2021

Modular Robots Morphology Transformation And Task Execution

Chao Liu
University of Pennsylvania

Modular Robots Morphology Transformation And Task Execution

Abstract

Self-reconfigurable modular robots are composed of a small set of modules with uniform docking interfaces. Different from conventional robots that are custom-built and optimized for specific tasks, modular robots are able to adapt to many different activities and handle hardware and software failures by rearranging their components. This reconfiguration capability allows these systems to exist in a variety of morphologies, and the introduced flexibility enables self-reconfigurable modular robots to handle a much wider range of tasks, but also complicates the design, control, and planning.

This thesis considers a hierarchy framework to deploy modular robots in the real world: the robot first identifies its current morphology, then reconfigures itself into a new morphology if needed, and finally executes either manipulation or locomotion tasks. A reliable system architecture is necessary to handle a large number of modules. The number of possible morphologies constructed by modules increases exponentially as the number of modules grows, and these morphologies usually have many degrees of freedom with complex constraints. In this thesis, hardware platforms and several control methods and planning algorithms are developed to build this hierarchy framework leading to the system-level deployment of modular robots, including a hybrid modular robot (SMORES-EP) and a modular truss robot (VTT). Graph representations of modular robots are introduced as well as several algorithms for morphology identification. Efficient mobile-stylereconfiguration strategies are explored for hybrid modular robots, and a real-time planner based on optimal control is developed to perform dexterous manipulation tasks. For modular truss robots, configuration space is studied and a hybrid planning framework (sampling-based and search-based) is presented to handle reconfiguration activities. A non-impact rolling locomotion planner is then developed to drive an arbitrary truss robot in an environment.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Mechanical Engineering & Applied Mechanics

First Advisor

Mark Yim

Keywords

Control, Modular Robots, Motion Planning, Reconfiguration

Subject Categories

Robotics

MODULAR ROBOTS MORPHOLOGY TRANSFORMATION AND TASK EXECUTION

Chao Liu

A DISSERTATION

in

Mechanical Engineering and Applied Mechanics

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2021

Mark Yim, Supervisor of Dissertation
Director of GRASP Lab and Asa Whitney Professor of Mechanical Engineering and Applied Mechanics

---

Jennifer R. Lukes, Graduate Group Chairperson
Professor of Mechanical Engineering and Applied Mechanics

Dissertation Committee

Mark Yim, Director of GRASP Lab and Asa Whitney Professor of Mechanical Engineering and Applied Mechanics
Camillo J. Taylor, Associate Dean of Penn Engineering and Raymond S. Markowitz President's Distinguished Professor of Computer and Information Science
Cynthia Sung, Gabel Family Term Assistant Professor of Mechanical Engineering and Applied Mechanics

MODULAR ROBOTS MORPHOLOGY TRANSFORMATION

AND TASK EXECUTION

Dedicated to my loving family.

Acknowledgments

I would like to first thank my advisor, Professor Mark Yim, for the opportunity to work in this world-class laboratory on frontier robotics research. I gratefully acknowledge his support throughout my studies at Penn. In the past few years, I was fortunate to have the freedom to explore these exciting research topics with his encouragement and inspiration. His vision on research and education deeply affects my thoughts and inspires me to pursue more challenges.

I would also like to express my gratitude to the other members of my thesis committee, Professor Camillo J. Taylor and Professor Cynthia Sung. They gave me lots of insightful advice on my research and provided great comments on this thesis.

Many thanks would also go to my collaborators and the students that I worked with for their invaluable contributions to my research projects. I would also like to thank my lab-mates. I can always seek help from them and the discussions with them were immensely helpful for my work. I also truly appreciate the support from the MEAM and the GRASP staff.

I am indebted to my family for their continuous support and encouragement. My parents always allow me to make decisions and do whatever I am interested in, and provide countless support for my life. My grandparents worked very hard when they were young and I can always learn a lot from their attitude toward life. I would also like to thank my girlfriend Min Wang for her long-term support and encouragement with patience.

I shall extend my thanks to the National Science Foundation (NSF), the Air Force Office of Scientific Research, and the Army Research Lab.

ABSTRACT

MODULAR ROBOTS MORPHOLOGY TRANSFORMATION
AND TASK EXECUTION

Chao Liu

Mark Yim

Self-reconfigurable modular robots are composed of a small set of modules with uniform docking interfaces. Different from conventional robots that are custom-built and optimized for specific tasks, modular robots are able to adapt to many different activities and handle hardware and software failures by rearranging their components. This reconfiguration capability allows these systems to exist in a variety of morphologies, and the introduced flexibility enables self-reconfigurable modular robots to handle a much wider range of tasks, but also complicates the design, control, and planning.

This thesis considers a hierarchy framework to deploy modular robots in the real world: the robot first identifies its current morphology, then reconfigures itself into a new morphology if needed, and finally executes either manipulation or locomotion tasks. A reliable system architecture is necessary to handle a large number of modules. The number of possible morphologies constructed by modules increases exponentially as the number of modules grows, and these morphologies usually have many degrees of freedom with complex constraints. In this thesis, hardware platforms and several control methods and planning algorithms are developed to build this hierarchy framework leading to the system-level deployment of modular robots, including a hybrid modular robot (SMORES-EP) and a modular truss robot (VTT). Graph representations of modular robots are introduced as well as several algorithms for morphology identification. Efficient mobile-style reconfiguration strategies are explored for hybrid modular robots, and a real-time planner based on optimal control is developed to perform dexterous manipulation tasks. For modular truss robots, configuration space is studied and a hybrid planning framework (sampling-based and search-based) is presented to handle reconfiguration activities. A non-impact rolling locomotion planner is then developed to drive an arbitrary truss robot in an environment.

Contents

List of Tables

List of Figures

Part I

Preliminaries

Chapter 1

Introduction

Robots have been developed for years. There are a number of robotic systems in various morphologies. These robots are usually good at different applications or used in different scenarios. For example, many types of manipulators have been developed for factory automation, mobile robots can explore indoor environments efficiently, and snake robots and legged robots are very good at locomotion on uneven terrains. Given these systems which are specifically designed for some certain scenarios, it is still a challenge to deploy robots in the real world, which requires robots to respond to huge uncertainty, including the complexity of environments and the unknown nature of tasks that robots have to execute. It is desired to have robots change morphologies which is the key idea of self-reconfigurable modular robots.

Self-reconfigurable modular robots are composed of repeated building blocks (modules) from a small set of types. Often all modules have uniform docking interfaces that allow the transfer of mechanical forces and moments, electrical power, and communication throughout the robot [160]. These systems are able to adapt themselves to new circumstances or new tasks and recover from damage by changing their shapes. There are three key advantages for modular robots over fixed-morphology robots: versatility, robustness, and low cost. Self-reconfigurable modular robotic systems are potentially more adaptive due to their reconfiguration capability. Modules are interchangeable, so these systems can do self-repair by

replacing faulty parts. Modules are usually very simple and only have limited functionality, thus modular robotic systems are potentially cheaper with batch fabrication.

1.1 Motivation

Self-reconfigurable modular robots are able to deliberately change their own morphologies by rearranging the connectivity of their components so that they can adapt to new circumstances, perform new tasks, and recover from damage. For example, a cluster of modules can initially form a mobile vehicle and move on the ground, and then reconfigure into a snake morphology to traverse some narrow passage, and finally reconfigure into a manipulator on a base for some manipulation task. This is a promising goal for modular robots and it is highly dependent on three fundamental parts: *configuration recognition*, *morphology transformation*, and *task execution*.

Modular robotic systems consist of repeated building units resulting in a large number of different ways to arrange them. This characteristic enables these systems to form different morphologies when needed. This also brings about a fundamental requirement that the robot needs to be capable of recognizing its morphology in order to behave as expected and this is called *configuration recognition*. Note that the term *configuration* in reconfigurable robotics is commonly generalized to represent the connectivity of modules. For example, various locomotion behaviors using gait table control for Polypod [157] are programmed for different configurations, such as slinky locomotion for a biped walker, caterpillar locomotion and rolling-track locomotion for a loop [158]. A robot constructed from Polypod modules has to know how modules are arranged in order to apply the right locomotion policy. Configuration recognition is also shown to be useful when executing tasks using SMORES-EP for which a design library composed of multiple behaviors using different configurations is built [53]. In order to deploy a cluster of SMORES-EP modules to accomplish given tasks, the system has to identify itself and figure out the mapping from the hardware to the isomorphic configuration in the library in order to implement existing controllers.

Once a modular robotic system identifies its current configuration that is suitable for some given tasks, motion planning and control techniques have to be developed for *task ex-*

ecution. While this is a general problem for applications of robotic systems, some challenges still have to be taken into account regarding modular robots. One advantage of modular robots over traditional robotic systems with fixed morphologies is the flexibility: modular robots are able to address tasks in more scenarios by altering their methodologies or configurations, and the number of configurations increases exponentially as the number of modules increases. Hence, a universal model or a model generator is required for arbitrary configurations. In addition, a single module usually has one or more degrees of freedom (DOFs), and combining multiple modules to form a structure results in a high-dimensional system. Furthermore, multiple motion goals may share many DOFs which complicates the control and motion planning problem.

Other than the motion planning problem that is similar to traditional robotic systems, *morphology transformation* is in particular for modular robots. In order to respond to various scenarios, a modular robot is capable of reconfiguring its current morphology into another more suitable one for some given task which is known as reconfiguration motion planning. The reconfiguration capability is fundamental for modular robots to execute a variety of tasks, explore different environments, and address potential unknowns. In a reconfiguration process, motions of multiple modules are involved leading to the global change of the overall system. However, the planning is usually difficult due to a couple of reasons, such as the exponential increase of the number of possible configurations with respect to the number of modules and complex physical constraints. Furthermore executing reconfiguration actions is usually hard and time-consuming, and the efficiency of the reconfiguration process can be highly affected by the transformation strategy.

This thesis considers a hierarchy framework for modular robots composed of these three components which are distributed in three levels. Morphology identification and planning, as the top level of the framework, enable the robot to identify its current morphology and select a suitable morphology. The second level is the reconfiguration strategy that can output a sequence of reconfiguration actions if the robot needs to alter its morphology. The third level includes dexterous manipulation planners and locomotion controllers for task execution. In

this thesis, these components are mainly studied for two modular robots, *SMORES-EP* and *VTT*, and these work can be easily extended to other modular robots.

1.2 SMORES-EP

The SMORES-EP, standing for **S**elf-assembly **MO**dular **R**obot for **E**xtreme **S**hape-Shifting with **EP**-Face connector, is a hybrid modular robot. Each SMORES-EP module is in a cubic shape with four active rotational DOFs: pan, tilt, and left/right wheels. It has differential wheeled drive using its left and right wheels. There are four active connectors being equipped on a module leading to numerous ways to connect a set of modules. A SMORES-EP module is highly space-constrained. A low-cost position sensing solution is presented for SMORES-EP and the solution is shown to be reliable to estimate a DOF position leveraging a modified Kalman filter that can handle piece-wise nonlinear signals. The estimation and control of all DOFs are all running on a single microcontroller.

A number of SMORES-EP modules can form many different morphologies and these morphologies are modeled as graphs. Their topologies or configurations that describe how modules are connected are the fundamental to distinguish them, and these configurations should be invariant under different labeling of modules. The unique graph representation of a configuration is introduced and the robot can identify its current configuration by distributed messages transmitted among modules. An algorithm to quickly match two isomorphic configurations is also presented in order to utilize existing behaviors. Different from chain-style or lattice-style reconfiguration strategies, a cluster of SMORES-EP modules can reconfigure themselves in a mobile-style process in which modules can undock from the cluster in a carefully planned sequence and then dock with desired modules to form a new configuration. In addition, several separated modules can self-assemble into a desired morphology to enhance their capability that is similar to the collective behaviors of insects in nature. This mobile-style morphology transformation strategy is shown to be efficient and demonstrated on real hardware. Finally, dexterous manipulation planning using modular robots is formulated as a linearly constrained quadratic program to handle real-time control and planning simultaneously, and automatic locomotion control based on CPG control

networks has been studied well, such as [9, 57].

1.3 Variable Topology Truss

A **V**ariable **T**opology **T**russ (VTT) is a modular robot in truss structure. Similar to variable geometry truss (VGT) robots, a VTT is composed of edge modules or members with intersections being nodes. These edge modules have variable lengths to achieve *geometry reconfiguration*. Additionally, a VTT is able to self-reconfigure the attachment of members at the nodes called *topology reconfiguration*, changing its topology by merging or splitting nodes. Two separate nodes in the truss can dock to form one node which connects all of the involved members. Reversely, a single node with a sufficient amount of members can undock into a pair of nodes. The reconfiguration capability and the high extension ratio of the actuators used enable a VTT to change its shape and size dramatically. One example application using VTT structures aims to build a robot system that can be deployed into a disaster scenario. The robot is mobile by tumbling and can move into a building and reconfigure into large support structures to reinforce the building, shoring to prevent further collapse as first responders search for victims. This application exploits the inherent large strength-to-weight ratio of truss structures.

The configuration model and the kinematics model of an arbitrary VTT are derived. A motion controller as well as the corresponding hardware design are developed to guarantee the motion of a robot. Because of the topology reconfiguration capability, a VTT can achieve some highly dexterous motions. This behavior is similar to the DNA replication process in which topoisomerase can change the topology of DNA by cutting and resealing strands as tanglements form [18]. A VTT has the powerful reconfiguration capability but the reconfiguration planning is difficult. Reconfiguration actions happen at nodes and the configuration space of a single node is complicated. A fast algorithm to compute the obstacle region and the free space of a given node is provided so that the motion of a node can be planned easily. The geometry reconfiguration planning can be simplified by considering a group of nodes at each time with an efficient sampling strategy and a fast collision checking approach. Several complex physical constraints are also under consideration to ensure the feasibility of the

planned motions. The topology reconfiguration planning is solved by explicitly computing the free space of a node which is partitioned by its obstacle regions. A sequence of topology reconfiguration actions can be generated to navigate a node in its free space. Locomotion is necessary to deploy a VTT to perform tasks. An efficient rolling locomotion planner is presented for an arbitrary VTT while avoiding impact from the environment which may damage the mechanical components of the robot.

