=1}^{3} ({}^{H_r}\boldsymbol{X}^{init} \circ ({}^{H_w}\boldsymbol{X}^{target} Rot_Z(\theta)))(j,k) \right)^2 + arccos \left(\sum_{k=1}^{3} ({}^{H_r}\boldsymbol{Z}^{init} \circ ({}^{H_w}\boldsymbol{Z}^{target} Rot_Z(\theta)))(j,k) \right)^2 \,] \tag{8}$$

This can efficiently estimate an initial θ that prevents local minimum when the initial configuration is not far from the target. However, the non-linear optimization algorithm is possibly trapped in the local minimum or even failed to find a feasible solution. A sampling-based approach similar to BioIK (Starke et al. 2019) is applied to avoid trapping into a local minimum, which samples on the joint space and solves the optimization problems concurrently.

Iterative Inverse Kinematics The proposed inverse kinematics algorithm cannot guarantee finding an optimal solution within a limited time for all configurations. In real-world applications where the joint motion consumes time, an optimal solution is not strictly required at each time step. The solution from the previous time step can be used as the initial variables for the current time step, which helps accelerate the algorithm. The iterative optimization can also help avoid local minima, provided that the local minimum guides the solution toward a correct motion direction.

During the initial solving phase, the algorithm continuously samples candidate solutions and optimizes them until a feasible one is found. Subsequently, a tracking thread solves the optimization problem at each time step using the previous solution as the initial variables. Meanwhile, an exploration thread optimizes candidate solutions sampled globally from the variable space. If the exploration thread finds a better solution, the tracking thread updates its initial variables accordingly.

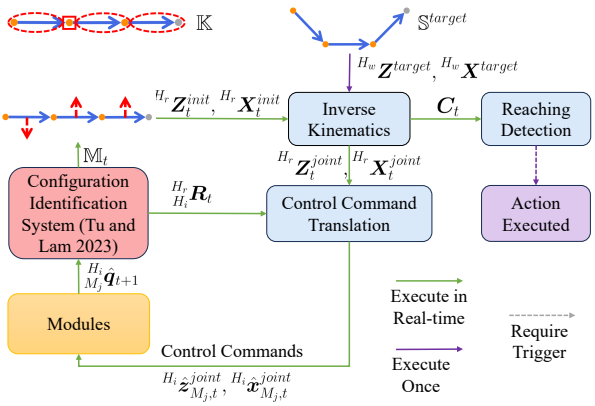

Figure 8. Moving action execution workflow. The green arrows represent that data is calculated and communicated in real-time. The purple arrows represent that the data is one-time, while the dashed line means that a conditional trigger is required.

4 Locomotion and Self-reconfiguration Execution

For a group of interconnected modules with an initial configuration $\mathbb{M}$ and a defined kinematic tree $\mathbb{K}$, we have a target skeletal configuration $\mathbb{S}^{target}$ with identical module mapping. The inverse kinematics algorithm can estimate the desired joint space variables to move to the target configuration.

We define the task to reach a target skeletal configuration as an action. A behavior can be a mapping between system motion parameters and module motions or the task of reaching a sequence of skeletal configurations one by one. In this section, an action execution framework is introduced.

4.1 Action Execution

The basic actions can be classified into four types based on the changing of configuration topology graph.

- Moving: The system moves to a target configuration $\mathbb{S}^{target}$ without topology changing.
- Connection: The system moves to $\mathbb{S}^{target}$ while a module connects to some module. A node and a virtual node of the topology graph are merged.
- Disconnection: A module disconnects with some module, and then the system moves to the target configuration. A node of the topology graph is split into a node and a virtual node.
- Flow: The leaf module of some MSRRs, such as FreeBOT and Snailbot, can move from its parent module to the neighbors of the parent module directly. The disconnection and connection happen simultaneously, and the control strategy depends on the hardware implementation.

Given the configuration of the current system and a target, the type of action can be detected by comparing the topology graph of the configurations. Then, the action can be executed with the corresponding execution strategy.

Moving Action Moving action is an essential action, where the modules are controlled in real-time until the system configuration is close enough to the target. Assume that a configuration identification system can estimate the system

morphology in real-time. The inverse kinematic algorithm can estimate target joint space variables, and the system can ideally move to the configuration by converting the corresponding joint vectors to the control commands of modules. However, the kinematic model may not be accurate due to the wheel slippage, and the reference coordinate frame of the control relative to the root frame changes as the modules move.

As shown in Figure 8, an example system forms the configuration $\mathbb{M}_t$ at initial, estimated by the configuration identification system in real-time. Given the kinematic tree $\mathbb{K}$ and a target skeletal configuration $\mathbb{S}^{target}$, the inverse kinematics algorithm calculates the joint space variables in real-time, which can be represented as joint vectors ${}^{H_r}Z^{joint}$ relative to the horizontal reference frame H_r. All kinematic nodes have corresponding controller implementation in the module, controlling the kinematic node to move to the target in the joint space. The control command of the kinematic nodes can be translated by combining the inverse kinematic results and module relative orientations, depending on the controller design. The above procedures are executed iteratively in real time to achieve robust configuration control. The controllers are terminated if the current system configuration is close to the target by monitoring the estimated motion costs.

Connection Action The connection action requires accurate perception and control of the chain structured MSRR system, which is challenging for most MSRRs due to the tolerance stack of the chain and the limited alignment tolerance of the connector. The freeform modular robots have a large alignment tolerance, making robust self-reconfiguration in 3D more realizable. However, the configuration identification error of the chain configuration accumulates with the chain length and may exceed the alignment tolerance of the connector. We define a strategy to ensure the robustness of the connection action execution.

Given a target configuration $\mathbb{S}^{target}$, we have the corresponding configuration with the same topology as the system configuration, as shown in Figure 9, denoted as $\mathbb{S}^{conn}$. An extruded configuration $\mathbb{S}^{extr}$ can be calculated by reducing the length of the edges connecting the merged nodes in the upcoming cycle to guarantee a robust connection. The system is controlled to move to $\mathbb{S}^{conn}$ without connection at first. After reaching $\mathbb{S}^{conn}$, the connection command is triggered by the reaching detection, and the system is controlled to move to $\mathbb{S}^{extr}$ simultaneously. The action is executed after the expected topology change is detected.

Disconnection Action The topological changing required to finish the disconnection action can be detected by comparing the topology graphs $\mathbb{G}^{init}$ and $\mathbb{G}^{target}$. The disconnection command of the connector can be generated from the change, and the disconnection action can be executed by first disconnecting the connector and then moving to the target configuration.

As shown in Figure 10, after sending the disconnection command to the connector and the topological change is successfully detected, the moving action starts using the switched target configuration. The action is executed after moving to the target configuration.

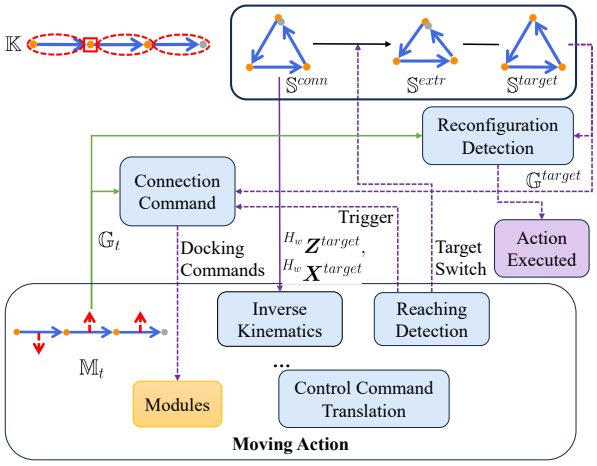

Figure 9. Connection action execution workflow. The modules first approach the target configuration while maintaining topological invariance and then execute the connection command until the connection is detected.

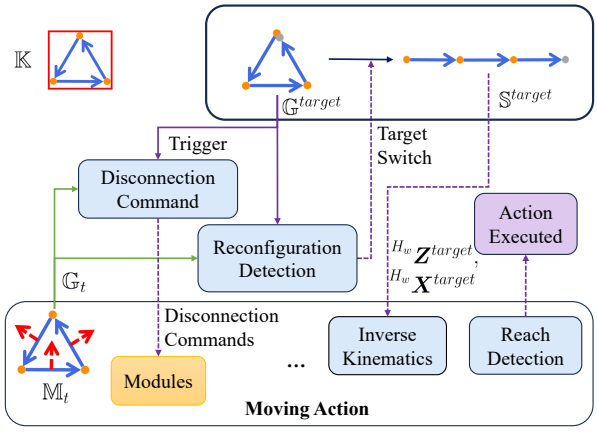

Figure 10. Disconnection action execution workflow. The modules first complete the disconnection command and then move to the target configuration using a moving action.

Parallel Connection and Disconnection Action Multiple connections or disconnections may be required during one action, where the connection and disconnection of connectors are executed in parallel. Such action can be executed in the following way. The topological changes are classified into connections and disconnections. First, the disconnection commands are sent to the modules in parallel. After all topological disconnection changes are detected, the system moves to the target configuration without connections. The connection commands are sent to the modules in parallel, and the extruded target configuration is switched after reaching $\mathbb{S}^{conn}$. Finally, all topological connection changes are detected, and the action is successfully executed.

The presented strategy targets executing the action but does not consider the optimality. The optimum parallel execution strategy of such action can be complicated, which is out of the scope of this article.

4.2 Example Control Commands of FreeSN

As shown in Figure 11(a), the modules communicate with the centralized computer through a Wi-Fi router. The relative orientation filter results and docking events of the modules

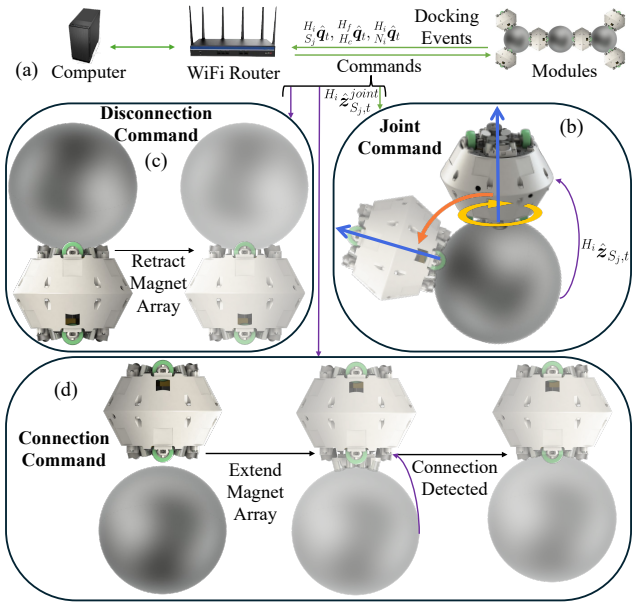

Figure 11. (a) The centralized computer communicates with the modules through a Wi-Fi router. The computer coordinates the module actions by transmitting commands to the modules, and the modules transmit their relative orientation filter results to the computer as feedback. (b) Joint command that controls the connection position vector. (c) Disconnection command that retracts the magnet array. (d) Connection command that improves the misalignment tolerance by extending the magnet array.

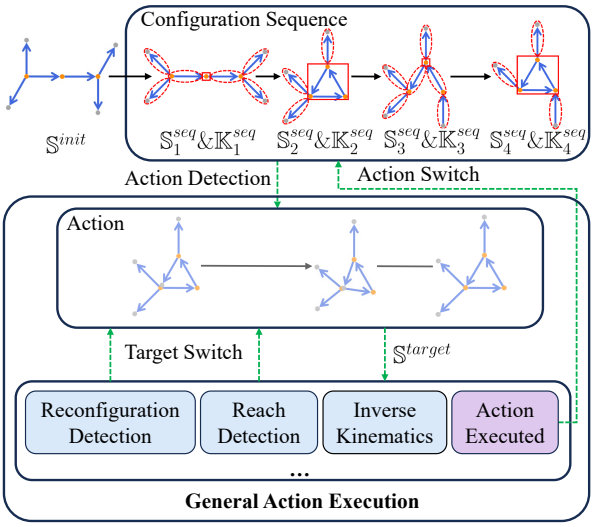

Figure 12. Behavior execution by decomposing the configuration sequence into actions and executing them in order.

are transmitted to the computer in real time. The computer estimates the system configuration and outputs module commands based on action design and kinematics. The modules execute the control commands distributively, using the sensing feedback from the adjacent modules.

As shown in Figure 11(b), the strut module controls its connection position vector ${}^{H_i}\hat{\boldsymbol{z}}_{S_j}$ relative to the horizontal frame of the adjacent node module. The joint connection position vectors are transformed into the node horizontal frames of the corresponding adjacent node modules. The local controller of the strut module can finish the control command with feedback from the local orientation filters.

The connection and disconnection of the connector can be achieved by controlling the position of the magnet array. As shown in Figure 11(c), when a disconnection command is sent to the controller, the corresponding magnet array is controlled to retract into the strut module. The magnetic force between the magnet array and the nearby node module becomes negligible, which can be detected by the configuration identification system. As shown in Figure 11(d), when a connection command is sent to the controller, the corresponding magnet array is first controlled to extend out of the strut module. The misalignment tolerance of the connector can be significantly improved in this way. After reconfiguration is detected, the magnet array is controlled back to the origin position, where the connector has the largest driving force.

4.3 Behavior Execution

The action execution assumes that the modules execute the control commands in the shortest arc without cooperation between modules, and moveable target configurations are limited. By reaching a sequence of configurations, the system can finish complicated behaviors such as locomotion and self-reconfiguration.

As shown in Figure 12, a sequence of skeletal configurations and the corresponding kinematic trees are predefined, where the action from any configuration to the next one should be feasible. Assume that the first configuration is a feasible target of the system, the first configuration is selected as the target, and the corresponding kinematic tree is loaded. The action can be automatically detected and executed based on the action type. After executing the action, the following configuration and kinematic tree are switched. The configuration sequences can be executed by repeating this procedure.

5 Configuration Matching and Mapping

The action execution in Section 4 requires that the target and system configurations have identical module mapping. For a newly constructed system, online motion planning for self-reconfiguration or locomotion can be time-consuming, and optimality can hardly be guaranteed since the system typically has a large number of degrees of freedom. The system configuration may be isomorphic to the existing configurations, and the system can move to the isomorphic configuration if the isomorphic configuration is matched and the optimal module mapping between them is searched.

In this section, a heuristic isomorphism tree search algorithm is proposed to search the candidate mappings with the minimum estimated distance between configurations as heuristic. Then, a configuration matching and mapping algorithm for the spherical freeform modular robots is proposed, which determines a suboptimal module mapping from a given library of kinematic trees and configurations, prioritizing minimal inverse kinematics cost.

Algorithm 2 Isomorphism Tree Search

Input: $\mathbb{S}^{sys}, \mathbb{S}^{ref}$
Output: $\{\varepsilon', \boldsymbol{f}'_{node} : N^{sys} \rightarrow N^{ref}, \boldsymbol{f}'_{edge} : E^{sys} \rightarrow E^{ref}, inv\}$

$\overline{\mathbb{S}}^{sys} = undirected(\mathbb{S}^{sys}), \overline{\mathbb{S}}^{ref} = undirected(\mathbb{S}^{ref})$
if $numModules(\mathbb{S}^{sys}) \neq numModules(\mathbb{S}^{ref})$ **then**
  RETURN // not isomorphic
end if
if $orderedDegree(\overline{\mathbb{S}}^{sys}) \neq orderedDegree(\overline{\mathbb{S}}^{ref})$ **then**
  RETURN // not isomorphic
end if
Create a priority queue $\boldsymbol{q}$ // with connection position vector matching distance ε as the key.
$put(\boldsymbol{q}, 0, (\emptyset, \emptyset, [\ ], [\ ][\ ][\ ]))$
while not $empty(\boldsymbol{q})$ **do**
  $\varepsilon', \Lambda' = get(\boldsymbol{q})$ // pop the item with min distance
  $\boldsymbol{f}_{node}{}', \boldsymbol{f}_{edge}{}', \boldsymbol{Z}^{sys\prime}, \boldsymbol{Z}^{ref\prime}, \boldsymbol{X}^{sys\prime}, \boldsymbol{X}^{ref\prime} = \Lambda'$
  if $size(\boldsymbol{f}_{node}{}') = numNodes(\mathbb{S}^{sys})$ **then**
    YIELD $\varepsilon', \boldsymbol{f}_{node}{}', \boldsymbol{f}_{edge}{}'$ // outputs the mapping
    CONTINUE
  end if
  $N^{sys}_{next} = nextNode(\boldsymbol{f}_{node}{}')$ // select next node
  $C = feasibleCandidates(N^{sys}_{next}, \boldsymbol{f}_{node}{}', \overline{\mathbb{S}}^{sys}, \overline{\mathbb{S}}^{ref})$
  for $N^{ref}_{next} \in C$ **do** // multiple feasible node mappings
    $\varepsilon'', \Lambda'' = subConfigMatch(\Lambda', N^{sys}_{next}, N^{ref}_{next}, \mathbb{S}^{sys}, \mathbb{S}^{ref}, \overline{\mathbb{S}}^{sys}, \overline{\mathbb{S}}^{ref})$
    $put(\boldsymbol{q}, \varepsilon'', \Lambda'')$ // update priority queue
  end for
end while

5.1 Isomorphism Tree Search

The topology graph isomorphism is a common subgraph isomorphism problem in graph theory. Many algorithms (Lueker and Booth 1979; Cordella et al. 2004; Carletti et al. 2015; Jüttner and Madarasi 2018) have been proposed to efficiently reduce the searching space by optimizing the node match order and the cutting rules, considering node degrees and discrete node labels. Based on the VF2++ (Jüttner and Madarasi 2018) subgraph isomorphism algorithm, we propose an isomorphism tree search algorithm that takes the current system skeletal configuration and a reference skeletal configuration as inputs and outputs the node and edge mappings between the two configurations in ascending order of their distances.

As presented in Algorithm 2, the algorithm first checks the signatures of the input configurations to reduce the computational overhead when the input configurations are not isomorphic. Then, it builds an isomorphism tree by continually selecting node mapping pairs and estimating the distance between the mapped sub-skeletal configurations. Each node in the isomorphism tree includes the current module mappings of configurations, cached configuration vectors, and the estimated distance for the current mapping. The node searching order of the configuration is predefined based on the type and degrees of the node in the same way as VF2++. The searching order of the isomorphism tree is decided by the estimated distance, which is managed by a priority queue q. If a leaf node of the isomorphism tree is searched, the algorithm outputs the distance and the module mapping as a candidate, which is then used by the configuration matching and mapping algorithm.