1.4 Thesis Contributions and Outline

It is a challenge for robots to work in the real world and one significant reason is that robots have to encounter a large number of unknown scenarios, including various environments and tasks. The compatibility between a robot and a scenario is highly dependent on the morphology of the robot. For example, a legged robot can be good at running on the ground efficiently but bad at climbing stairs, while a snake robot is able to climb stairs easily. In order to make robots more universal to handle more tasks, functional structures can be added. A vehicle can be equipped with an arm for manipulation tasks, or more limbs can be added to a legged robot to have extra DOFs to handle manipulation tasks. However, these added functional structures also have limitations. This thesis focuses on modular robots and considers the idea that robots can change their morphologies as needed to better address different scenarios. A self-reconfigurable modular robotic system is composed of a number of modules with uniform docking interfaces. These modules are capable of forming various morphologies and transforming from one morphology to another.

In order to deploy self-reconfigurable modular robots, a hierarchy framework is considered and studied. Modular robots can be in different morphologies and the morphology identification relying on the proposed graph representation is necessary. Then the morphology transformation strategy allows modules to reconfigure themselves into a desired morphology when needed for the current scenario. Finally, dexterous manipulation planners and automatic locomotion controllers are running to command the robot to execute tasks. This framework also heavily relies on a robust hardware platform, including a reliable hardware design and control, and a carefully designed software architecture. In particular,

a hybrid modular robot SMORES-EP and a modular truss robot VTT are studied in this thesis.

Chapter 3 introduces the hardware design of SMORES-EP and VTT, including actuating, docking interfaces, sensing, joint state estimation, and communication. A general software architecture for modular robots is presented in Chapter 4 and this architecture is implemented for hardware platforms and shown to provide good performance to handle various planning and control tasks. This is the foundation to apply modular robots in real scenarios.

For SMORES-EP, my work in hardware development and control is first presented in Chapter 5. A SMORES-EP module is a highly space-constraint system and customized position sensors are used. These sensors are manufactured manually and yield non-consistent measurement and performance. In order to provide reliable joint state estimation, a fast characterization process is presented and a Kalman filter with a simplified observation model is developed to handle the non-linearity issues. All the joint state estimators and controllers of a module are running on a low-cost microcontroller. In Chapter 6, an efficient docking strategy is presented. SMORES-EP is a hybrid modular robot that can reconfigure in all styles. Rather than reconfiguring in chain style or lattice style, a docking controller in mobile style is developed. This docking process can simplify the execution of a reconfiguration action and it is possible to execute multiple reconfiguration actions in parallel. Chapter 7 defines the graph representation of a modular robot morphology and develops some algorithms for morphology discovery and recognition. This work can be extended to other modular robots easily and is tested with the SMORES-EP system. Chapter 8 delves into the morphology transformation strategy based on the mobile-style docking process. The goal is to enable a cluster of modules to transform into a desired morphology in an efficient way. These modules may initially form a morphology and a set of undocking and docking action pairs are computed and executed to reconfigure into a new morphology, or initially they are separated on the ground and self-assemble into a structure to perform a task that cannot be solved by an individual. Chapter 9 introduces a manipulation planner for modular robots in

chain-style structures. A general kinematics model for arbitrary morphologies is developed and the planning and control problems are formulated as a linearly constrained quadratic program that can be solved in real time. The CKBot system is also used for demonstration.

A VTT is a modular truss robot. Some basic concepts, the configuration model and the kinematics model for an arbitrary VTT, and the hardware control framework are presented in Chapter 10. The additional capability of VTTs over other truss robots is that they can rearrange the connections of their members to reconfigure the *topology*. This topology reconfiguration capability allows more dexterous motions which is explored in Chapter 11. A new cell decomposition method is proposed with the shape of a VTT being considered so that efficient discrete motion actions can be applied. A VTT is a parallel robot and it is easier to control its shape by planning the motion of nodes. However, when moving nodes in a VTT, the collision between members is difficult to avoid. Chapter 12 delves into the configuration space for VTT nodes. A fast algorithm is presented to compute the configuration space of a given node so that the motion of this node can be easily planned. This chapter also shows that motions of multiple nodes are strongly coupled. Moving one node can significantly change the configuration space of other nodes in a truss robot. Hence it is a challenge to do reconfiguration planning for a VTT while satisfying all physical constraints. A motion planning framework to deal with this challenge is presented in Chapter 13. Chapter 14 solves the locomotion tasks for VTT robots. A non-impact rolling locomotion planner is developed to drive an arbitrary VTT in an environment.

The SMORES-EP system and the VTT system are the core platforms in this thesis, but the presented work here can be applied to other modular robots. The graph representation of the SMORES-EP robot is a general model for modular robots, and the morphology identification and recognition algorithm based on this model can also be used for other modular robots. The proposed reconfiguration strategy is valid for mobile-type or hybrid modular robots. And the dexterous manipulation planner is compatible with modular robots that can be in chain-style structures. The VTT has a similar structure with other modular parallel robots or truss robots but also has an additional topology reconfiguration capability.

Hence, the model and the control framework can also be applied to these robots. The configuration space algorithm and the geometry reconfiguration planner can be applied as well to change the shape of a truss robot. Similarly, the non-impact locomotion planner can be used to drive any truss robot.

Chapter 2

Overview of Related Work

This chapter provides an overview of the literature related to modular robots, including hardware, algorithms on control and planning.

2.1 Modular Robotic Systems

The performance of a self-reconfigurable modular robotic system is highly dependent on the hardware design and there are many challenges in order to overcome a lot of limiting factors, such as strength, precision, module dexterity and complexity, and so on. These systems are typically classified into three categories by the geometric arrangement of their units: *lattice-type*, *chain-type*, and *mobile-type*.

Lattice-type modular robotic systems are composed of modules that are arranged and connected nominally on a regular, three-dimensional pattern such as a cubical or hexagonal grid. Various systems have been developed, including Metamorphic [21], 3D Fracta [95], Telecubes [138], and Miche [35]. These architectures usually offer simpler reconfiguration as well as easily scaled representations as collision detection need only be considered locally. However, they are usually limited in application due to their motion constraints.

Chain-type modular robots, such as PolyBot [159], M-TRAN [96], and CKBot [163], consist of chains of modules which are usually in tree-like configurations and are more versatile as the serial chains of modules can act like articulated robot arms. Nevertheless, these systems are computationally more difficult to represent and analyze and more difficult

to control.

Mobile-type modular robots are able to use the environment to maneuver around, and can either hook up to form complex chains or lattices or form a number of smaller robots, like Millibots [98], Swarm-bots [38], Planar Catoms [62], and Kilobots [112]. They can usually simulate collective intelligence of creatures from nature to overcome limited capability of individuals by executing tasks as a group.

Hybrid-type modular robotic systems can be in all of these architectures, and thus are able to achieve all types of reconfiguration, locomotion, and manipulation activities. Examples of hybrid systems include SUPERBOT [119], SMORES [26], and Roombots [134].

In addition to these modular robotic systems, some modular robots are in truss structures which are commonly called *variable geometry trusses (VGT)* [91] and the truss members have variable lengths, such as TETROBOT [42], Odin [86], and LAR [148]. VTTs are similar to VGTs with additional capability to reconfigure nodes resulting in topology reconfiguration.

Hardware is important to demonstrate the potential of modular robots. A more comprehensive review of major modular robots can be found in [160] and [125]. In this thesis, SMORES-EP, CKBot, and VTT are the hardware platforms. SMORES-EP is a hybrid modular robot, CKBot is a chain-type modular robot, and VTT is a modular truss robot.

2.2 Control and Motion Planning

Robot behaviors heavily rely on robust controllers and motion planners. These behaviors are mainly aiming at manipulation of objects and locomotion. Gait tables have been commonly used to encode locomotion behaviors for known morphologies constructed by chain-type modular robots [158]. Other approaches to program locomotion gaits for chain-type structures include hormone-based gait control [120], role-based gait control [136], phase automata model [165], and CPG-based locomotion control [9, 57, 133]. These approaches can be implemented in a distributed way. Compared with chain-type modular robots, lattice-type modular robots perform locomotion by cluster-flow in which modules continuously execute detachment and attachment actions locally to achieve the global motion of the cluster [30, 166]. Redundant systems are common in modular robotics. Manipulation behaviors usually

involve the control and planning of a hyper-redundant arm built by several modules [22, 87, 127]. Some frameworks are well developed to address this type of motion planning problem. However, some difficulties need to be considered for modular robots. More details on existing manipulation strategies can be found in Chapter 9.

A significant capability of self-reconfigurable modular robots that differentiates them from normal robotic systems is the morphology transformation. These robots are able to execute reconfiguration actions to alter the connections among modules resulting in the change of their overall morphologies. The reconfiguration process of lattice-type robots can be simplified through discretization and the control actions are also easy to execute, such as the reconfiguration planning for Telecubes [150]. It is more difficult to do reconfiguration for chain-style robots in which closed kinematic chains have to be formed [124]. Control and planning schemes of multiple rigid bodies in an environment with complex obstacles are usually involved in the process. Efforts for mobile-style reconfiguration planning have mainly focused on self-assembly that several separated modules can form a target structure efficiently. Some simple mobile-style self-reconfiguration behaviors using SMORES-EP are shown in [25] and only one or two modules were involved in each reconfiguration process. A detailed review on the reconfiguration planning is in Chapter 8.

Truss modular robots have a different type of architecture. Previous truss robots can metamorphose through changing the lengths of truss members, and the VTT system can additionally rearrange the connections among truss members. Related work on shape control, locomotion, and reconfiguration for truss modular robots can be found in Chapter 13 and Chapter 14.

Part II

System Design and Architecture

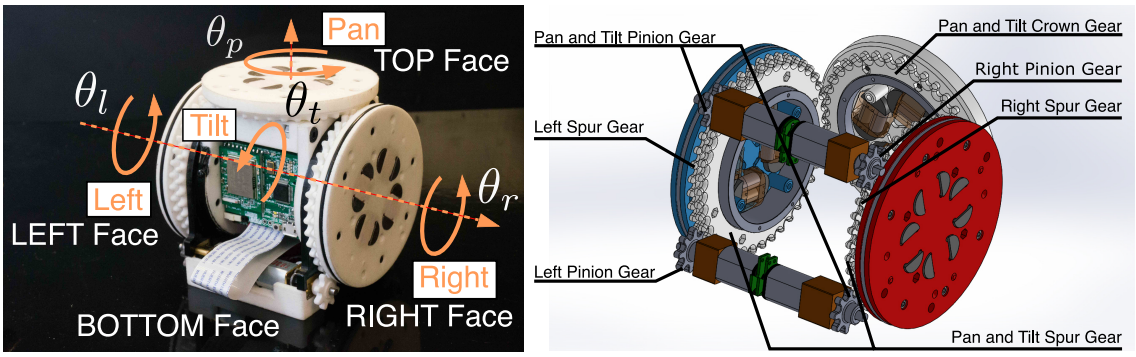
Chapter 3

Modular Robot Hardware

SMORES-EP and VTT are the core hardware systems in my work. This chapter provides an overview of these two systems and my contributions to their development.

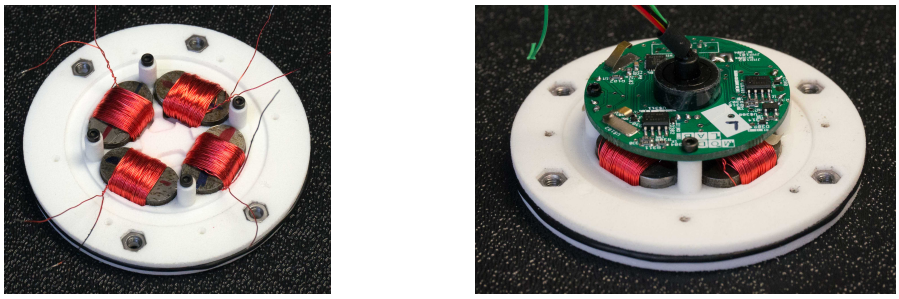
3.1 SMORES-EP

The SMORES-EP modular robot is in hybrid architecture that can behave in all three types of reconfiguration, locomotion, and manipulation activities (chain-type, lattice-type, and mobile-type). The conceptual design of this robot is first introduced in [26]. The system can emulate many past modular robotic systems by rearranging modules in ways that replicate their kinematics. A SMORES-EP module has four active rotational DOFs and four connectors shown in Figure 3.1a. The motion of a module is actuated through a gear train shown in Figure 3.1b. The left wheel and the right wheel can continuously rotate and are driven by DC brushed motors with 298:1 gearboxes independently. The pan and tilt DOFs are coupled and driven by two DC brushed motors with 1000:1 gearboxes driving three gears in a differential configuration where pan DOF can continuously rotate to produce a twist motion and tilt DOF is limited to $\pm 90^\circ$ to produce a bending joint. A SMORES-EP module has an 80 mm cube-like form factor with four EP-Face connectors [143] equipped on its four sides (LEFT, RIGHT, TOP, and BOTTOM) for docking. Each face can form a strong connection with other modules by the use of four EP magnets arranged in a ring, with south poles counterclockwise of north shown in Figure 3.2. The ring arrangement



(a) (b)

Figure 3.1: (a) A SMORES-EP module. (b) The SMORES-EP module driving mechanism.



(a) (b)

Figure 3.2: (a) Internal view of the magnets in EP-Face. (b) Internal view of the EP-Face with circuit board.

of the magnets makes the connector able to connect in four possible configurations. Also connected EP-Faces are able to exchange data through the magnetic coupling of connected EP-magnets which are capable of UART serial communication [143]. These four faces can be independently assembled and a module can be easily assembled using eight screws. All four motors are attached to the bottom face, and the space in the center of the module is occupied by circuit boards, batteries, and wires. A single module can lift four SMORES-EP modules held out in a cantilever due to the limitations of motors and connector forces.