Algorithm 3 Sub-configuration Matching

Input: $\Lambda', N^{sys}_{next}, N^{ref}_{next}, \mathbb{S}^{sys}, \mathbb{S}^{ref}, \overline{\mathbb{S}}^{sys}, \overline{\mathbb{S}}^{ref}$
Output: ε'', Λ''

$\boldsymbol{f}_{node}{}', \boldsymbol{f}_{edge}{}', \boldsymbol{Z}^{sys\prime}, \boldsymbol{Z}^{ref\prime}, \boldsymbol{X}^{sys\prime}, \boldsymbol{X}^{ref\prime} = \Lambda'$
$\boldsymbol{f}_{node}{}'' = \boldsymbol{f}_{node}{}' \cup (N^{sys}_{next} \rightarrow N^{ref}_{next}), \boldsymbol{f}_{edge}{}'' = \boldsymbol{f}_{edge}{}'$
$\boldsymbol{Z}^{sys\prime\prime} = \boldsymbol{Z}^{sys\prime}, \boldsymbol{Z}^{ref\prime\prime} = \boldsymbol{Z}^{ref\prime}$
$\boldsymbol{X}^{sys\prime\prime} = \boldsymbol{X}^{sys\prime}, \boldsymbol{X}^{ref\prime\prime} = \boldsymbol{X}^{ref\prime}$
for $N^{sys}_{neigh} \rightarrow N^{ref}_{neigh} \in neighbor(N^{ref}_{next}, \boldsymbol{f}_{node}{}'', \overline{\mathbb{S}}^{sys}, \overline{\mathbb{S}}^{ref})$ **do**
  $E^{sys}_{neigh} = (N^{sys}_{next}, N^{sys}_{neigh})$
  $E^{ref}_{neigh} = (N^{ref}_{next}, N^{ref}_{neigh})$
  $inv = XOR(E^{sys}_{neigh} \in \mathbb{S}^{sys}, E^{ref}_{neigh} \in \mathbb{S}^{ref})$
  // extend edge mapping
  $\boldsymbol{f}_{edge}{}'' = \boldsymbol{f}_{edge}{}'' \cup (E^{sys}_{neigh} \rightarrow E^{ref}_{neigh}, inv)$
  $\hat{\boldsymbol{z}}^{sys}, \hat{\boldsymbol{x}}^{sys} = getVectors(\overline{\mathbb{S}}^{sys}, E^{sys}_{neigh})$
  $\hat{\boldsymbol{z}}^{ref}, \hat{\boldsymbol{x}}^{ref} = getVectors(\overline{\mathbb{S}}^{ref}, E^{ref}_{neigh})$
  if inv **then**
    $\hat{\boldsymbol{z}}^{ref} = -\hat{\boldsymbol{z}}^{ref}$
  end if
  // extend the configuration vectors
  $\boldsymbol{Z}^{sys\prime\prime} = \begin{bmatrix} \boldsymbol{Z}^{sys\prime\prime} & \hat{\boldsymbol{z}}^{sys} \end{bmatrix}$
  $\boldsymbol{X}^{sys\prime\prime} = \begin{bmatrix} \boldsymbol{X}^{sys\prime\prime} & \hat{\boldsymbol{x}}^{sys} \end{bmatrix}$
  $\boldsymbol{Z}^{ref\prime\prime} = \begin{bmatrix} \boldsymbol{Z}^{ref\prime\prime} & \hat{\boldsymbol{z}}^{ref} \end{bmatrix}$
  $\boldsymbol{X}^{ref\prime\prime} = \begin{bmatrix} \boldsymbol{X}^{ref\prime\prime} & \hat{\boldsymbol{x}}^{ref} \end{bmatrix}$
end for
// estimate the distance of sub-skeletal configurations
if $size(\boldsymbol{f}_{node}{}'') <= 2$ **then**
  $\varepsilon'' = 0$
else
  $\varepsilon'' = vectorMatch(\boldsymbol{Z}^{sys\prime\prime}, \boldsymbol{Z}^{ref\prime\prime}, \boldsymbol{X}^{sys\prime\prime}, \boldsymbol{X}^{ref\prime\prime})$
end if
$\Lambda'' = \boldsymbol{f}_{node}{}'', \boldsymbol{f}_{edge}{}'', \boldsymbol{Z}^{sys\prime\prime}, \boldsymbol{Z}^{ref\prime\prime}, \boldsymbol{X}^{sys\prime\prime}, \boldsymbol{X}^{ref\prime\prime}$

undirected transforms the directed skeletal configuration to the undirected graph. *put* and *get* are the priority queue operation functions. *size* gets the number of items in the mapping. *numModules* returns the number of each type of module in the configuration, *numNodes* returns the number of graph nodes, and *orderedDegree* returns the degree of each node type for all nodes in order. *nextNode* decides the next node of the system configuration that should be mapped next based on the predefined node searching order. *feasibleCandidates* finds the candidate nodes of the reference configuration that can possibly mapped with the given node based on the degrees of neighbor nodes. *subConfigMatch* extends the isomorphism tree with a new node mapping pair, as presented in Algorithm 3.

Algorithm 3 takes the previous mappings, cached configuration vectors, and new node mapping pair as inputs. The algorithm first extends the node mappings and edge mappings. Then, it updates the vectors of the sub-skeletal configurations and estimates the distance. *neighbor* finds the node mapping pairs that have been

searched and are neighbors of the new node mapping pair. *getVectors* gets the connection position vector and forward direction vector given the skeletal configuration and edge. *vectorMatch* estimates the distance between two sub-skeletal configurations by aligning the connection position vectors based on equation (8).

For FreeSN, the node mappings represent the node module mappings, and edge mappings represent the strut module mappings and whether the connection directions are the same. The node mappings represent the module mappings, and edge mappings represent the connection direction for other freeform modular robots. The forward direction vectors are not included in the skeletal configuration for FreeSN, and they can be ignored during the isomorphism tree search.

5.2 Matching and Mapping

Given the current system skeletal configuration $\mathbb{S}^{sys}$ and a library $\mathbb{L}$ of kinematic trees and skeletal configurations, the configuration matching and mapping algorithm aims to find the kinematic tree and reference skeletal configuration $\mathbb{S}^{ref}$ in the library and the module mapping with the minimum motion cost. The full-body inverse kinematics is used to estimate the motion cost without motion planning.

The algorithm first finds the possible reference configurations and kinematic trees from the library by indexing the module number and the ordered degrees. The isomorphism searching algorithm outputs the mappings with estimated configuration distances in order for each pair of candidate configuration and kinematic tree. The estimated distances of the mappings have a similar order to the motion cost of inverse kinematics if the two configurations are not far away. If the estimated distance is much larger than the minimum motion cost estimated by the inverse kinematics, the isomorphism searching of candidates can be terminated. A sub-optimum candidate and module mapping can be found within a limited time, as shown in the algorithm below.

Algorithm 4 Configuration Matching and Mapping

Input: $\mathbb{S}^{sys}, \mathbb{L}$
Output: $\boldsymbol{f}_{node} : N^{sys} \rightarrow N^{ref}, \boldsymbol{f}_{edge} : E^{sys} \rightarrow E^{ref}, inv\ \mathbb{K}^{ref}$

$\boldsymbol{f}_{node} = \emptyset, \boldsymbol{f}_{edge} = \emptyset, \mathbb{K}^{ref} = null$
$cost_{min} = +\infty$
for $\mathbb{S}^{ref'}, \mathbb{K}^{ref'} \in$ *subset*$(\mathbb{L},$ *numModules*$(\mathbb{S}^{sys}),$ *orderedDegree*$(\mathbb{S}^{sys}))$ **do**
  for $\varepsilon', \boldsymbol{f}_{node}{}', \boldsymbol{f}_{edge}{}' \in$ *isoSearch*$(\mathbb{S}^{sys}, \mathbb{S}^{ref'})$ **do**
    if $\varepsilon' > cost_{min} * \sigma$ **then**
      BREAK
    end if
    $cost =$ *inverseKin*$(\mathbb{K}^{ref'}, \mathbb{S}^{sys}, \mathbb{S}^{ref'}, \boldsymbol{f}_{edge}{}')$
    if $cost < cost_{min}$ **then**
      $\boldsymbol{f}_{node} = \boldsymbol{f}_{node}{}', \boldsymbol{f}_{edge} = \boldsymbol{f}_{edge}{}'$
      $\mathbb{K}^{ref} = \mathbb{K}^{ref'}$
      $cost_{min} = cost$
    end if
  end for
end for

isoSearch is the isomorphism searching algorithm proposed in the previous section. *inverseKin* is the inverse kinematics algorithm proposed in Section 3.4. σ is the parameter of the confidence that using the configuration distance to estimate the motion cost of inverse kinematics, which balances the recognition speed and the probability of finding an optimum motion cost.

However, inverse kinematics does not consider the validity of motion, and it only estimates the actual cost from the system configuration to reach the reference configuration. For the robotic system with a high degree of freedom, the algorithm can not guarantee finding the optimum solution within a limited time. Our proposed configuration matching and mapping algorithm aims to find a sub-optimal solution within limited time. The configuration distance can generally estimate a similar mapping order if the system configuration is not far from the reference configuration, where the inverse kinematics is more likely to have an accurate cost estimation without motion planning. The configuration distance estimation is much faster than the inverse kinematics algorithm, significantly improving configuration matching and mapping speed.

6 Locomotion and Self-reconfiguration Autonomy Framework

The freeform modular robot systems can finish locomotion and self-reconfiguration tasks by executing configuration sequences. However, real-time planning of optimum configuration sequence is hard since the system is floating-base and always includes high degrees of freedom and strong constraints. This section proposes a locomotion and self-reconfiguration autonomy framework for freeform modular robots.

The autonomy framework is introduced from three aspects: library design, library creation, and behavior retrieval and execution. As shown in Figure 13(a-c), we implement a GUI tool to design kinematic tree and skeletal configurations and capture configurations from real robots or simulations. Then, a library can be built from the collected data, which hierarchically stores the behaviors and feasible kinematic transitions between configurations, as shown in Figure 13(d). The feasibility of configuration sequences are evaluated using simulation during the library build. If a sequence fails to execute in simulation, the user should tune it until it succeeds. In this way, we can assume that the configuration sequences defined in the built library are feasible. The modules can move from one configuration to the next in the sequence without requiring synchronized module motions, while avoiding self-collision, gravity instability, and motor torque insufficiency. For a newly constructed robot system, if the system topology is isomorphic to a known one in the library, the configuration in the library that is closest to it can be found by configuration matching and mapping. After a target configuration or behavior is selected, the configuration sequence with intermediate kinematic trees and module mappings can be retrieved from the library, as shown in Figure 13(f). As shown in Figure 13(g), the sequence can be mapped to have identical module mappings with the system and then executed as described in Section 4.

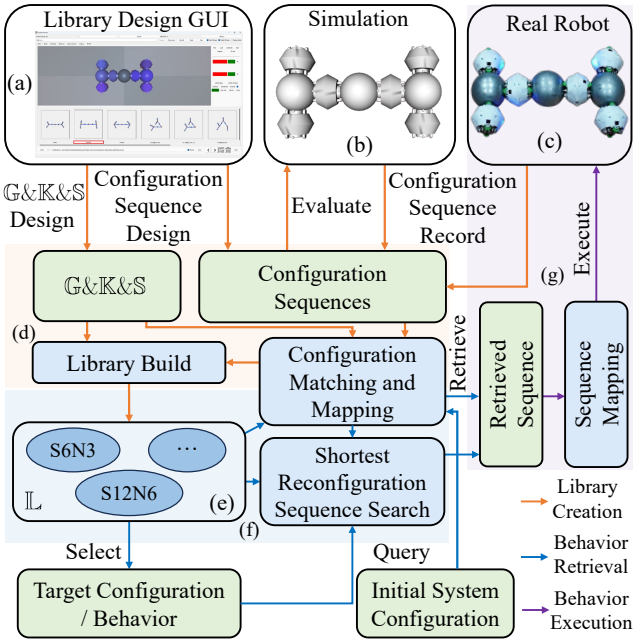

Figure 13. Locomotion and self-reconfiguration framework. (a) A Graphical User Interface (GUI) tool for library design. (b) A simulation environment using MuJoCo for recording configurations and evaluating the feasibility of actions. (c) The real robot platform for recording configurations and executing behaviors. (d) Automatic library creation with kinematic trees and configuration sequences as inputs. (e) The library is created and hierarchically stored as a graph, which can be divided into sub-libraries based on module numbers. (f) Behavior retrieval with initial and target configurations as inputs. (g) The retrieved configuration sequence can be mapped and then executed on real robot.

6.1 Library Design

The library $\mathbb{L}$ adopts a hierarchical design. The library is divided into many sub-libraries based on the number of modules. Taking FreeSN as an example, as shown in Figure 13(e), the library is divided into "S6N3", "S12N6", etc. Sub-library "S6N3" contains configurations with six strut modules and three node modules. For each sub-library, the configurations are classified into different groups based on the isomorphism of the topology graph. For each group, a reference topology graph $\mathbb{G}^{group}$ and at least one kinematic tree $\mathbb{K}_k^{group}$ and skeletal configuration $\mathbb{S}_i^{group}$ with the same topology as $\mathbb{G}^{group}$ should be defined. A sub-library can be regarded as a directed graph, where graph nodes are the skeletal configurations. An edge connection represents a feasible action from one node to another, with a kinematic tree and motion cost as edge attributes. An optimum mapping between two nodes should also be defined as an edge attribute for the edges crossing two groups. A sub-library "S6N3" containing four groups is shown in Figure 14(f) as an example. The four green circles represent four types of isomorphism topology graphs, and each includes kinematic trees in yellow rounded rectangles, skeletal configurations, and behaviors in blue rounded rectangles. The feasible actions with kinematic tree and module mapping as edge attributes are represented with black arrows.

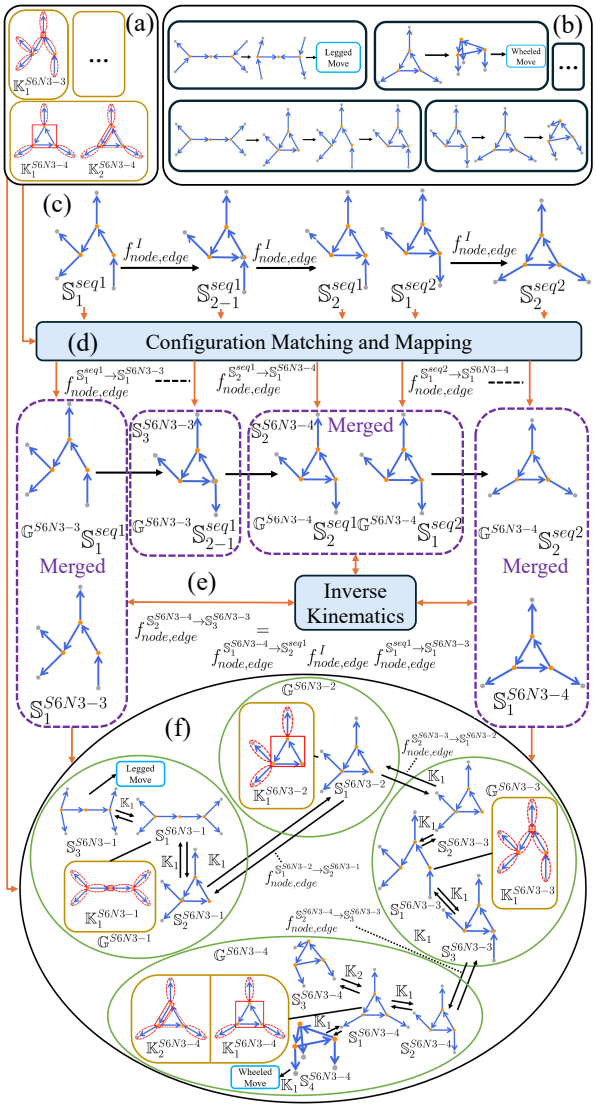

Figure 14. Example of automatic library creation. (a) Designed kinematic trees with topology graphs. (b) Collected configuration sequences. (c) Example configuration sequences with intermediate configurations during topology changes. (d) The configurations in the sequences are matched with the library and remapped to share the same topology graph. (e) Configurations merging by checking the costs from inverse kinematics. (f) Example of constructed sub-library that includes four groups with different topology graphs, utilizing structured data from (a) and (e). Each group includes kinematic trees in yellow rounded rectangles, skeletal configurations, and defined behaviors in blue rounded rectangles. The feasible actions between the skeletal configurations are defined as the edges of the directed graph, represented by black arrows.

6.2 Library Creation

We implement a GUI tool to design the library, as shown in Figure 13(a). At least one kinematic tree and reference skeletal configuration for each isomorphism topology graph should be defined using the tool.

A simulation environment of FreeSN is implemented with MuJoCo engine (Todorov et al. 2012), as shown in Figure 13(b). The controllers of modules are implemented as the same interface as real modules, and a simulation can be directly launched from a configuration. The modules can be

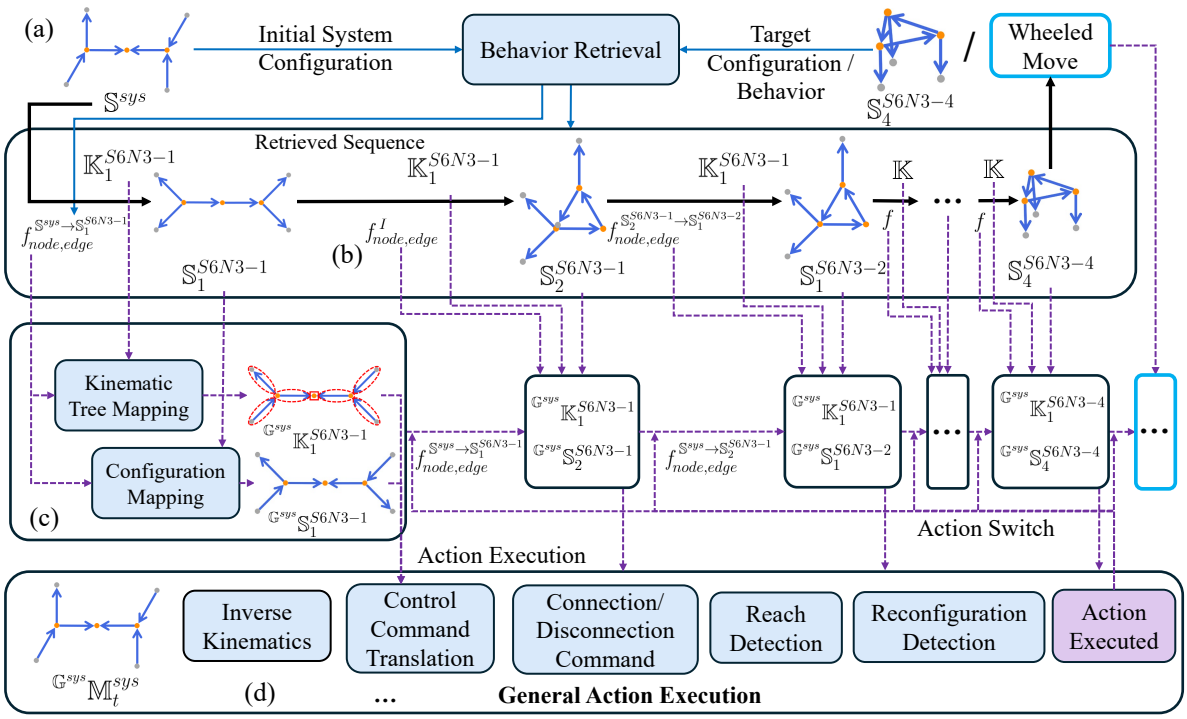

Figure 15. Example of library behavior retrieval and execution. (a) Configuration matching and shortest reconfiguration sequence search, using the current system configuration and a selected target configuration as inputs. (b) The retrieved configuration sequence with kinematic trees and module mappings. (c) The kinematic trees and configurations are mapped to have identical module mappings with the system, ensuring that the sequence is executable. (d) The sequence is executed using the framework proposed in Section 4.

teleoperated, while the system configurations and behaviors can be recorded from simulation and real robots.