The electronics in a SMORES-EP module is distributed among multiple bodies. Each EP-Face has its own electronics to control its four EP magnets (Figure 3.2b). These four magnets are driven by an array of five half-H bridges that is capable of sourcing 6 A current. The circuit is shown in Figure 3.3 allowing bi-directional drive of each magnet (activation

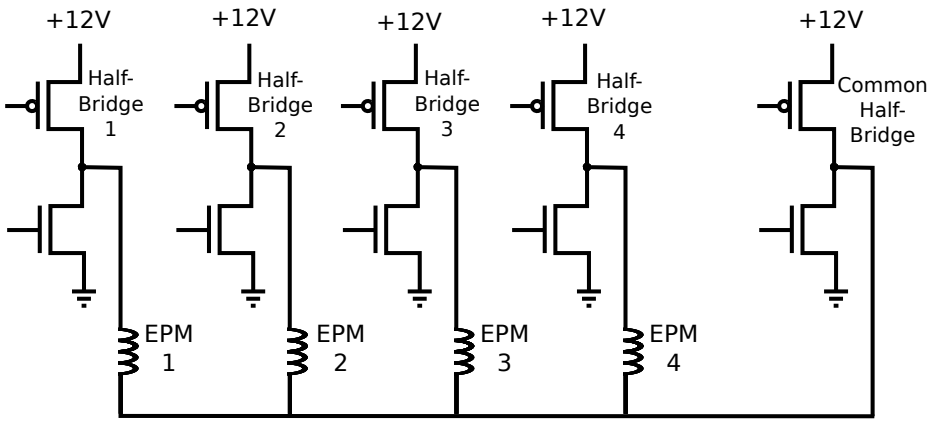
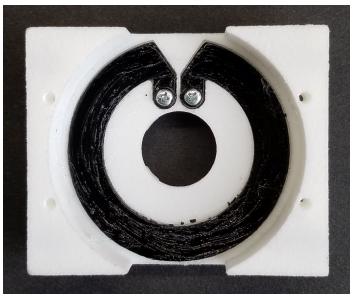


Figure 3.3: The circuit for driving EP magnets.

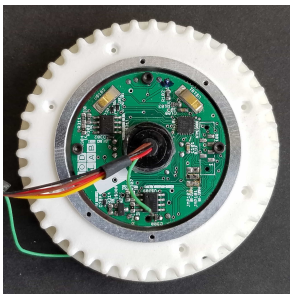
and deactivation) in a sequential manner. This circuit is controlled by an ATmega168A microcontroller running at 8 MHz. Three pulses of length 3 ms are applied at intervals of 3 ms when activating or deactivating a magnet. Two mated EP-Faces are able to exchange data through the magnetic coupling of connected EP-magnets: when a coil is pulsed, the generated magnetic field also flows through the core of the connected coil, and the changing field generates a voltage across that coil. Similar capabilities have been demonstrated in [34].

A low-cost position sensing solution is applied for every articulated joint in a SMORES-EP module [79]. An inexpensive carbon-embedded polymer spray paint is used to generate a resistive track surface so that one or more wipers can measure the voltage at the point of contact with the resistive strip [144]. The three continuously-rotating faces (LEFT, RIGHT, and TOP) have *Wheel PaintPot*s with circular tracks and two wipers offset parallel to the axis of rotation are mounted on their circuit boards (Figure 3.4), while the central hinge has a *Tilt PaintPot* that covers 180° arc with a wiper offset normal to the axis of rotation installed on the BOTTOM Face (Figure 3.5). The measurement from a PaintPot encoder of each joint is also gathered by the microcontroller of the corresponding face.

A 32 bit microcontroller STM32F303 running at 72 MHz is used as the central onboard processor of a module. It communicates with four faces via I^2C to handle all EP-Face connector activities (activation or deactivation), all DOFs' position estimation and control (gathering measurements of all DOFs and sending commands to motors), and Wi-Fi com-



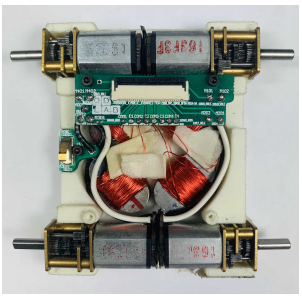
(a)



(b)

Figure 3.4: (a) A wheel PaintPot sensor is installed in a chassis. (b) Two wipers (Harwin S1791-42) are mounted on a circuit board at a 50° angle to one another fixed to the wheel.

(a)



(b)

Figure 3.5: (a) A tilt PaintPot sensor is installed on a chassis. (b) A single wiper installed on the base of a SMORES-EP module contacts the track.

munication with the central computer (UDP protocol) via a Wi-Fi module (TI CC3000 chip).

3.2 Variable Topology Truss

The variable topology truss, or VTT, is a class of self-reconfigurable modular robots in truss framework composed of edge modules or members (beam elements in a truss) and nodes (intersections of members). An example is shown in Figure 3.6. Each module is a linear actuator with two passive chainable spherical joints that can be docked with other modules (Figure 3.7). VTTs have similar structures with variable geometry trusses (VGTs) [91] but with additional topology reconfiguration capability. While VGTs only have control over the shape or geometry of trusses, VTTs are able to rearrange the connections among members by splitting or merging nodes.

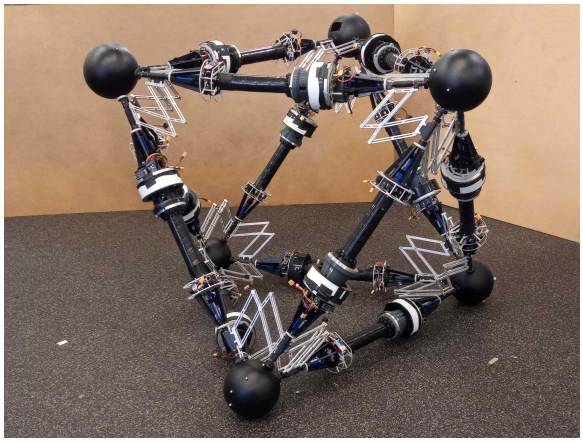
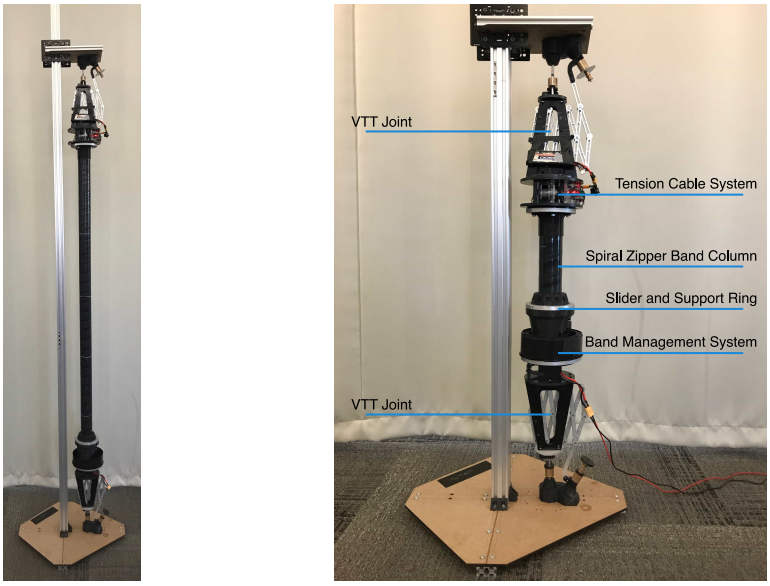


Figure 3.6: The hardware of a VTT in octahedron configuration is composed of twelve members. Note that at least eighteen members are required for topology reconfiguration [132].



(a) (b)

Figure 3.7: (a) An extended VTT edge module. (b) A shorter configuration with components marked.

The Spiral Zipper joint [24] is operated as both a prismatic actuator and a structural component in an edge module. It can achieve a high extension ratio as well as form a high strength-to-weight ratio column to support a large compression load. In the current version, the joint is driven by a brushed DC motor with a Planetary gear box via a friction drive mechanism. The friction drive has an interference fit with the band column and uses friction

to extend or retract the band column. The addition of a support ring is incorporated into the drive system in order to enable operation when the column is subjected to a bending load. The drawback of the Spiral Zipper joint is that it cannot withstand high tension. An elastic tension cable system (Figure 3.8) is applied in order to overcome this difficulty. This system can apply constant tension to the Spiral Zipper band and this fixed tension is configurable.

The electronics of each module is separated into two places for tension cable control and Spiral Zipper control respectively. The tension cable system (Figure 3.8) is located at the opposite end of the edge module from the Spiral Zipper friction drive. A torsion spring acts as a torque sensor. The cable is stored in a spool inside the tension cable module. The spool is connected in proportional to the tension in the cable. A pair of encoders measure this twist by measuring the rotational offset in the spool and drive motor, providing the input for feedback control. Controlling this twist allows the cable to extend and contract along with the zipper while maintaining a fixed tension. An ESP-32 based development board with an OLED screen is used to control the tension applied to the cable and also communicate with a central computer. The Spiral Zipper driving board is located underneath the band management system shown in Figure 3.9. A NUCLEO-F303K8 development board running

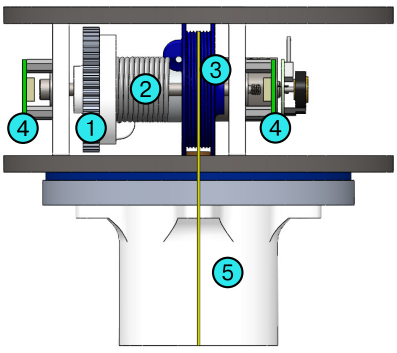
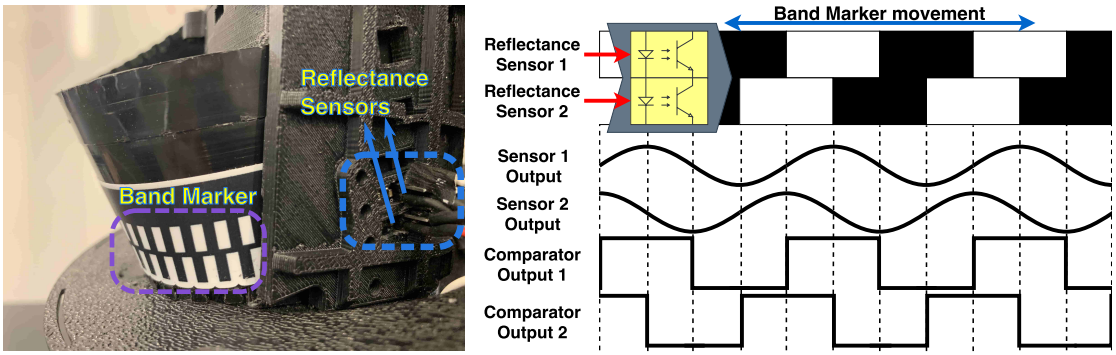


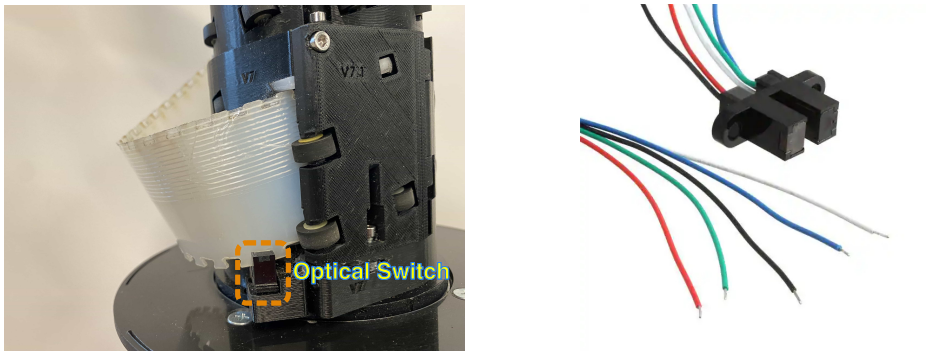
Figure 3.8: The series elastic tension cable system: The drive gear (1) is connected to a torsion spring (2), which is connected to the spool (3). The yellow line is the path of the cable, which continues down through the band column and attaches to the friction wheel. The drive gear and spool positions are measured by encoders (4). The cap (5) is the attachment point to the zipper.

Figure 3.9: The Spiral Zipper driving board is located underneath the band management system. This board measures the edge module length, drive the Spiral Zipper joint, and communicate with the central computer over Wi-Fi.



(a) (b)

Figure 3.10: (a) The band marker strip is attached to the band to act as a quadrature encoder. (b) Quadrature encoder signal is generated after processing the raw signals from two reflectance sensors by a comparator.



(a) (b)

Figure 3.11: (a) An optical switch is installed on the band management system to count band teeth. (b) The optical switch used in VTTs.

at 72 MHz controls the DC motor and is able to measure the length of the member by reading quadrature encoder signals (Figure 3.10b) generated from two alternating black and white strips attached to the body of the Spiral Zipper shown in Figure 3.10a. This functionality has been demonstrated in [72]. The length of a member can also be measured by an optical switch combined with the DC motor quadrature encoder shown in Figure 3.11a. The optical switch (Figure 3.11b) can count the number of band teeth going through its slot and the moving direction is determined by the DC motor and can be derived from the quadrature encoder of the motor. In the current experiment setup, every member length is also measured by a VICON motion capture system for initialization. Another ESP-32 based Wi-Fi module handles the communication with the central computer and commands the NUCLEO-F303K8 to control the Spiral Zipper joint.

Chapter 4

Software Architecture

This chapter presents a general software architecture for modular robots, the fundamental to control hardware platforms. This architecture is implemented for SMORES-EP, CKBot, and VTT with minor modifications for each system.

4.1 Hybrid Architecture

A modular robotic system is usually composed of a number of modules, and these modules not just need to communicate with each other but also execute physical interaction, such as docking and undocking. A robust and efficient architecture is required when executing tasks on a hardware platform in real-time. A simulator is also helpful when exploring planning and control algorithms. Although the gap between the simulator and the reality is a challenge to overcome, it is tedious to regenerate programs that work only in simulation for real robots. This also requires a proper design of the software architecture so that the sim-to-real transfer can be derived easily.

In this thesis, the control architecture for modular robots is hybrid: distributed at low-level actions and centralized at high-level planning. The framework has to handle a number of modules: each module has its own processor to handle hardware-level control and communication, and a central computer is used for high-level control and planning. Simulation tools are also provided and the communication with a simulator is in the same way with real hardware platforms, so there is no need to regenerate programs when switching from

simulation to real hardware platforms. This design of software architectures makes use of distributed computing power and also achieves efficiency through centralized information processing.

4.2 Architecture Design

The general software architecture is shown in Figure 4.1. The core of the architecture is `Server` that communicates with `User Interface`, `Parameter Server`, and robots (`Simulator` and `Hardware`).

4.2.1 Configuration

The robot setup is defined by users and Server can get these information from Parameter Server.