A recorded configuration sequence or a mapping from low-dimension behavior parameters to the high-dimension control parameters of the system can be labeled as a behavior that guides the self-reconfiguration or locomotion of the system. The designed configurations and recorded configurations may have different topology graphs. The library can be automatically built from these data by applying the configuration matching and mapping algorithm so that the relationships between skeletal configurations, kinematics trees, and behaviors are stored. During the library building process, the feasibility of the configuration sequences and behaviors are evaluated using the simulation. If a sequence or behavior fails to execute in simulation, the user should tune it until it succeeds. The connection strength of FreeSN modules is measured and evaluated (Wu et al. 2024) through experiments. There always exists a gap between the simulation and real robots, so the connection strength and torque limits in the simulation are set lower than those of real robots to better ensure the validity of the simulation.

As shown in Figure 14, the creation of the "S6N3" library is illustrated as an example. The designed topology graph and kinematic trees are presented in Figure 14(a), and the collected configuration sequences are shown in Figure 14(b). Each sequence of configurations should have identical module mappings, and intermediate configurations during topology changes can be generated in the same way as described in Section 4. As shown in Figure 14 (c), two subsequences containing intermediate configurations $\{\mathbb{S}_i^{seq1}\}, \{\mathbb{S}_i^{seq2}\}, i \in (1, 2)$ with isomorphic topology graphs are shown as examples.

The proposed configuration matching and mapping algorithm can be applied to find the module mapping from the sequence configurations to the reference configurations defined in the library. For example, the optimal mapping $\boldsymbol{f}_{node,edge}^{\mathbb{S}_1^{seq1}\to\mathbb{S}_1^{S6N3-3}}$ between $\mathbb{S}_1^{seq1}$ and $\mathbb{S}_1^{S6N3-3}$ are searched by the configuration matching and mapping, and the sequence configuration are mapped to $^{\mathbb{G}^{S6N3-3}}\mathbb{S}_1^{seq1}$ by the mapping. Then, the sequence configurations can be mapped to share the same topology graph as the reference ones. Finally, the configurations with the same topology graph are merged by checking the cost of inverse kinematics, as shown in Figure 14(e). The feasible actions defined in the configuration sequences and the module mapping between configurations are calculated and stored as the edge connection of the library. The whole library can be built by matching, mapping, and merging all the configuration sequences in the same way. The built "S6N3" library is shown in Figure 14(f).

6.3 Locomotion and Self-reconfiguration Execution

After library creation, for a newly constructed system with a target configuration or behavior, the configuration sequence and the behavior can be retrieved from the library, which is then executed by the system.

As shown in Figure 15 (a), a newly constructed system with initial configuration $\mathbb{S}^{sys}$ is presented, and a target configuration $\mathbb{S}_4^{S6N3-4}$ or a behavior labeled as "wheeled move" starting from $\mathbb{S}_4^{S6N3-4}$ is selected as the target. The isomorphic configuration $\mathbb{S}_1^{S6N3-1}$ and the corresponding kinematic tree $\mathbb{K}_1^{S6N3-1}$ in the library are matched by

configuration matching and mapping, and the optimal module mapping $\boldsymbol{f}_{node,edge}^{\mathbb{S}^{sys}\to\mathbb{S}_1^{S6N3-1}}$ is searched, as described in Figure 13(f). The shortest reconfiguration sequence as shown in Figure 15 (b) from $\mathbb{S}_1^{S6N3-1}$ to $\mathbb{S}_4^{S6N3-4}$ can be retrieved from the library by graph shortest path search. The configurations in the sequence are connected by black arrows, which mean feasible actions with the given kinematic tree and module mapping. The configurations $\mathbb{S}^{sys}$ and $\mathbb{S}_1^{S6N3-1}$ can be different and the feasibility of action from $\mathbb{S}^{sys}$ to $\mathbb{S}_1^{S6N3-1}$ is not guaranteed.

Assuming that the two configurations are not far away and the action is feasible, we have the feasible configuration sequence from the system to the target. Then, the configurations and the kinematic trees are mapped to have the identical module mapping with the system in sequence, as shown in Figure 15 (c). The system can execute the sequence as described in Section 4. After reaching the target configuration $\mathbb{S}_4^{S6N3-4}$, the selected behavior can be executed similarly, and autonomous locomotion and self-reconfiguration are executed.

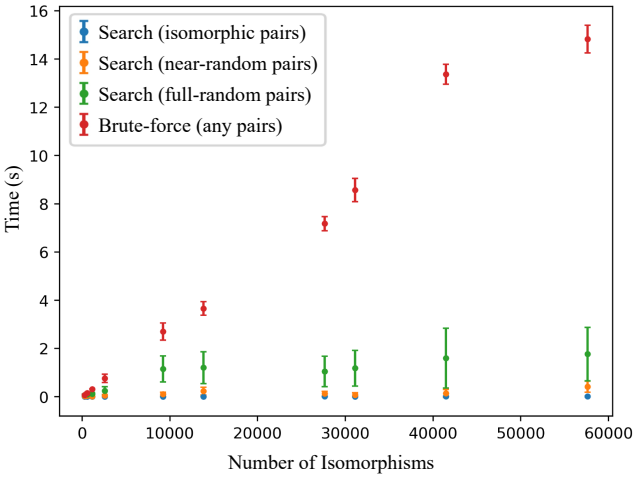

Figure 16. The computational times of isomorphism tree search and brute-force sorting algorithms using three different types of random configurations to find the mapping that minimizes distance. The brute-force sorting time increases nearly linearly with the number of isomorphisms using any configuration pairs.

7 Experiments

In this section, the isomorphism tree search and full-body inverse kinematics efficiency is preliminarily evaluated. We also present case studies with FreeSN hardware to demonstrate the capability of our system, with up to twelve strut modules and six node modules containing 48 motors for joints motion control and 24 motors for docking control. The proposed autonomy framework is executed centrally on a computer equipped with an Intel Core i9-12900H processor and communicates with the FreeSN modules through a Wi-Fi router at 10 Hz. The communication bandwidth of the computer is approximately $60\,\mathrm{kbit\,s^{-1}}$ at maximum and increases linearly with the number of modules. The FreeSN system can accomplish autonomous locomotion and self-reconfiguration behaviors without external sensors using a library. The videos of autonomous locomotion and self-reconfiguration demonstrations can be found in the attached multimedia files.

7.1 Isomorphism Tree Search Efficiency

The isomorphism tree search algorithm is designed to efficiently search the candidate module mappings between two configurations in ascending order of their distance. Only a few candidate mappings need to be searched if the estimated distances of the mappings have the similar order with the motion cost of inverse kinematics. Ten topology graphs with sixteen edges and five nodes are designed with numerous isomorphisms from dozens to tens of thousands. The computational times of isomorphism tree search and brute-force sorting algorithms are compared by generating random configuration pairs in various ways to find the mapping that minimizes distance. As shown in Figure 16, three types of random configuration pairs are generated: isomorphic, near-random, and full-random. The first configurations of the random configuration pairs are randomly sampled from the whole joint space, and the second configurations have the same position vectors with random θ and module mapping for isomorphic type. The near-random type samples the second configurations near the zero point of joint space of the first configurations, and the root-mean-square (RMS) value of angles between generated position vectors is about $24°$. The full-random type samples the second configuration on the whole joint space.

The brute-force sorting time is nearly linear with the number of isomorphisms using any random configuration pairs. The isomorphism tree searching is much faster and does not depend on the number of isomorphisms using isomorphic type of random configurations pairs. The searching times are within $25\,\mathrm{ms}$ with the mean value of $10\,\mathrm{ms}$ for the topology graph with 57600 isomorphism. The distance between two configurations mainly decides the searching time since a smaller distance leads to a more accurate heuristic.

7.2 Full-body Inverse Kinematics Efficiency

In this section, the efficiency of full-body skeletal inverse kinematics is evaluated, which also determines the main computation demands of the system. To better evaluate the inverse kinematics speed, we select tree-based topology graphs with edge numbers ranging from 2 to 30. For each topology graph, one thousand near-random configuration pairs are generated. The inverse kinematics is solved using L-BFGS-B optimization algorithm (Byrd et al. 1995), with analytical Jacobian, zero initial variables, and threshold value of 1×10^{-5}. The complexity of single L-BFGS-B iteration is $O(n)$, where n is the variable dimension and is linear with the number of edges of the topology graph. The number of iterations depends on characteristics of objective function, and should be weakly correlated with variable dimension in practice. As shown in Figure 17(a), the RMS value of angles between generated position vectors of configuration pairs for each topology graph is about $24°$, and the inverse kinematics complexity of the numerical approximation for the generated configurations is roughly $O(n^{1.35})$.

The inverse kinematic efficiency is also related with the distance between the initial variables and target. One

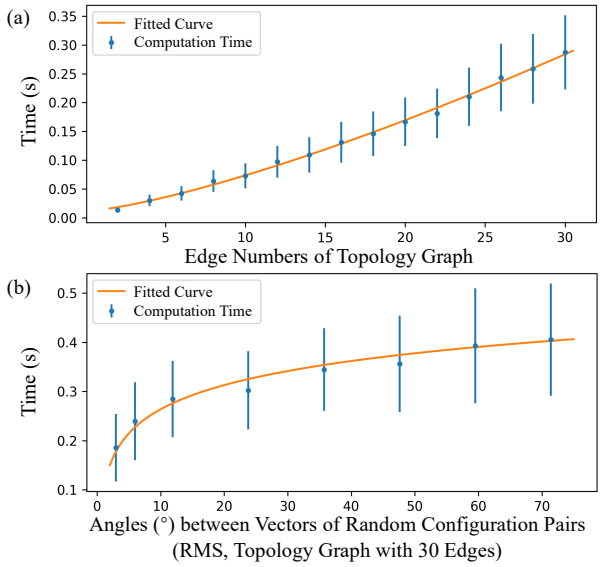

Figure 17. (a) Computational times for full-body inverse kinematics of tree-based topology graphs with varying edge numbers using near-random configuration pairs. (b) The computational time versus the RMS value of the angles between the generated position vectors of configuration pairs.

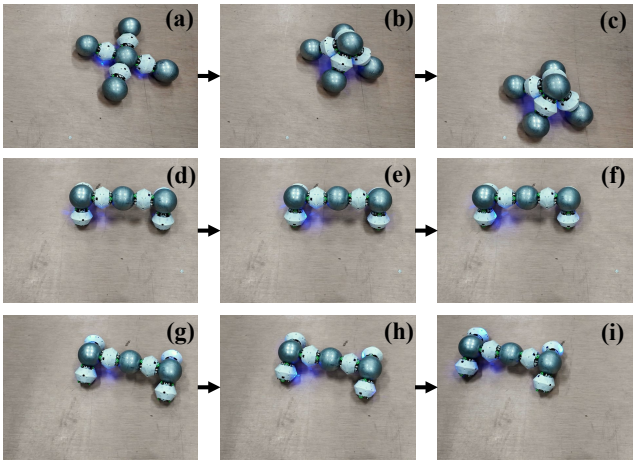

Figure 18. (a)-(c): The modules move forward using node modules as wheels. (d)-(f): The modules move forward using wheels of strut modules. (g)-(i): The modules move forward using strut modules as wheels.

thousand near-random configuration pairs are generated with varying variances for the topology graph with 30 edges. As shown in Figure 17(b), the x-axis represents the RMS value of angles between generated position vectors of configuration pairs. The inverse kinematics speed increases roughly logarithmically with the angular distance for the generated configurations, which also decides the efficiency of iterative inverse kinematics.

7.3 Autonomous Locomotion Demonstration

The FreeSN system has various functions and can accomplish locomotion in many different ways, such as moving as wheeled and legged robots. We design the sub-libraries and the locomotion behaviors, and the real robot system can execute the behaviors with arbitrary isomorphic topology connection.

We define three behaviors, where the modules move to a target configuration and then drive the system as a wheeled vehicle in different ways. As shown in Figure 18(a)-(c), the system consisting of four strut modules and five node modules moves to a standup configuration at first. Then, the four leaf node modules are driven by the connectors connected to them as wheels, and the system can move using four wheels with the defined mapping between the system speed and the target speed of the connectors.

The FreeSN system can move as a wheeled robot if some wheels of the strut modules are in contact with the ground. As shown in Figure 18(d)-(f), the system moves to a morphology configuration where a wheel of each of the four leaf strut modules contacts with the ground. The system can move forward by driving the four wheels in the same direction.

The FreeSN system can also move as a wheeled robot by driving strut modules as wheels. As shown in Figure 18(g)-(i), the system reaches a configuration where the leaf strut modules are well in contact with the ground. Then, the system can move fast by driving the strut modules to rotate around its z-axis simultaneously.

The FreeSN is capable of squirming forward or crawling forward by reaching a sequence of configurations in order, and we define the loop execution of the sequences as behaviors. As shown in Figure 19(a)-(e), the configuration sequence only contains a squirming-up configuration and a squirming-down configuration. The motion of one of the stem strut modules can be disabled by defining a kinematic tree containing fixed nodes so that the direction of the squirming can be controlled. The FreeSN system successfully squirms forward by alternately reaching two target configurations with different kinematic trees.

As shown in Figure 19(f)-(j), the configuration sequence is designed to crawl forward with the parallel motion of the four leaf strut modules as legs. The system successfully crawls forward a small step for each loop of configuration sequence execution and finally moves to the center of the figure. Systems with similar configurations can adopt this crawling strategy. As shown in Figure 19(k)-(o), the system containing twelve strut modules and six node modules crawls from the left side of the figure to the right side.

7.4 Autonomous Self-reconfiguration Demonstration

We demonstrate autonomous self-reconfiguration of FreeSN in 3D with up to twelve strut modules and six node modules. We define the reference topology graphs and kinematic trees for the "S6N3" and "S12N6" sub-libraries with the library design tool. The configurations and behaviors for the demonstration are also designed, and the sub-libraries are automatically generated. The FreeSN system with a random topology graph realizes continual self-reconfiguration with the autonomy framework, which is rarely demonstrated in previous modular self-reconfigurable robot systems.

The "S6N3" sub-library contains four groups, and a reference topology graph and a kinematic tree are defined for each group. Sequences of configurations are designed. A newly constructed system forms the configuration in Figure 20(a) initially, and a configuration in the library

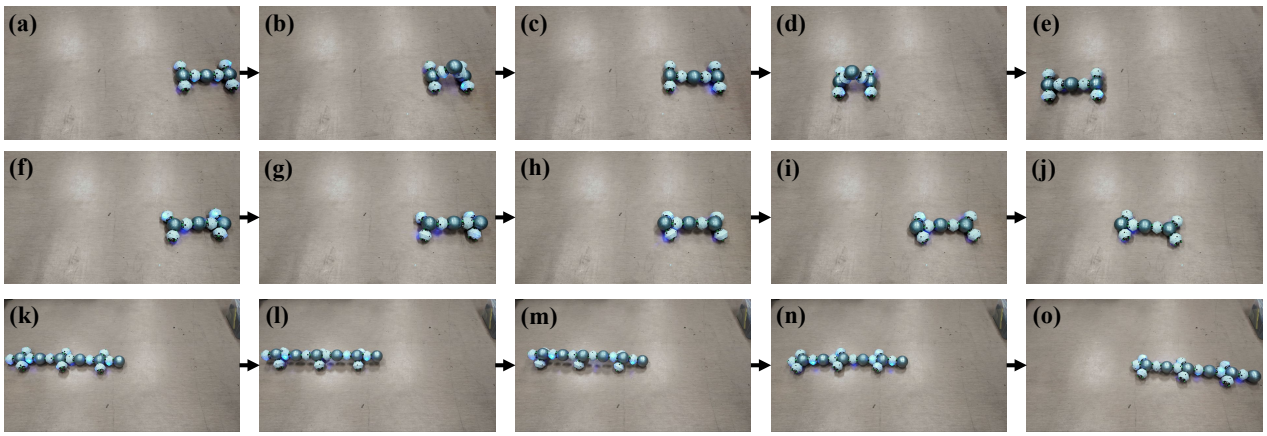

Figure 19. (a)-(e): The modules squirms forward. (f)-(j): The modules crawl forward with the parallel motion of the four leaf strut modules as legs. (k)-(o): The modules with twelve strut modules and six node modules crawl forward using the same approach.

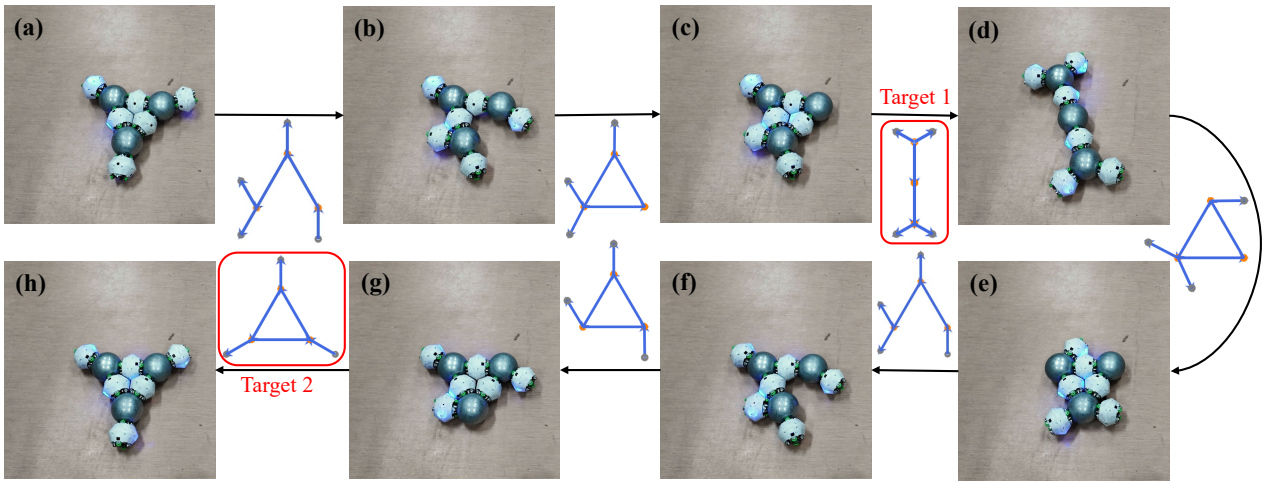

Figure 20. (a): The modules initially form a triangle configuration with three legs, aiming to first reach the configuration in (d) and then reconfigure back to the initial skeletal configuration. (b)-(d): The configuration sequence is retrieved from the library, and the system reaches the first target. (e)-(h): The system successfully reconfigures back to the initial skeletal configuration.

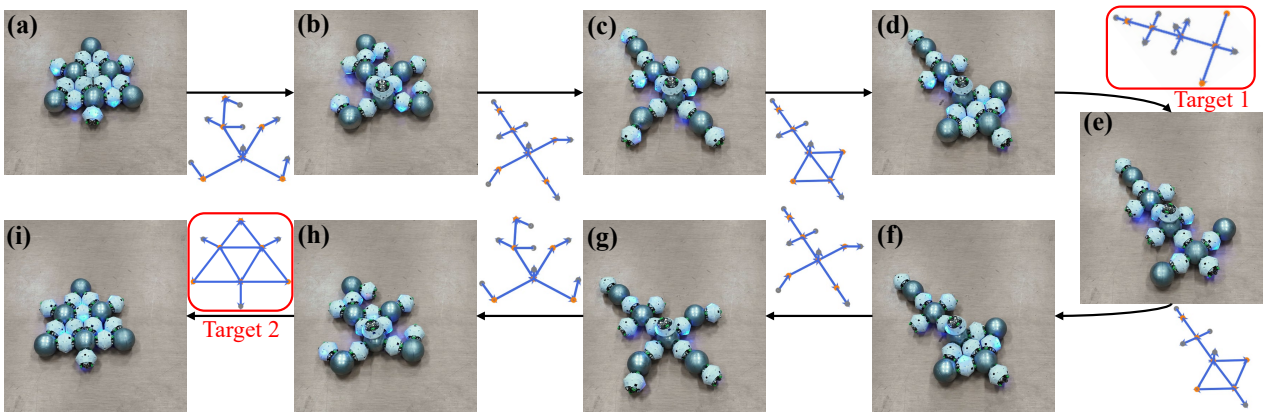

Figure 21. (a): Twelve strut modules and six node modules form a hexagonal star configuration, targeting reconfiguring to a tree configuration and finally reconfiguring back to the hexagonal star. (b)-(e): The modules automatically reconfigure to the tree configuration. (f)-(i): The modules successfully reconfigure back to the hexagonal star configuration.

is chosen as the target, which is isomorphic to the configuration in Figure 20(d). As shown in Figure 20(a)-(d), the reconfiguration sequence is retrieved and then executed by the system. Then, the initial skeletal configuration is chosen as the new target, and the system autonomously reconfigures back to the initial skeletal configuration, as shown in Figure 20(d)-(h).