1. **Configuration** — Configuration description parameters are provided to Server in order to generate models for the defined robot. For modular robots, these parameters may include a graph representation and module labels.

2. **Constraints** — Constraints related to the planning and control problems are defined and transmitted to Server, such as hardware limits, workspace boundary, and obstacles.

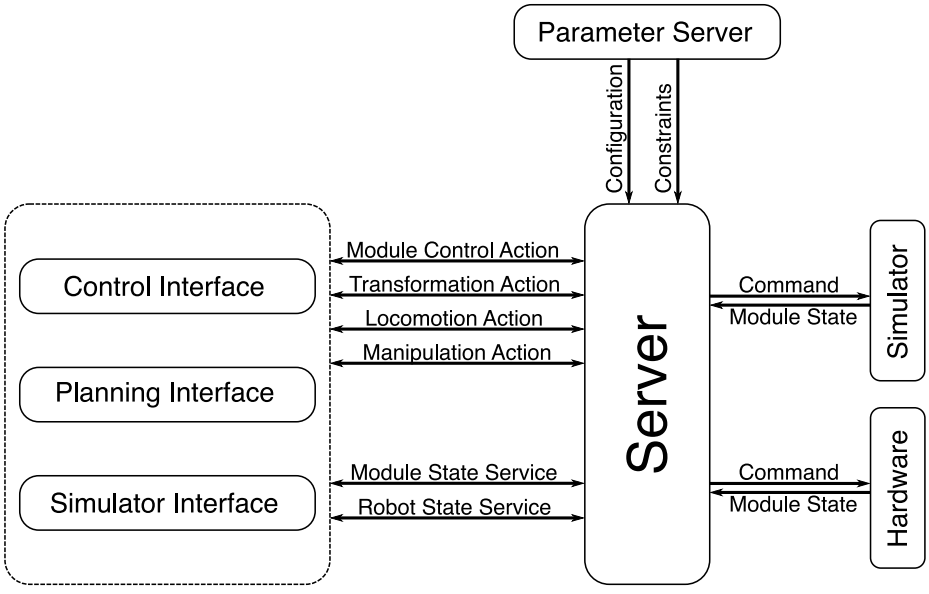


Figure 4.1: The general software architecture for modular robots.

4.2.2 Robot Interface

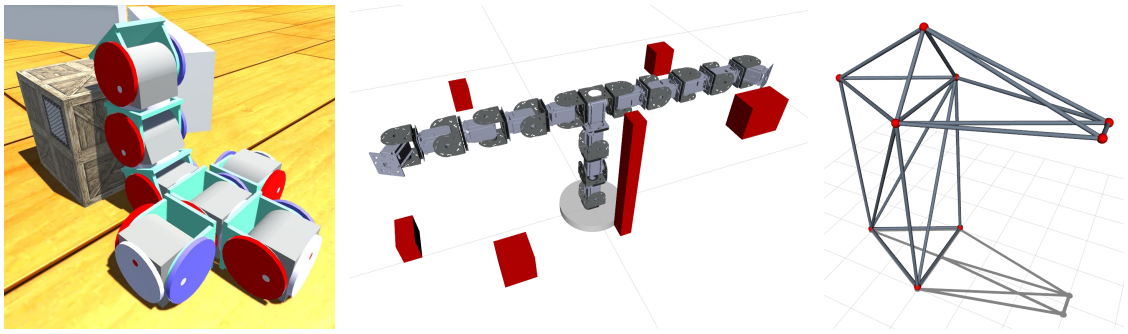
Server communicates with either a simulator or a hardware platform to execute control commands and track module states. This communication should be fast enough in order to achieve real-time control, such as a trajectory following task for a chain of modules. This communication is also important to deal with hardware failures.

4.2.3 User Interface

Users can access several actions and services related to control, planning, and options for simulation. Core actions and services include Module Control Action, Transformation Action, Locomotion Action, Manipulation Action, Module State Service, and Robot State Service. Module Control Action provides the interface to execute module-level commands, such as moving a joint. Transformation Action, Locomotion Action, and Manipulation Action provide high-level control and planning interfaces for a cluster of modules. Users can also access information related to an individual module by Module State Service, or access information of the whole structure built by modules through Robot State Service.

4.2.4 Implementation

This architecture is applied to SMORES-EP, CKBot, and VTT, and implemented in Robot Operating System (ROS) [141]. Server is composed of several ROS nodes. The robot configuration and constraints are fully defined in a YAML file and Server is able to get and parse these parameters from the ROS parameter server easily. Then Server creates the communication with hardware, and also initializes the robot and the workspace in a simulator. Rviz (ROS visualizer) is used as the visualization tool for CKBot and VTT (Figure 4.2b and Figure 4.2c). For CKBot, the URDF of the robot configuration is generated automatically first from the corresponding YAML file so that Rviz can display the robot. For SMORES-EP, a simulator [53] is developed on Unity platform [147] (Figure 4.2a). The communication between ROS and Unity is achieved by `rosbridge` package [111]. All hardware-level control actions are implemented locally on each module, such as estimation and control of each articulated joint in a SMORES-EP module, the state of an EP-Face connector, velocity



(a) (b) (c)

Figure 4.2: (a) A SMORES-EP configuration is created in Unity environment. (b) A dual-arm system is created by CKBot UBar modules in Rviz with a cluttered environment. (c) A VTT configuration is created in Rviz.

control of a VTT edge module, and so forth. Several high-level control and planning interfaces are constructed for users, such as the manipulation planners for SMORES-EP and CKBot, the locomotion controller for VTT, and the morphology transformation planners for SMORES-EP and VTT.

Part III

SMORES-EP

Introduction

This part presents the research work in order to deploy SMORES-EP, a newly developed hybrid modular robot. A SMORES-EP module is highly space-constrained with four DOFs and four connectors so that the system has high flexibility that can be applied in a large range of scenarios. In order to achieve a fully autonomous modular robotic system, several core components are necessary: a reliable hardware control strategy, an algorithm to allow the robot to transform its morphology as needed, and a framework to enable the robot to execute tasks, including manipulation and locomotion.

Chapter 5 presents the complete local control and estimation solution for SMORES-EP modules. The control of every DOF is straightforward as long as reliable position feedback can be guaranteed. The difficulty of integrating reliable position sensors is addressed by using low-cost absolute encoders. These encoders can be manually manufactured easily but require extra effort in order to perform well.

Chapter 6 introduces the docking controller that is the fundamental for SMORES-EP to transform from one morphology into another one. The docking process is divided into three phases which can be executed continuously by the corresponding motion controllers. With the efficiency of the hybrid architecture in Chapter 4, multiple docking action queries can be handled simultaneously.

Chapter 7 defines the model to describe an arbitrary SMORES-EP morphology that can also be extended to other modular robots, such as CKBot. Mathematical analysis and several algorithms that can be applied for automatic configuration recognition are provided, and these tools are prerequisites to building high-level motion planning frameworks.

Chapter 8 delves into the morphology transformation problem for SMORES-EP and other hybrid or mobile-type modular robots. This mobile-style transformation strategy is shown to be an easy and efficient solution for such systems. Two categories of transformation — self-assembly and self-reconfiguration — are discussed. Multiple modules can execute reconfiguration actions in parallel in a cooperative manner.

Chapter 9 deals with the manipulation planning using modular robots. Modules can form a large variety of morphologies which are inherently high-DOF systems. A general kinematics model is developed for arbitrary morphologies and the manipulation planning problem is formulated as a quadratic program with a novel way to embed all motion constraints as linear constraints. This framework can also be applied to single-arm or dual-arm manipulation and whole-body manipulation.

Chapter 5

Module Control

This chapter presents the control framework for a SMORES-EP module, including a low-cost position sensing technique, a position estimation method, and a controller to control its four joints. This chapter excerpts heavily from [79]. Credit is due to co-author Tarik Tosun.

A SMORES-EP module is a highly space-constrained system equipped with four DOFs driven by DC brushed motors and four EP-Face connectors. It is important to have accurate position sensing for state estimation and control in robotics. Reliable and accurate position sensors are usually expensive and difficult to customize. Incorporating them into systems that have very tight volume constraints such as modular robots is particularly difficult. In SMORES-EP modules, PaintPots are used for position sensing. PaintPots are low-cost, reliable, and highly customizable position sensors [144], but their performance is highly dependent on the manufacturing and calibration process. A complete solution is presented for the use of PaintPots in a variety of sensing modalities including manufacturing, characterization, and estimation for SMORES-EP modules. A Kalman filter with a simplified observation model is developed to deal with the piece-wise non-linearity issues that result in the use of low-cost microcontrollers. With this technique, it is shown that each DOF of a SMORES-EP module can be controlled using a feedback controller easily and precisely.

5.1 Introduction

Robotics control process usually requires a robust controller as well as accurate state sensing for feedback. For a SMORES-EP module, hardware control involves a position controller for all four DOFs shown in Figure 3.1. Every DOF needs to be equipped with an absolute position sensor to report its current state. This is challenging for SMORES-EP that is a highly space-constrained robotic system. Commercial position sensors are available for many applications. However, the form factor often presents a challenge to use these commercial off-the-shelf sensors. Customizable position sensors give more flexibility to fit tight space constraints.

Servos have been used in many modular robotic systems, such as Conro [16] and CK-Bot [163]. Servo motors have built-in position control circuits. Some modular robotic systems, such as 3D Fracta [95], M-TRAN III [65], and SUPERBOT [119], use rotary potentiometers for position feedback. Optical or hall-effector sensor encoders are used in PolyBot [159], Crystalline [114], and ATRON [54] to report position information. These devices are usually too large for highly space-constrained systems.

PaintPots are highly customizable and low-cost position sensors that can be easily manufactured by widely accessible materials (spray paint and plastic sheets) and tools (laser cutters or scissors) [144]. The sensors can exist in different forms in terms of size, shape, and surface curvature. Thus, these sensors can be easily integrated with well designed parts or systems. The sensing performance and cost of PaintPot sensors make them competitive with commercial potentiometers yet the customizability enables the use in situations which are not possible with commercial potentiometers [144].

Two different designs of PaintPot sensors are used in SMORES-EP modules. In each SMORES-EP module, there are four DOFs requiring position sensing shown in Figure 3.1a: three continuously rotating joints (*LEFT DOF*, *RIGHT DOF*, *PAN DOF*) and one bending joint with a 180° range of motion (*TILT DOF*). Conductive spray paint is used to generate a resistive track surface. The manufacturing process is easy enough for a person to make a sensor quickly, but often does not yield consistent measurement and performance. The

terminal-to-terminal resistance can vary over a large range depending on the thickness of the paint with a non-linear output. This fact complicates the position estimation for every DOF.

For state estimation, stochastic techniques based on the probabilistic assumptions of the uncertainties in the system are widely applied. For linear systems, the Kalman filter [56] has been shown to be a reliable approach where uncertain parts in systems are assumed to have a particular probability distribution, usually Gaussian. Extensions including the extended Kalman filter (EKF) and the unscented Kalman filter (UKF) [151] have been developed for nonlinear systems. For SMORES-EP DOF state estimation, a new Kalman filter with a simpler observation model considering the non-linearity of PaintPot sensors is developed. A complete and convenient calibration process is developed to precisely characterize each position sensor quickly. Four estimators can run on a 72 MHz microcontroller at the same time to track the states of all DOFs in a SMORES-EP module. The reported position information can be used directly by the onboard feedback controller to drive four DOFs with good performance.

5.2 PaintPot Sensor in SMORES-EP

The tight space requirement of a SMORES-EP module made it very difficult to incorporate off-the-shelf position encoders into the design, motivating the development of custom PaintPot encoders that occupy very little space within the robot. This section provides an overview of the manufacturing techniques used to create PaintPots, the fundamental material resolution of the sensors, and the design of the PaintPots used in SMORES-EP. More details can be found in [144].

5.2.1 Manufacturing Overview

A potentiometer is a three-terminal resistor with a sliding or rotating contact (or wiper) that functions as a voltage divider (Figure 5.1). The resistive track surface of a PaintPot encoder is made from conductive spray paint on a plastic track substrate. The PaintPots in SMORES-EP use three coats of MG Chemicals Total Ground conductive paint [145]

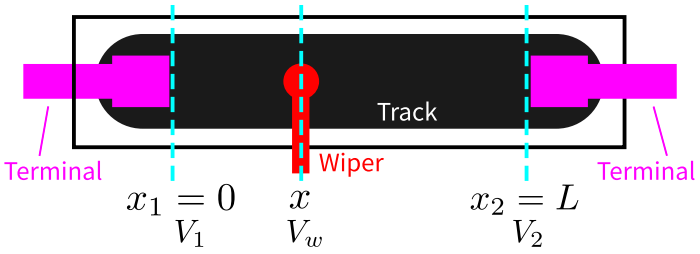


Figure 5.1: A potentiometer has three terminals: two fixed electrical terminals at the resistive track ends and one electrical contact (wiper) that can move along the track surface.

(a) (b)

Figure 5.2: A bead of conductive paint applied beneath the screw head forms a good electrical connection with the track.

sprayed onto acrylonitrile butadiene styrene (ABS) plastic sheets. Three coats of paint are applied, with five minutes of drying time between coats, following the painting guidelines in the datasheet. ABS plastic can be cut to precise shapes in a laser cutter and forms an ideal substrate for the paint, which readily bonds to the surface [145]. Electric terminals are created by mounting zinc-coated screws at the ends of the resistive strip. Applying paint above and beneath the screws creates an electrical connection between the track surface and the screw (Figure 5.2). Leaded solder adheres to zinc-coated screws, allowing wires to be attached and detached.

Each PaintPot uses one or more wipers to measure the voltage at the point of contact with the resistive strip. Wipers with high contact pressure should be avoided, as they may scratch the paint. A larger contact surface area also reduces contact resistance, improving signal quality. The Harwin S1791-42 EMI Shield Finger Contact [115] is used in SMORES-EP modules. The wiper is a 4 mm high gold-plated tin spring contact with a 1.45 mm×2.05 mm contact area, and a contact force of 1 N (mounted on a PCB at a 3 mm working height).

5.2.2 Performance Characteristics

The fundamental material resolution of the resistive track was determined by characterizing the signal-to-noise ratio of the measured voltage on small length scales. Based on the tests, the fundamental resolution of the track material is $8.63 \pm 0.126\,\mu$m, which is of the same order of magnitude as some commercially available high-precision potentiometers. In SMORES-EP, the limiting factor in resolution is the analog-to-digital conversion bit depth (10 bit, or $84\,\mu$m); to reach the material limit, 14 bit of analog-to-digital conversion depth would be required.