A sub-library "S12N6" is also designed, which is generated from many designed kinematic trees with parallel robot kinematic nodes, skeletal configurations, and recorded configuration sequences from simulation. As shown in

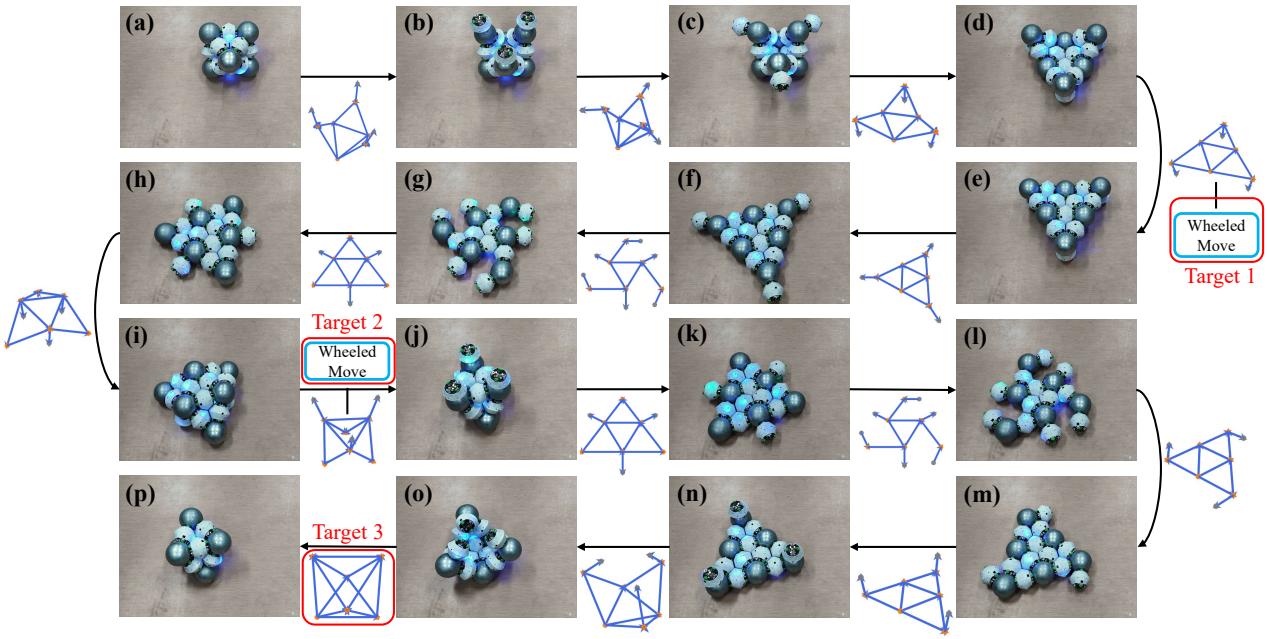

Figure 22. (a): Twelve strut modules and six node modules form a octahedron configuration initially, targeting executing two designed behaviors and finally reconfiguring back to the initial configuration. (b)-(e): The modules automatically reconfigure to the configuration in (e) and execute the wheeled move behavior. (f)-(j): The modules continue to reconfigure to the configuration in (j) and execute the behavior moving with node modules as wheels. (k)-(p): The system successfully reconfigures back to the octahedron configuration.

Figure 21(e),(i), two configurations are set as the targets in order. The system starts from Figure 21(a), reconfigures to the configuration in Figure 21(e) by a sequence of moving and parallel reconfiguration actions, and finally reconfigures back to the original configuration autonomously.

As shown in Figure 22(a), the system forms a octahedron initially, which is structurally stable but has zero degree of freedom. The system aims to finish the wheeled locomotion behaviors in sequence and finally reconfigure back to the octahedron. The two behaviors are defined with the initial configuration shown in Figure 22(e),(j). The whole skeletal configuration sequences and behaviors are retrieved from the library and are shown in Figure 22(a)-(p), and the autonomous execution of the self-reconfiguration and locomotion are successfully finished in 3D with parallel connection and disconnection actions.

8 Discussions and Future Work

This article presents a locomotion and self-reconfiguration autonomy framework for spherical freeform modular robots. The proposed skeletal kinematics provides a general kinematic modeling approach for spherical freeform modular robots, so that the system can efficiently control the skeletal configuration without considering the redundant kinematic parameters during self-reconfiguration. We also proposed a configuration matching and mapping algorithm for freeform modular robots, where the optimality among the feasible isomorphisms is first efficiently considered by heuristic isomorphism tree search. Based on this, a library of relationships between kinematic trees, skeletal configurations, and behaviors can be automatically generated with the designs from a tool and the recorded configuration sequences from modules. Autonomous locomotion and self-reconfiguration are achieved by retrieving the configuration sequence from the library with identical module mapping with the system and then executing the sequence as a set of basic actions.

The presented autonomy framework is only demonstrated on FreeSN, and the configuration identification and control modules within the framework are explained based on the concrete implementation of FreeSN. The framework can be generalized to other modular robotic systems, on the condition that their configurations can be represented as skeletal configurations, and that their joint kinematic nodes and controllers are properly implemented. Additionally, the accuracy of the joint sensing must be sufficient to ensure robust execution of actions by the robot.

A limitation is that the initial system configuration should be closed to its isomorphic configuration in the library so that the action moving to the first target configuration can be assumed to be feasible. Real-time self-collision avoidance is not implemented in the current system, given the assumption that the collected configuration sequences in the library are self-collision free. The implementation of the control system is relatively preliminary. The trajectory tracking controllers and synchronization control of multiple modules should be further researched for better control performance.

The presented autonomous framework provides the foundation to explore and evaluate the high-level algorithms for freeform modular robots. In future work, motion planning and reconfiguration planning that consider self-collision, gravity stability, and torque limitations are of primary interest. In this context, the configurations in the library can be planned fully automatically, and the modules can accomplish locomotion and self-reconfiguration by combining the library and online planning. Other promising

research directions include decentralized relative localization between modules and environment perception using visual sensors on the modules, which are essential for the self-assembly of multiple groups of modules and for solving real-world tasks.

References

Belke CH, Holdcroft K, Sigrist A and Paik J (2023) Morphological flexibility in robotic systems through physical polygon meshing. *Nature Machine Intelligence* 5(6): 669–675. DOI: 10.1038/s42256-023-00676-8. URL https://doi.org/10.1038/s42256-023-00676-8.

Bray E and Groß R (2023) Recent developments in self-assembling multi-robot systems. *Current Robotics Reports* 4(4): 101–116. DOI:10.1007/s43154-023-00106-y. URL https://doi.org/10.1007/s43154-023-00106-y.

Brunete A, Ranganath A, Segovia S, de Frutos JP, Hernando M and Gambao E (2017) Current trends in reconfigurable modular robots design. *International Journal of Advanced Robotic Systems* 14(3): 1729881417710457. DOI:10.1177/1729881417710457. URL https://doi.org/10.1177/1729881417710457.

Byrd RH, Lu P, Nocedal J and Zhu C (1995) A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing* 16(5): 1190–1208. DOI:10.1137/0916069. URL https://doi.org/10.1137/0916069.

Campbell J, Pillai P and Goldstein S (2005) The robot is the tether: active, adaptive power routing modular robots with unary inter-robot connectors. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 4108–4115. DOI: 10.1109/IROS.2005.1545426.

Carletti V, Foggia P and Vento M (2015) Vf2 plus: An improved version of vf2 for biological graphs. In: *Graph-Based Representations in Pattern Recognition: 10th IAPR-TC-15 International Workshop, GbRPR 2015, Beijing, China, May 13-15, 2015. Proceedings 10*. Springer, pp. 168–177.

Chen IM and Burdick J (1993) Enumerating the nonisomorphic assembly configurations of modular robotic systems. In: *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, volume 3. pp. 1985–1992 vol.3. DOI:10.1109/IROS.1993.583905.

Cordella LP, Foggia P, Sansone C and Vento M (2004) A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence* 26(10): 1367–1372.

Daudelin J, Jing G, Tosun T, Yim M, Kress-Gazit H and Campbell M (2018) An integrated system for perception-driven autonomy with modular robots. *Science Robotics* 3(23): eaat4983. DOI:10.1126/scirobotics.aat4983. URL https://www.science.org/doi/abs/10.1126/scirobotics.aat4983.

Davey J, Kwok N and Yim M (2012) Emulating self-reconfigurable robots - design of the SMORES system. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 4464–4469. DOI:10.1109/IROS.2012.6385845.

Dokuyucu Hİ and Özmen NG (2023) Achievements and future directions in self-reconfigurable modular robotic systems. *Journal of Field Robotics* 40(3): 701–746. DOI:https://doi.org/10.1002/rob.22139. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.22139.