5.2.3 Wheel and Tilt PaintPots in SMORES-EP

Each of the four articulated joints in a SMORES-EP module is equipped with a PaintPot, which provides absolute position encoding.

Wheel PaintPots

The wheel PaintPots, shown in Figure 3.4a, have a circular track and two wiper contacts, allowing continuous rotation and providing position information over the full 360° range of the left, right, and pan joints. The annular geometry allows a slip ring to fit through the center. Tabs on the track extend into the center of the circle to provide space for the terminal contacts. The V-shaped gap provides enough space for the wipers to pass from one side of the track to the other without contacting both simultaneously (which would cause a short circuit). The two wipers are mounted on a PCB above the track at a 50° angle to one another (Figure 3.4b); this configuration ensures that at least one wiper contacts the track at all times.

Tracks are cut in batches in a laser cutter. To facilitate easy mounting, a layer of double-sided adhesive is applied to the back of the ABS sheet before cutting. After cutting, three coats of paint are applied, and strips are allowed to dry for 24 hours. Strips are mounted in a mated groove in a 3D-printed chassis as shown in Figure 3.4a. The chassis has a raised triangular feature that mates with the gap in the strip so that the wipers remain at the same level as they pass through the gap region. Zinc-coated screws are used for electrical

terminals. The measured terminal-to-terminal resistance of wheel PaintPots ranges from 2 kΩ to 20 kΩ, depending on the thickness of the paint. Before use, a coat of petroleum-based grease is applied to the track surface.

Tilt PaintPots

The tilt PaintPots, shown in Figure 3.5a, have tracks with cylindrical curvature about their axis of rotation. A single wiper contacts the track and measures position through the full 180° of motion of the tilt joint (Figure 3.5b). The track geometry of the tilt PaintPot makes very efficient use of space inside the SMORES-EP module; to our knowledge, no off-the-shelf potentiometers replicate this unusual non-planar shape.

Tilt PaintPots have the same ABS/adhesive substrate as wheel PaintPots, and similar screw contacts. They are mounted to the 3D printed chassis before painting, allowing them to be painted in their final curved shape. This is preferable to painting flat and then bending: bending the paint after it has dried causes cracks to form, increases the resistance (three orders of magnitude), and causes a non-smooth variation of voltage along the length of the track. The terminal-to-terminal resistances of the tilt PaintPots range from 3 kΩ to 10 kΩ.

5.2.4 Cost

The PaintPots used in SMORES-EP are inexpensive. The wipers are available from Digikey.com for $0.35 USD in quantities of 100. A 12 oz MG Chemicals Total Ground spray paint can be purchased from Amazon.com for $16 USD, and 0.79 mm ABS sheets can be purchased from McMaster.com for $3.70 USD per square foot. Based on these, materials for wheel PaintPots cost $1.05 USD and tilt PaintPots cost $0.70 USD. In order to build SMORES-EP modules with quality control testing [144], we yield about 75% of our wheel PaintPots and 90% of our tilt PaintPots, making the effective materials costs $1.40 USD and $0.78 USD respectively.

5.3 Sensor Characterization

Potentiometers used as voltage dividers typically model the input position as having a linear relationship with the output voltage. Close adherence to the linear model has to be

achieved by ensuring that the resistance between two points along the track is constant, which requires uniform geometry, thickness, and material properties of the track. This is difficult for PaintPots which are manually spray-painted. In order to obtain accurate position control on all DOFs of a SMORES-EP module, a calibration process is needed to characterize the performance of the particular PaintPots installed.

One terminal of a wheel PaintPot is connected with 3.3 V and the other terminal is connected with ground. Two wipers can contact the track and report current voltage (V_0 and V_1) in the form of two 10 bit analog-to-digital conversion values ranging from 0 to 1023, and wheel position $\theta = 0$ rad is shown in Figure 5.3 and the whole range of θ is from $-\pi$ rad to π rad. When a wiper contacts on or around the V-shape gap, the voltage value is not usable. This is the reason for having two wipers, to enable sensing the full 360° range. So, V_0 should be ignored when θ is in the range from $\frac{2}{3}\pi$ rad to $\frac{5}{6}\pi$ rad and V_1 should be ignored when θ is in the range from $-\frac{5}{6}\pi$ rad to $-\frac{2}{3}\pi$ rad.

Similar to wheel PaintPots, the tilt PaintPot is also powered between ground and 3.3 V with one single wiper contacting the track all the time. The voltage V_0 from the voltage divider goes through a 10 bit analog-to-digital conversion (with range from 0 to 1023). When the tilt position $\theta = 0$ rad, the wiper is positioned in the middle of the track as shown in Figure 5.4.

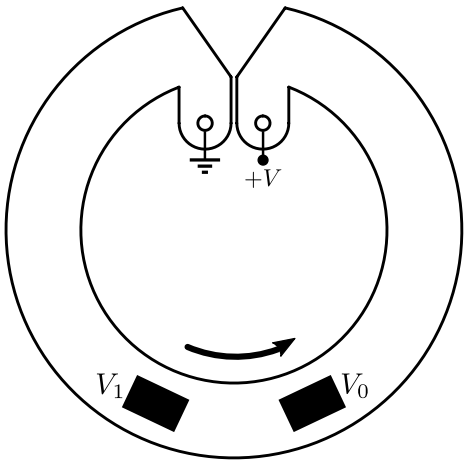


Figure 5.3: When the wheel position $\theta = 0$ rad, two wipers are contacting the track symmetrically to the middle location of it and θ ranges from $-\pi$ rad to π rad.

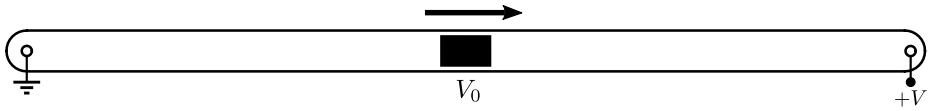
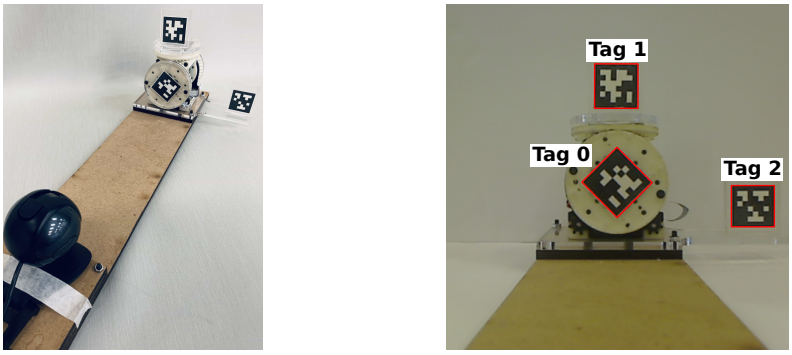


Figure 5.4: When the TILT DOF position $\theta = 0\,\text{rad}$, the wiper is contacting the middle of the track and θ ranges from $-\pi/2\,\text{rad}$ to $\pi/2\,\text{rad}$.

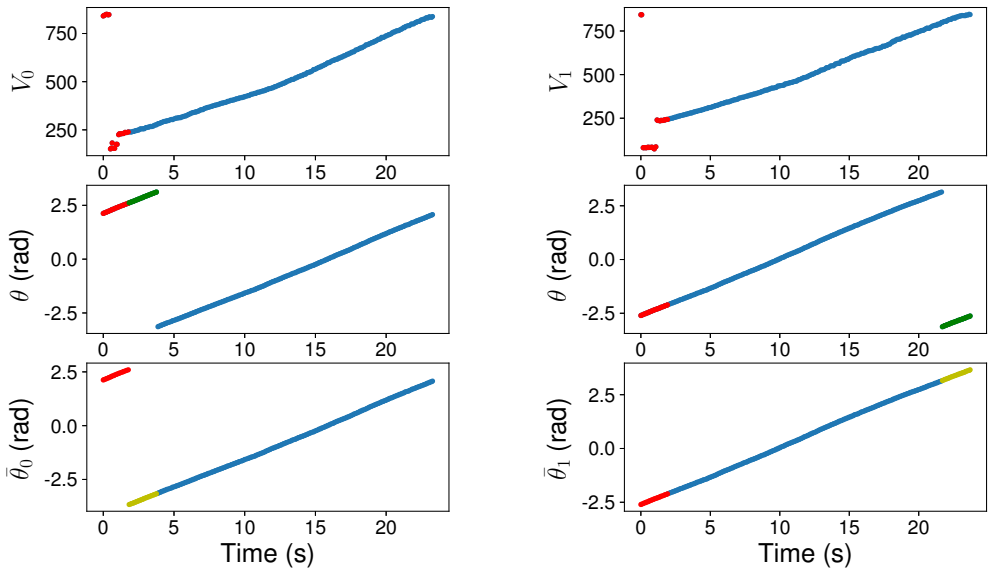


(a) (b)

Figure 5.5: (a) Sensor characterization setup using AprilTags tracking approach. (b) Camera view of three tags in the characterization process.

An automatic sensor calibration setup is developed based on AprilTags tracking [103] shown in Figure 5.5a. Three tags are used to track the rigid bodies of a SMORES-EP module. Tag 2 is fixed to the base to be the reference frame, Tag 1 is fixed to the TOP Face of a SMORES-EP module for TILT DOF tracking, and Tag 0 can be fixed to LEFT Face, RIGHT Face, or TOP Face for wheel DOF tracking (Figure 5.5b). During the characterization, one DOF is moved at a time through its entire range of motion ($2\pi\,\text{rad}$ for wheel DOF, $\pi\,\text{rad}$ for TILT DOF) in both directions. The data, including θ and reported voltage(s), are recorded at 14 Hz (speed limited by the AprilTag ROS package).

While the voltage data is not linear with the DOF position, it is monotonic (piece-wise monotonic for wheel DOFs). A third-order polynomial provides a suitable model. For a wheel PaintPot, the data from both wipers are shown in Figure 5.6a and Figure 5.6b respectively. For wiper 0, the reported voltage V_0 is not useful when DOF position θ is in the range from $\frac{2}{3}\pi\,\text{rad}$ to $\frac{5}{6}\pi\,\text{rad}$ (shown in red). Due to the gap of wheel PaintPots, $\theta = f_0(V_0)$ is a piece-wise function which can be converted into a continuous function $\bar{\theta}_0 = \bar{f}_0(V_0)$ by shifting the segment ranging from $\frac{5}{6}\pi\,\text{rad}$ to $\pi\,\text{rad}$ (shown in green) by $2\pi\,\text{rad}$ downward (shown in



(a) (b)

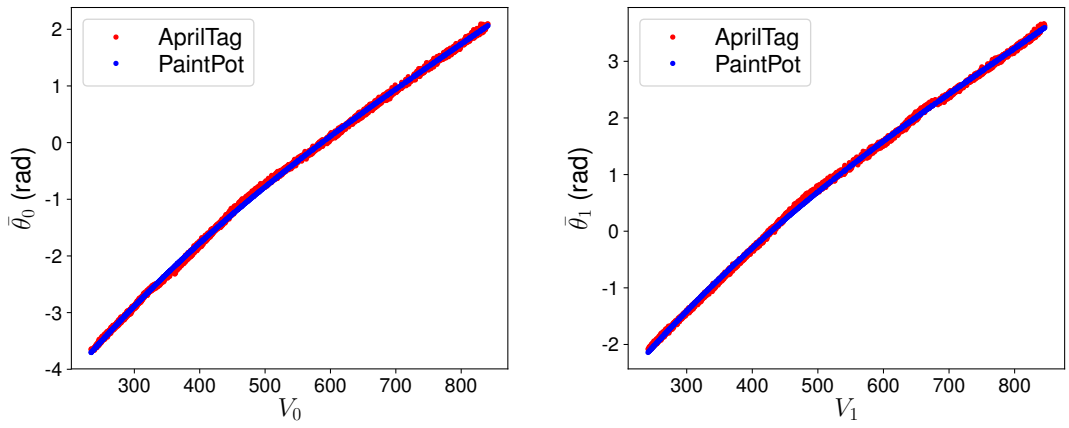
Figure 5.6: (a) Wiper 0 data through the entire range of a wheel PaintPot. (b) Wiper 1 data through the entire range of a wheel PaintPot.

yellow). Similarly, for wiper 1, the segment when θ is in the range from $-\frac{5}{6}\pi$ rad to $-\frac{2}{3}\pi$ rad (shown in red) is meant to be trimmed, and the piece-wise function $\theta = f_1(V_1)$ is converted into a continuous function $\bar{\theta}_1 = \bar{f}_1(V_1)$ by shifting the segment ranging from $-\pi$ rad to $-\frac{5}{6}\pi$ rad (shown in green) by 2π rad upward (shown in yellow). After taking 50 s data in both directions (showing little hysteresis), $\bar{f}_0(V_0)$ and $\bar{f}_1(V_1)$ are shown in Figure 5.7a and Figure 5.7b respectively. An example run from a tilt PaintPot is shown in Figure 5.8a and the characterization result $\theta = f(V_0)$ is shown in Figure 5.8b.

5.4 Position Estimation

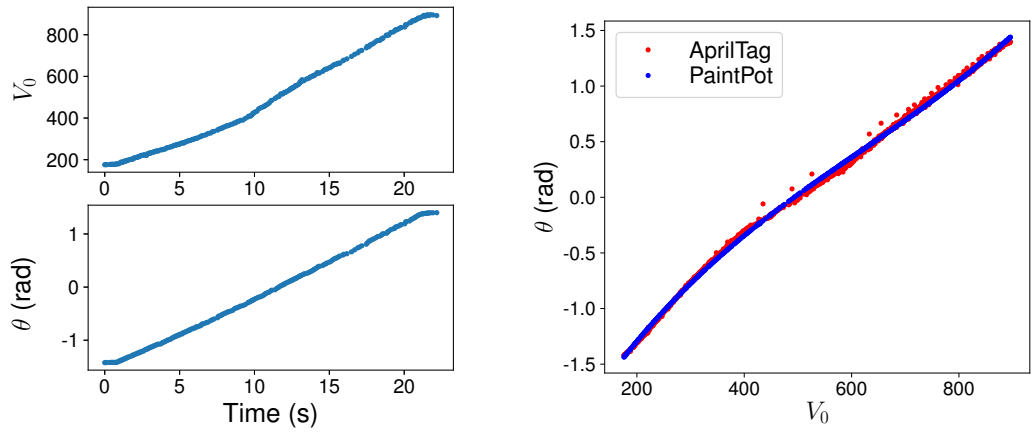
5.4.1 Transition Model

Four DC motors are used to drive four DOFs (LEFT DOF, RIGHT DOF, PAN DOF, and TILT DOF) with a geared drive train shown in Figure 3.1b. The drive has pinion gears driving four identical spur gears. Two outer spur gears are attached to the left wheel and the right wheel respectively for the LEFT DOF and the RIGHT DOF. The crown gear is coupled to the two inner spur gears. When these two inner gears spin in the opposite



(a) (b)

Figure 5.7: Wheel PaintPot sensor characterization results: (a) $\bar{\theta}_0 = \bar{f}_0(V_0) = 5.0281 \times 10^{-9}V_0^3 - 1.2255 \times 10^{-5}V_0^2 + 1.7856 \times 10^{-2}V_0 - 7.2750$; (b) $\bar{\theta}_1 = \bar{f}_1(V_1) = 5.1596 \times 10^{-9}V_1^3 - 1.2409 \times 10^{-5}V_1^2 + 1.7927 \times 10^{-2}V_1 - 5.8128$.



(a) (b)

Figure 5.8: (a) Wiper data from $-80°$ to $80°$ of a tilt PaintPot. (b) The characterization result $\theta = f(V_0) = 4.7517 \times 10^{-9}V_0^3 - 8.7608 \times 10^{-6}V_0^2 + 8.6756 \times 10^{-3}V_0 - 2.7173$.

direction, the top wheel attached to the crown gear rotates which is the PAN DOF. TILT DOF rotates when these two inner spur gears spin in the same direction. The transmission ratio for each DOF is determined by the gear train, then a linear relationship between the DOF velocity and the motor angular velocity can be obtained

$$\dot{\theta} = k\omega + n \tag{5.1}$$

in which k is the transmission ratio of the DOF, ω is the angular velocity of the driving motor(s), $n \sim N(0, Q)$ is the additive Gaussian white noise, and $\dot{\theta}$ is the angular velocity of the DOF. The transition model of a DOF in discrete time can be derived by one-step Euler integration:

$$\begin{aligned} \theta_t &= \theta_{t-1} + \dot{\theta}_{t-1}\delta t + n_{t-1}\delta t \\ &= \theta_{t-1} + k\omega_{t-1}\delta t + n_{t-1}\delta t \\ &= \theta_{t-1} + G\omega_{t-1} + Un_{t-1} \end{aligned} \tag{5.2}$$

in which δt is the finite time interval. Let θ be the state x and ω be the system input u, then the transition model is

$$x_t = x_{t-1} + Gu_{t-1} + Un_{t-1} \tag{5.3}$$

5.4.2 Observation Model

In Section 5.3, a PaintPot can be characterized with a nonlinear model — a third-order polynomial $\theta = f(V)$. Here V is the measurement, namely the reported voltage(s) from the wiper(s). Based on this, a nonlinear observation model $z = h(x)$ where z is the measurement and x is the state can be derived. $z = h(x)$ can be linearized about the current mean and variance of the DOF position x to apply an extended Kalman filter (EKF). However, this places some computational burden on the SMORES-EP microcontroller. Here, a simple approach is presented that can generate a linear observation model directly without linearization of the original nonlinear observation model about the current mean and variance of the state to overcome this nonlinearity. With this new approach, what needs to do is just checking if features are available using a simple rule and compute the newly converted measurements by evaluating polynomials obtained from the sensor characterization (Section 5.3).

Wheel PaintPots

For the wheel PaintPot sensor, the observation model is a piece-wise function due to the geometry of the sensor:

1. When $\theta \in \left(-\pi \text{ rad}, -\frac{5}{6}\pi \text{ rad}\right) \cup \left(-\frac{2}{3}\pi \text{ rad}, \frac{2}{3}\pi \text{ rad}\right) \cup \left(\frac{5}{6}\pi \text{ rad}, \pi \text{ rad}\right)$, there are two valid measurements which are the reported voltages V_0 and V_1 from both wipers;

2. When $\theta \in \left[-\frac{5}{6}\pi \text{ rad}, -\frac{2}{3}\pi \text{ rad}\right]$, only V_1 is valid;

3. When $\theta \in \left[\frac{2}{3}\pi \text{ rad}, \frac{5}{6}\pi \text{ rad}\right]$, only V_0 is valid.

To accommodate the 2π rad shift to obtain a continuous function for sensor characterization (Section 5.3), the observation can be modeled as

$$z = \begin{cases} \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} \bar{f}_0^{-1}(x) \\ \bar{f}_1^{-1}(x + 2\pi) \end{bmatrix} & x < -\frac{5}{6}\pi \text{ rad} \\ V_1 = \bar{f}_1^{-1}(x) & -\frac{5}{6}\pi \text{ rad} \le x \le -\frac{2}{3}\pi \text{ rad} \\ \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} \bar{f}_0^{-1}(x) \\ \bar{f}_1^{-1}(x) \end{bmatrix} & -\frac{2}{3}\pi \text{ rad} < x < \frac{2}{3}\pi \text{ rad} \\ V_0 = \bar{f}_0^{-1}(x) & \frac{2}{3}\pi \text{ rad} \le x \le \frac{5}{6}\pi \text{ rad} \\ \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} \bar{f}_0^{-1}(x - 2\pi) \\ \bar{f}_1^{-1}(x) \end{bmatrix} & x > \frac{5}{6}\pi \text{ rad} \end{cases} \tag{5.4}$$

in which x is the DOF position θ.

In order to avoid linearizing this piece-wise nonlinear observation model, the measurement is changed to be the reported states rather than the two reported voltages. During the motion, there is at least one feature available for tracking, namely at least one wiper is contacting the wheel PaintPot at any time. Let z^i be the measurement from the ith feature, then the measurement model with additive Gaussian white noise is simplified as

$$z^i = x^i + v^i = h^i(x, v^i) \quad i = 0, 1 \tag{5.5}$$

in which $v^i \sim N(0, R^i)$, and x^i is the reported state from the ith feature determined by

state x in the following way:

$$x^0 = \begin{cases} x & x \leq \frac{5}{6}\pi \\ x - 2\pi & x > \frac{5}{6}\pi \end{cases} \tag{5.6a}$$

$$x^1 = \begin{cases} x + 2\pi & x < -\frac{5}{6}\pi \\ x & x \geq -\frac{5}{6}\pi \end{cases} \tag{5.6b}$$

Here, the measurement model is linear and the predicted measurement can be computed easily. The actual measurement for the ith feature can be obtained by evaluating $\bar{f}_i(V_i)$ derived from the sensor characterization process. Recall that when $x \in \left[\frac{2}{3}\pi\,\text{rad}, \frac{5}{6}\pi\,\text{rad}\right]$, V_0 is not valid meaning this feature is not available. Otherwise the actual measurement from this feature is simply $\bar{f}_0(V_0)$ if $x \notin \left[\frac{2}{3}\pi\,\text{rad}, \frac{5}{6}\pi\,\text{rad}\right]$. And the valid range of V_0 is from $V^0_{\min} = \bar{f}_0^{-1}(\frac{5}{6}\pi - 2\pi)$ (because the segment from $\frac{5}{6}\pi\,\text{rad}$ to $\pi\,\text{rad}$ is shifted downward by $2\pi\,\text{rad}$) to $V^0_{\max} = \bar{f}_0^{-1}(\frac{2}{3}\pi)$. Similar procedures can be applied to V_1, and this feature is $\bar{f}_1(V_1)$ if $x \notin \left[-\frac{5}{6}\pi\,\text{rad}, -\frac{2}{3}\pi\,\text{rad}\right]$, otherwise this feature is not available. The valid range of V_1 is from $V^1_{\min} = \bar{f}_1^{-1}(-\frac{2}{3}\pi)$ to $V^1_{\max} = \bar{f}_1^{-1}(-\frac{5}{6}\pi + 2\pi)$.

Tilt PaintPots

The observation model for tilt PaintPots is straightforward which is

$$z = V_0 = f^{-1}(x) \tag{5.7}$$

and it is a nonlinear function. Similarly, in order to avoid linearizing this observation model, the measurement is changed to be the reported state and there is only one feature for tracking. Then the observation model with additive Gaussian white noise is simplified as

$$z = x + v = h(x, v) \tag{5.8}$$

in which $v \sim N(0, R)$ and the model is linear. The feature should always be available and the actual measurement for this feature is obtained by evaluating $f(V_0)$.

5.4.3 Kalman Filter

With the transition model and the new form of observation model, a Kalman filter framework can be applied for state estimation.

Kalman Filter for Wheels

The initial state of a wheel DOF can be derived by any available feature. First check V_0, and if it is inside the valid range from $V^0_{\min}$ to $V^0_{\max}$, compute the initial state $x_0 = \bar{f}_0(V_0)$, and shift x_0 if necessary. That is if $x_0 < -\pi$, let x_0 be $x_0 + 2\pi$. If $V_0 \notin (V^0_{\min}, V^0_{\max})$, then $x_0 = \bar{f}_1(V_1)$ because wiper 1 must contact the valid range of the track at this time, and similarly shift x_0 if necessary, namely if $x_0 > \pi$, let x_0 be $x_0 - 2\pi$. The prior state can be represented as a Gaussian distribution $p(x_0) \sim N(\mu_0, \Sigma_0)$ where $\mu_0 = x_0$ and Σ_0 is initialized to an arbitrarily small value.

With Eq. (5.3), the prediction step is

$$\bar{\mu}_t = \mu_{t-1} + Gu_t \tag{5.9a}$$

$$\overline{\Sigma}_t = \Sigma_{t-1} + U^2 Q \tag{5.9b}$$

With Eq. (5.5), Eq. (5.6a), and Eq. (5.6b), the predicted measurement $\bar{z}_t$ that is related to x^i_t (the reported state from ith feature with $i = 0, 1$) can be computed. The analog-to-digital value V_i from the ith feature at time t is used to compute the actual measurement z^i. If $V_i \in (V^i_{\min}, V^i_{\max})$, $z^i_t = \bar{f}_i(V_i)$. Otherwise, this feature is not available. If both features are available, then $z_t = \left[z^0_t, z^1_t\right]^\intercal$, $C_t = [1, 1]^\intercal$, and the Kalman gain is

$$K_t = \overline{\Sigma}_t C_t^\intercal (C_t \overline{\Sigma}_t C_t^\intercal + R)^{-1} \tag{5.10}$$

in which $R = \text{diag}(R^0, R^1)$. If there is only one feature (e.g. the ith feature) available, then $z_t = z^i_t$, $C_t = 1$, and the Kalman gain is

$$K_t = \overline{\Sigma}_t C_t (C_t^2 \overline{\Sigma}_t + R^i)^{-1} \tag{5.11}$$

The state is then updated:

$$\mu_t = \bar{\mu}_t + K_t(z_t - \bar{z}_t) \tag{5.12a}$$

$$\Sigma_t = \overline{\Sigma}_t - K_t C_t \overline{\Sigma}_t \tag{5.12b}$$

The estimated position for this wheel DOF at time t is μ_t.

Kalman Filter for Tilt

The initial state for a TILT DOF can be derived by evaluating $f(V_0)$. The prior state can be represented as a Gaussian distribution $p(x_0) \sim N(\mu_0, \Sigma_0)$ where $\mu_0 = x_0$ and Σ_0 is initialized with some small value. The prediction step is in the same form with wheel DOFs (Eq. (5.9a), Eq. (5.9b)). The predicted measurement $\bar{z}_t$ can be computed from Eq. (5.8) which is simply $\bar{\mu}_t$. The current actual measurement is computed by evaluating $z_t = f(V_0)$ where V_0 is the current reported voltage. Then the Kalman gain is simply

$$K_t = \overline{\Sigma}_t(\overline{\Sigma}_t + R)^{-1} \tag{5.13}$$

and the state is updated in the following:

$$\mu_t = \bar{\mu}_t + K_t(z_t - \bar{z}_t) \tag{5.14a}$$

$$\Sigma_t = \overline{\Sigma}_t - K_t \overline{\Sigma}_t \tag{5.14b}$$

And the estimated position for TILT DOF at time t is μ_t.

5.5 DOF Control

It is straightforward to develop a position controller for SMORES-EP DOF control. For each DOF, the error on the state and its derivative are defined as

$$e = x_{\text{des}} - x \tag{5.15a}$$

$$\dot{e} = \dot{x}_{\text{des}} - \dot{x} \tag{5.15b}$$

Then a simple feedback position controller can be derived:

$$u = K_{\mathrm{p}} e + K_{\mathrm{d}} \dot{e} \tag{5.16}$$

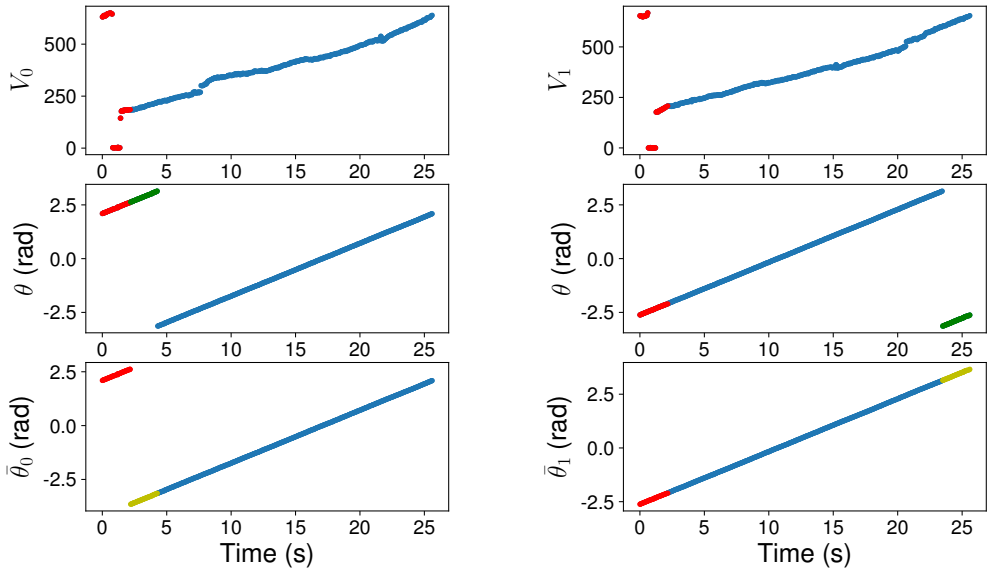
For LEFT DOF and RIGHT DOF, the signed output u_l and u_r computed from Eq. (5.16) can be applied to the corresponding motors directly. PAN DOF and TILT DOF are coupled and driven by the same pair of motors. According to the configuration of these two motors, one motor is driven by command $u_p + u_t$ and the other one is driven by command $u_p - u_t$ where u_p and u_t are the signed output computed from Eq. (5.16).