Eckenstein N and Yim M (2014) Area of acceptance for 3d self-aligning robotic connectors: Concepts, metrics, and designs. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 1227–1233. DOI:10.1109/ICRA.2014.6907010.

Garcia RFM, Hiller JD, Stoy K and Lipson H (2011) A vacuum-based bonding mechanism for modular robotics. *IEEE Transactions on Robotics* 27(5): 876–890. DOI:10.1109/TRO.2011.2153010.

Gregg CE, Catanoso D, Formoso OIB, Kostitsyna I, Ochalek ME, Olatunde TJ, Park IW, Sebastianelli FM, Taylor EM, Trinh GT and Cheung KC (2024) Ultralight, strong, and self-reprogrammable mechanical metamaterials. *Science Robotics* 9(86): eadi2746. DOI:10.1126/scirobotics.adi2746. URL https://www.science.org/doi/abs/10.1126/scirobotics.adi2746.

Gross R, Bonani M, Mondada F and Dorigo M (2006) Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics* 22(6): 1115–1130. DOI:10.1109/TRO.2006.882919.

Hayat A (2020) A framework for taxonomy and evaluation of self-reconfigurable robotic systems. *IEEE Access* .

Jing G, Tosun T, Yim M and Kress-Gazit H (2018) Accomplishing high-level tasks with modular robots. *Autonomous Robots* 42(7): 1337–1354. DOI:10.1007/s10514-018-9738-1. URL https://doi.org/10.1007/s10514-018-9738-1.

Jüttner A and Madarasi P (2018) Vf2++—an improved subgraph isomorphism algorithm. *Discrete Applied Mathematics* 242: 69–81. DOI:https://doi.org/10.1016/j.dam.2018.02.018. URL https://www.sciencedirect.com/science/article/pii/S0166218X18300829. Computational Advances in Combinatorial Optimization.

Kirby BT, Aksak B, Campbell JD, Hoburg JF, Mowry TC, Pillai P and Goldstein SC (2007) A modular robotic system using magnetic force effectors. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 2787–2793. DOI:10.1109/IROS.2007.4399444.

Liang G, Luo H, Li M, Qian H and Lam TL (2020) FreeBOT: A freeform modular self-reconfigurable robot with arbitrary connection point - design and implementation. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 6506–6513. DOI:10.1109/IROS45743.2020.9341129.

Liang G, Wu D, Tu Y and Lam TL (2024) Decoding modular reconfigurable robots: A survey on mechanisms and design. *The International Journal of Robotics Research* 0(0): 02783649241283847. DOI:10.1177/02783649241283847. URL https://doi.org/10.1177/02783649241283847.

Liang G, Zong L and Lam TL (2023) Disg: Driving-integrated spherical gear enables singularity-free full-range joint motion. *IEEE Transactions on Robotics* 39(6): 4464–4481. DOI:10.1109/TRO.2023.3311911.

Liu C and Yim M (2017) Configuration recognition with distributed information for modular robots. In: *IFRR International Symposium on Robotics Research*. Puerto Varas, Chile. DOI: 10.1007/978-3-030-28619-4_65.

Liu C and Yim M (2020) Configuration recognition with distributed information for modular robots. In: Amato NM, Hager G, Thomas S and Torres-Torriti M (eds.) *Robotics Research*.

Cham: Springer International Publishing. ISBN 978-3-030-28619-4, pp. 967–983.

Lueker GS and Booth KS (1979) A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM (JACM)* 26(2): 183–195.

Luo H and Lam TL (2022) Adaptive flow planning of modular spherical robot considering static gravity stability. *IEEE Robotics and Automation Letters* 7(2): 4228–4235.

Luo H and Lam TL (2023) Auto-optimizing connection planning method for chain-type modular self-reconfiguration robots. *IEEE Transactions on Robotics* 39(2): 1353–1372. DOI:10.1109/TRO.2022.3218992.

Malley M, Haghighat B, Houel L and Nagpal R (2020) Eciton robotica: Design and algorithms for an adaptive self-assembling soft robot collective. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 4565–4571. DOI:10.1109/ICRA40945.2020.9196565.

Neubert J, Rost A and Lipson H (2014) Self-soldering connectors for modular robots. *IEEE Transactions on Robotics* 30(6): 1344–1357. DOI:10.1109/TRO.2014.2344791.

Park M, Chitta S, Teichman A and Yim M (2008) Automatic configuration recognition methods in modular robots. *The International Journal of Robotics Research* 27(3-4): 403–421. DOI:10.1177/0278364907089350. URL https://doi.org/10.1177/0278364907089350.

Romanishin JW, Gilpin K, Claici S and Rus D (2015) 3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 1925–1932. DOI:10.1109/ICRA.2015.7139450.

S Sankhar Reddy Chennareddy AK Anita Agrawal (2017) Modular self-reconfigurable robotic systems: A survey on hardware architectures. *Journal of Robotics* 2017: 1–19. DOI:10.1155/2017/5013532.

Saintyves B, Spenko M and Jaeger HM (2024) A self-organizing robotic aggregate using solid and liquid-like collective states. *Science Robotics* 9(86): eadh4130. DOI:10.1126/scirobotics.adh4130. URL https://www.science.org/doi/abs/10.1126/scirobotics.adh4130.

Seo J, Paik J and Yim M (2019) Modular reconfigurable robotics. *Annual Review of Control, Robotics, and Autonomous Systems* 2(1): 63–88. DOI:10.1146/annurev-control-053018-023834. URL https://doi.org/10.1146/annurev-control-053018-023834.

Shimizu M, Mori T and Ishiguro A (2006) A development of a modular robot that enables adaptive reconfiguration. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 174–179. DOI:10.1109/IROS.2006.282216.

Shiu MC, Fu LC and Chia YJ (2010) Graph isomorphism testing method in a self-recognition velcro strap modular robot. In: *2010 5th IEEE Conference on Industrial Electronics and Applications*. pp. 222–227. DOI:10.1109/ICIEA.2010.5516792.

Spröwitz A, Pouya S, Bonardi S, Den Kieboom JV, Möckel R, Billard A, Dillenbourg P and Ijspeert AJ (2010) Roombots: Reconfigurable robots for adaptive furniture. *IEEE Computational Intelligence Magazine* 5(3): 20–32. DOI:10.1109/MCI.2010.937320.

Starke S, Hendrich N and Zhang J (2019) Memetic evolution for generic full-body inverse kinematics in robotics and animation. *IEEE Transactions on Evolutionary Computation* 23(3): 406–420. DOI:10.1109/TEVC.2018.2867601.

Swissler P and Rubenstein M (2020) FireAnt3D: a 3d self-climbing robot towards non-latticed robotic self-assembly. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 3340–3347. DOI:10.1109/IROS45743.2020.9341116.

Swissler P and Rubenstein M (2022) Reactivebuild: Environment-adaptive self-assembly of amorphous structures. In: Matsuno F, Azuma Si and Yamamoto M (eds.) *Distributed Autonomous Robotic Systems*. Cham: Springer International Publishing. ISBN 978-3-030-92790-5, pp. 363–375.

Swissler P and Rubenstein M (2023) Fireantv3: A modular self-reconfigurable robot toward free-form self-assembly using attach-anywhere continuous docks. *IEEE Robotics and Automation Letters* 8(8): 4911–4918. DOI:10.1109/LRA.2023.3290796.

Todorov E, Erez T and Tassa Y (2012) Mujoco: A physics engine for model-based control. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 5026–5033. DOI:10.1109/IROS.2012.6386109.

Tosun T, Davey J, Liu C and Yim M (2016) Design and characterization of the EP-Face connector. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 45–51. DOI:10.1109/IROS.2016.7759033.

Tu Y and Lam TL (2023) Configuration identification for a freeform modular self-reconfigurable robot - freesn. *IEEE Transactions on Robotics* 39(6): 4636–4652. DOI:10.1109/TRO.2023.3303848.

Tu Y, Liang G and Lam TL (2022) FreeSN: A freeform strut-node structured modular self-reconfigurable robot - design and implementation. In: *2022 International Conference on Robotics and Automation (ICRA)*. pp. 4239–4245. DOI:10.1109/ICRA46639.2022.9811583.

Veenstra J, Scheibner C, Brandenbourger M, Binysh J, Souslov A, Vitelli V and Coulais C (2025) Adaptive locomotion of active solids. *Nature* 639(8056): 935–941. DOI:10.1038/s41586-025-08646-3. URL https://doi.org/10.1038/s41586-025-08646-3.

Wu D, Liang G, Tu Y, Zong L and Lam TL (2024) Linear-time quasi-static stability detection for modular self-reconfigurable robots. *The International Journal of Robotics Research* 0(0): 02783649241286491. DOI:10.1177/02783649241286491. URL https://doi.org/10.1177/02783649241286491.

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

Zhao D and Lam TL (2022) SnailBot: A continuously dockable modular self-reconfigurable robot using rocker-bogie suspension. In: *2022 International Conference on Robotics and Automation (ICRA)*. pp. 4261–4267. DOI:10.1109/ICRA46639.2022.9811779.

Zhao D, Luo H, Tu Y, Meng C and Lam TL (2024) Snail-inspired robotic swarms: a hybrid connector drives collective adaptation in unstructured outdoor environments. *Nature Communications* 15(1): 3647. DOI:10.1038/s41467-024-47788-2. URL https:

//doi.org/10.1038/s41467-024-47788-2.

Zhu Y, Li G, Wang X and Cui X (2012) Automatic function-isomorphic configuration recognition and control for ubot modular self-reconfigurable robot. In: *2012 IEEE International Conference on Mechatronics and Automation*. pp. 451–456. DOI:10.1109/ICMA.2012.6282885.

Zong L, Liang G and Lam TL (2022) Kinematics modeling and control of spherical rolling contact joint and manipulator. *IEEE Transactions on Robotics* : 1–17DOI:10.1109/TRO.2022.3190790.