Given a desired position command for a DOF and the expected motion duration, a polynomial trajectory $x_{\mathrm{des}}(t)$ can be generated from the current position to this desired position. At each time step, the controller first fetch the current position of this DOF from its estimator, then compute the desired position and desired velocity at this moment governed by the pre-generated trajectory $x_{\mathrm{des}}(t)$, and finally compute the output from Eq. (5.16). A similar approach can be used when doing velocity control on a DOF where the trajectory is a linear polynomial with the slope being the desired velocity.

5.6 Experiments

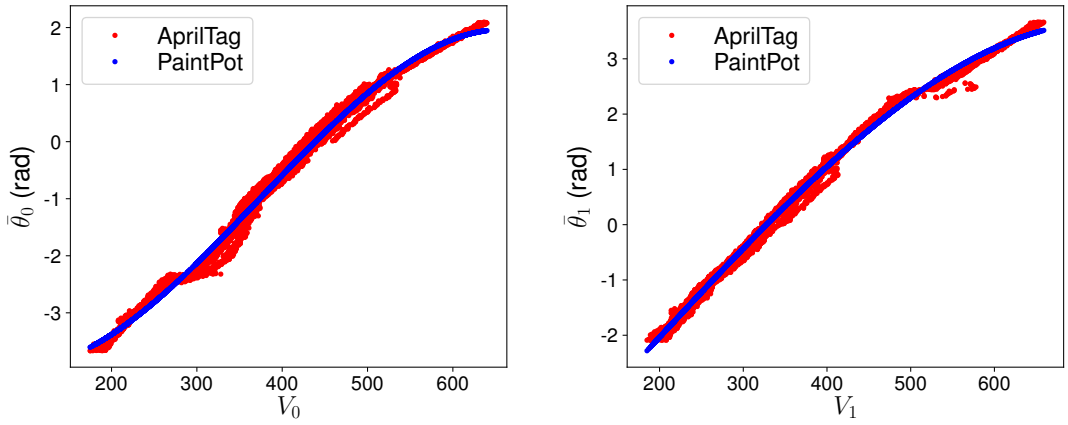
The PaintPot sensors, including wheel PaintPots and tilt PaintPots, are installed in SMORES-EP modules for position sensing. Currently, 25 SMORES-EP modules have been assembled. In the experiments, both a wheel PaintPot and a tilt PaintPot are characterized first and then the newly developed Kalman filters are implemented and used with the DOF controller to show the effectiveness of the low-cost position sensing solution.

The data from both wipers for a wheel PaintPot is shown in Figure 5.9a and Figure 5.9b respectively. The segments labeled by red color are useless, and the green segments are shifted to the yellow segments to generate continuous functions to describe the relationship between the reported voltage and the angular position. This sensor is installed for PAN DOF on a SMORES-EP module. All sensors are painted manually, so the quality is not consistent with no guarantees on bounds. The sensor characterization results for both wipers



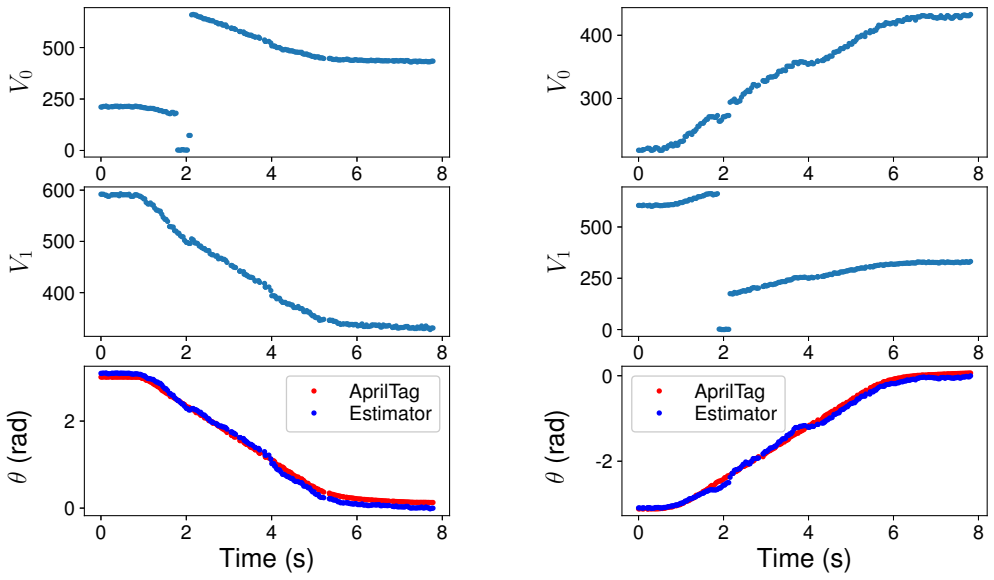
(a) (b)

Figure 5.9: (a) Wiper 0 data through the entire range of a wheel PaintPot. (b) Wiper 1 data through the entire range of a wheel PaintPot.



(a) (b)

Figure 5.10: Wheel PaintPot sensor characterization results: (a) $\bar{\theta}_0 = \bar{f}_0(V_0) = -6.5012 \times 10^{-8} V_0^3 + 7.2912 \times 10^{-5} V_0^2 - 1.1587 \times 10^{-2} V_0 - 3.4595$; (b) $\bar{\theta}_1 = \bar{f}_1(V_1) = -1.8511 \times 10^{-8} V_1^3 + 1.0419 \times 10^{-5} V_1^2 + 1.4362 \times 10^{-2} V_1 - 5.1767$.



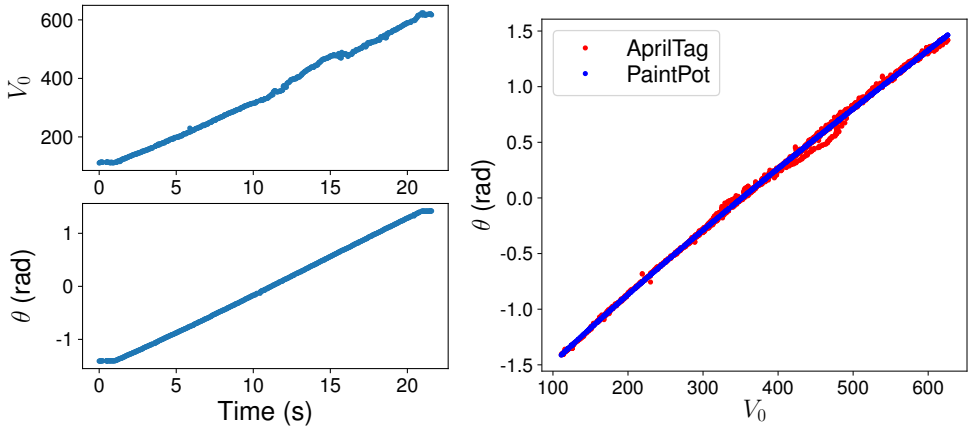
(a) (b)

Figure 5.11: (a) Command PAN DOF to move from angular position π rad to 0 rad. (b) Command PAN DOF to move from angular position $-\pi$ rad to 0 rad.

are shown in Figure 5.10a and Figure 5.10b respectively.

Using the Kalman filter, the estimator is still able to derive a good estimation of the angular position for the PAN DOF on this module. A simple controller is used to command the PAN DOF to a desired position from its current position along a fifth-order polynomial trajectory. The first experiment commands the PAN DOF to angular position 0 rad from π rad resulting in an average error of 0.0878 rad as shown in Figure 5.11a. In this experiment, V_0 is not valid for a while because wiper 0 contacts the gap of the track. The second experiment commands the PAN DOF to angular position 0 rad from $-\pi$ rad with an average error being 0.0698 rad as shown in Figure 5.11b. In this experiment, V_1 is not valid when wiper 1 contacts the gap of the track.

The data from the wiper for a tilt PaintPot is shown in Figure 5.12a and the sensor characterization result is shown in Figure 5.12b. In the experiment, this TILT DOF is commanded to traverse most of the range. The result from the estimator is shown in Figure 5.13 with an average error being around 0.0325 rad.



(a) (b)

Figure 5.12: (a) Wiper data from $-80°$ to $80°$ of a tilt PaintPot. (b) The characterization result $\theta = f(V_0) = 4.4674 \times 10^{-10} V_0^3 - 1.5933 \times 10^{-6} V_0^2 + 6.5369 \times 10^{-3} V_0 - 2.1117$.

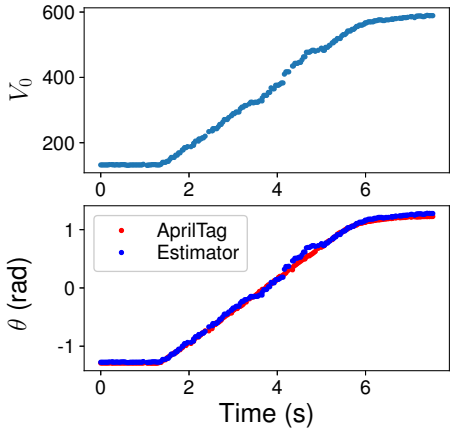


Figure 5.13: Command TILT DOF from angular position -1.3 rad to 1.3 rad.

5.7 Conclusion

In this chapter, the onboard DOF control and estimation are presented. A complete low-cost and highly customizable position estimation solution is presented, especially suitable for highly space-constrained designs that are very common in modular robotic systems. PaintPots are low-cost, highly customizable, and can be manufactured easily by accessible materials and tools in low quantities. For the SMORES-EP system, two different types of PaintPot sensors are used, and a convenient automatic calibration approach is developed

using AprilTags. A modified Kalman filter is developed to overcome the piece-wise nonlinearity of the sensors and some experiments show the accuracy. The successful application of PaintPots in the SMORES-EP system shows that they can provide reliable position information with a simple hardware setup. PaintPots can be easily adapted to and installed on a variety of systems, the not consistent performance due to the manufacturing process can be resolved by the presented characterization process that can be easily set up, and reliable state estimation can be derived by the modified Kalman filter that can be running on a low-cost microcontroller. The overall solution provides a new position sensing technique for a wide range of applications.

Chapter 6

Docking and Undocking

This chapter presents the docking control for the SMORES-EP system, the fundamental to achieve morphology transformation. This chapter excerpts heavily from [78]. Credit is due to co-authors Qian Lin and Hyun Kim, who contributed significantly to this work.

Docking and undocking capability makes self-reconfigurable modular robots different from other robotic systems. This capability enables the dramatic change of topology connections among a set of modules. The docking and undocking process heavily relies on the latching mechanism design. It is desired to have a robust controller to handle a variety of scenarios with high efficiency which significantly affect the efficiency of a morphology transformation process. SMORES-EP uses EP-Faces as its connectors and a pair of connected EP-Face connectors can undock easily. The docking process is divided into three phases with carefully designed controllers to guarantee success. This approach is demonstrated on the SMORES-EP hardware platform and can be easily extended to other similar modular robots. In this work, sensing of module poses is provided by the VICON motion capture system. When operating outdoors, the sensing solution designed for SMORES-EP from [25] can be leveraged.

6.1 Introduction

Docking is a necessary, but usually difficult task for modular robots, which occurs at certain module faces, or connectors. There are many connector designs that can be either gendered

or ungendered. The general requirements for modular robot docking interfaces are high strength, high speed of docking/undocking, low power consumption, and a large area of acceptance [29].

Mechanical devices or structural hook-type connectors are widely designed for docking activities. Robotic arms and grippers are used in SMC Rover [93], Gunryu [45], and Swarm-bot [38]. The mechanical design is straightforward and easy to be strong, but difficult to be compact, which limits the robot flexibility. Structural hook-type connector are used in CEBOT [32], Millibot trains [12], T.E.M.P. [102], M-TRAN III [94], and Sambot [152]. The SINGO connector [129] for SUPERBOT is hermaphroditic, and capable of disconnection even when one module is unresponsive, allowing for self-repair. These docking systems are mechanically complicated, and usually require a large amount of space. Accurate positioning is also crucial for these connectors resulting in a small area of acceptance. The design complexity may also cause failure over time.

Magnets exist in many modular robotic systems for docking. Permanent magnet interfaces are easy to implement and have a relatively large area of acceptance. Modules can be easily docked as long as they are close to each other within some distance, with the misalignment adjusted automatically. However, extra actuation is needed for undocking. M-TRAN [96] uses permanent magnets for latching and undocks using shape memory alloy (SMA) coils to generate the required large force. However, it takes minutes for these coils to cool leading to slow response, and this connector is not energy efficient. Permanent magnets are also used in programmable parts [63] for latching. This docking interface is gendered. The original SMORES system utilizes permanent magnets as docking interfaces as well and a unique docking key for undocking [26]. Permanent magnets are placed at the frame corners of a flying system called ModQuad [116]. Here modules dock by adjoining the frame corner magnets and undock with aggressive maneuvers of vehicle structures [118].

The docking interface design can significantly affect the docking execution. In general, the aim is to make the level of positioning precision required for docking as low as possible, or extra high-quality sensors are needed to provide precise pose feedback. Connector area-

of-acceptance and some preferred properties are studied in [28, 101]. Infrared (IR) sensors are installed on the docking faces for assisting alignment of hook-type connectors, such as PolyBot [162], CONRO [113], and the heterogeneous system in SYMBRION project [85]. Two mobile robots are shown to dock using a visual-based system that uses a black-and-white camera to track a visual target [8]. Vision-based approaches were developed for M-TRAN III [94] by placing a camera module on a cluster of modules that can detect LED signals. However, guidance by this visual feedback is not sufficient to realize positional precision for its mechanical docking interface and extra efforts from the cluster are required for connection. Similarly, Swarm-bot applies an omni-directional camera to detect docking slots with a ring of LEDs, but cannot guarantee the success of docking. In comparison, the docking process can be easier with magnet-based connectors. Using a similar visual-based solution with M-TRAN III, CKBot clusters with magnet faces show more robust and easier docking procedures [161]. The large area-of-acceptance of magnet docking faces can also be seen by stochastic self-assembly modular robots [40, 155, 156].

In SMORES-EP, electro-permanent magnets (EP magnets) [64] are used as its connectors [143]. The magnets can be switched on or off with pulses of current, unlike electromagnets which require sustained current while on. In the design, the EP magnet is driven by three pulses under 11.1 V battery voltage. A pair of EP-Face connectors can provide as much as 90 N to maintain their docking status and very little energy is consumed to connect with or disconnect from each other. Magnetic forces can draw two modules together through a gap of 4 mm normal, and 7 mm parallel to the faces leading to a large area of acceptance. Incorporated with the proposed control approach, a single docking action can be executed within seconds.

6.2 Docking Control

In SMORES-EP, undocking can be executed by simply applying pulses of current to switch all involved magnets off. However, docking is more difficult. Given a docking task, that is to connect module m_i's connector c_i with module m_j's connector c_j, the whole process is divided into three phases to ensure its success: *navigation*, *pose adjustment*, and *approach*.

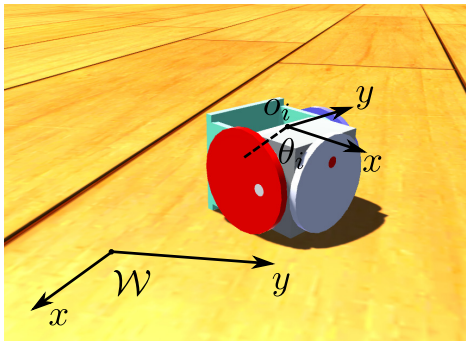


Figure 6.1: A SMORES-EP module is on the ground.

6.2.1 Navigation

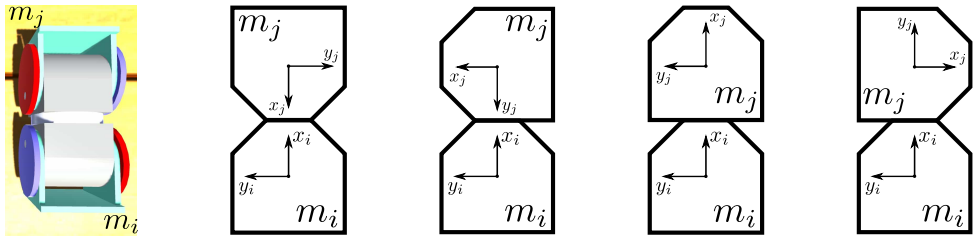
Each SMORES-EP module can behave as a differential-drive vehicle. The state of a module m_i on the ground is defined as $p_i = [x_i, y_i, \theta_i]$ where $o_i = [x_i, y_i]^\intercal$ is the location of the center of m_i and θ_i is the orientation of m_i (Figure 6.1). Its differential-drive kinematics model is

$$\dot{p} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{6.1}$$

in which v is the linear velocity along x-axis of the body frame of m_i and ω is the angular velocity around z-axis of the body frame of m_i which are determined by the velocity of LEFT DOF and RIGHT DOF. A simple control strategy to control this module m_i to go to a desired position is first adjusting its pose θ_i to face the destination and then moving along a straight line toward the destination. Given the docking task mentioned earlier, a collision-free path can be generated for m_i to go to a location close to m_j along the routes predefined by a roadmap. A simple PID controller is used to calculate the linear and angular command (v and ω).

6.2.2 Pose Adjustment

Once the navigation procedure is done for module m_i, namely m_i is located somewhere that is close to module m_j, module m_i starts to adjust its pose to align the involved connector.



(a) (b) (c) (d) (e)

Figure 6.2: (a) m_i TOP Face is connected with m_j TOP Face. (b) its kinematic diagram. (c) — (e) are the kinematic diagrams of the three other cases when m_i TOP Face is involved in the connection.

The desired pose of module m_i is fully determined by module m_j. For example, TOP Face of m_j is attached to TOP Face of m_i shown in Figure 6.2a (kinematic diagram in Figure 6.2b). There are three more cases with TOP Face of m_i being involved shown in Figure 6.2c — Figure 6.2e. The pose of m_i with respect to m_j denoted as p_{ij} is determined by the involved connectors, e.g. $p_{ij} = [w, 0, \pi]^\intercal$ for Figure 6.2b. Then given $p_j = [x_j, y_j, \theta_j]^\intercal$, the pose of m_i is

$$p_i = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} p_{ij} + p_j \tag{6.2}$$

in which $R = \begin{bmatrix} \cos\theta_j & -\sin\theta_j \\ \sin\theta_j & \cos\theta_j \end{bmatrix}$. In this way, the target pose of module m_i can be calculated easily.

If c_i (the connector of m_i to be attached with m_j) is either LEFT Face or RIGHT Face, then adjust x_i' and θ_i' to zeros where x_i' is the x location of m_i with respect to the goal

(a) (b)

Figure 6.3: (a) The docking task is to connect LEFT Face of m_i with m_j and the goal pose of m_i is shown in dashed line. In this case, m_i needs to align the connector by adjusting x_i' and θ_i' to zeros. (b) If the assembly action is to connect TOP Face of m_i with m_j, then m_i needs to align the connector by adjusting y_i' and θ_i' to zeros.

pose of m_i and θ_i' is the orientation of m_i with respect to the goal pose of m_i. Otherwise, adjust y_i' and θ_i' to zeros where y_i' is the y location of m_i with respect to the goal pose of m_i. Two cases are shown in Figure 6.3. Note that this process has nothing to do with c_j. A kinematics model for the second case can be derived as

$$\begin{bmatrix} \dot{y}_i' \\ \dot{\theta}_i' \end{bmatrix} = \begin{bmatrix} \sin\theta_i' & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{6.3}$$

A control law to make y_i' and θ_i' to converge to zeros is

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \sin\theta_i' & 0 \\ 0 & 1 \end{bmatrix}^{-1} K_{2\times 2} \begin{bmatrix} -y_i' \\ -\theta_i' \end{bmatrix} \tag{6.4}$$

where K is positive definite and $K = \mathrm{diag}(2, 1)$ is used in the experiments. A similar controller can be derived for the first case. For the pose adjustment controller, the constraint on one dimension is relaxed to make the system fully actuated. For example, with the controller in Eq. (6.4), only y_i' and θ_i' are under control and x_i' is no longer constrained, so x_i' may diverge possibly resulting in collision. However, this drift is not a concern for SMORES-EP modules which use differential drive. This is confirmed in the hardware experiments.

6.2.3 Approach

The last step is to approach c_j by moving in a straight line which is similar to the controller used in the navigation step following a given trajectory. If c_i is either TOP Face or BOTTOM Face, then module m_i will first adjust c_i to the right position and then keep moving and eventually pushing m_j until c_i and c_j are fully connected. Otherwise, when c_i is either LEFT Face or RIGHT Face, a helping module shown in Figure 6.4 is needed. An extra docking action that is docking TOP Face of the helping module m_{H} with m_i's connector $\bar{c}_i$ where $\bar{c}_i$ is LEFT Face if c_i is RIGHT Face, or the vice versa, is needed. After m_{H} is connected with m_i, m_i is lifted so that it can adjust c_i to the right position, delivered to its destination, and then placed down. Finally, m_{H} keeps moving to push m_i to approach m_j for docking. The helping

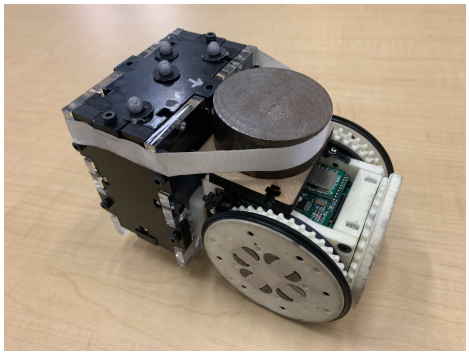


Figure 6.4: A helping module is a SMORES-EP module equipped with some payload so that it can lift another module.

module is a SMORES-EP module equipped with an extra mass on the BOTTOM Face as a counterbalance while lifting a module. While moving modules translate, the controller should keep the orientation of these modules fixed as docking requires the magnet faces to mate properly. To ease this requirement on the SMORES-EP modules, the EP-Faces have a relatively large area of acceptance.

6.3 Experiment

In the setup, an empty grid map where each grid cell is a square 10 cm×10 cm is used to provide a roadmap. In this experiment, the pose of Module 3 is $[-0.318\,\text{m}, -0.132\,\text{m}, 0.454\,\text{rad}]^\intercal$ and the pose of Module 4 is $[-0.16\,\text{m}, 0.0\,\text{m}, 0.454\,\text{rad}]^\intercal$. The goal is to dock TOP Face of Module 3 with BOTTOM Face of Module 4. VICON motion capture system is used to track

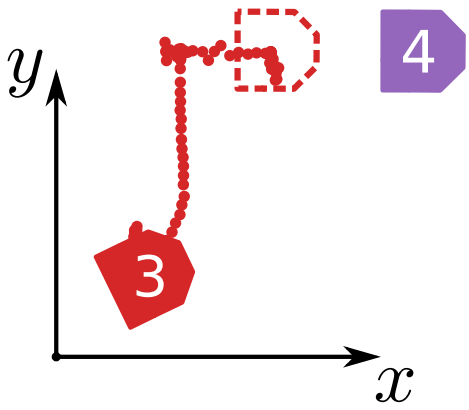
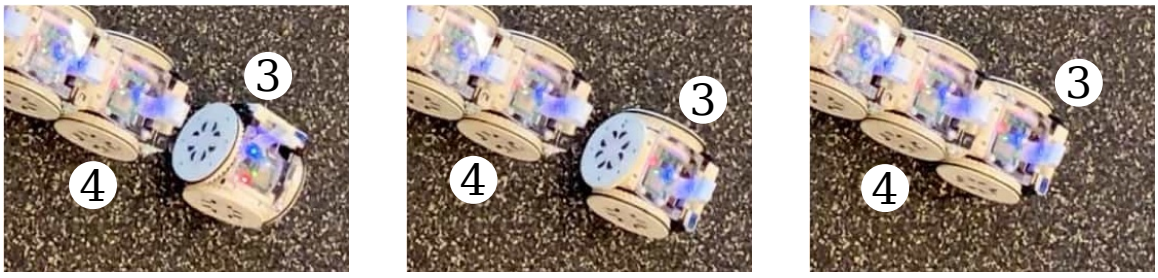


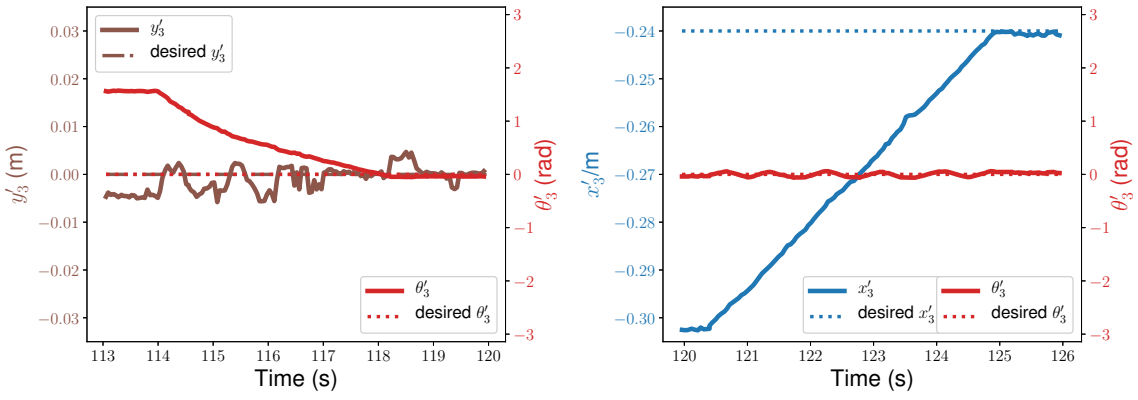
Figure 6.5: The tracked position of Module 3 in the docking process.

their poses. The tracked motion of Module 3 is shown in Figure 6.5. Module 3 first navigates to a location close to Module 4, then starts to adjust its body frame (Figure 6.6a) so that its TOP Face is aligned with BOTTOM Face of Module 4 (Figure 6.6b) and the performance is shown in Figure 6.7a. The controller can adjust the pose of Module 3 quickly and align its connector within 6 s while almost keeping x_3' fixed. Then Module 3 moves forward to Module 4 for final docking (Figure 6.6c) and the performance is shown in Figure 6.7b. It can be seen that Module 3 can steadily approach Module 4 while maintaining its orientation fixed (very little oscillation of θ_3' around zero value) so that the involved connector is always



(a) (b) (c)

Figure 6.6: Adjustment of the position and orientation before docking BOTTOM Face of Module 3 with TOP Face of Module 4: (a) Module 3 finished navigation process and started to adjust its pose; (b) y_3' and θ_3' have been adjusted and it started to approach the goal for docking; (c) The docking process of Module 3 was accomplished.



(a) (b)

Figure 6.7: Pose adjustment of Module 3 before docking: (a) adjusting y_3' and θ_3'; (b) adjusting x_3' while maintaining the correct orientation.

aligned.

6.4 Conclusion

In this chapter, a control strategy for docking is proposed. Given a docking action, the process is divided into three phases: navigation, pose adjustment, and approach. Motion controllers are developed to ensure the success of docking between modules which is demonstrated by the SMORES-EP hardware platform.

Chapter 7

Graph Model and Configuration Recognition

A modular robot can be in a large number of morphologies or configurations and the number of morphologies increases exponentially as the number of modules increases. This chapter presents a general modeling approach for modular robots as well as a configuration recognition algorithm. These tools are also the foundation of the self-adaption problem in the next chapter. This chapter excerpts heavily from [81].

In a modular robotic system, a module can have several DOFs and connectors, and there is usually a huge number of ways to connect a set of modules. Hence, the model of a robot varies with the change of its morphology. A general modeling method is necessary to characterize and distinguish them. This is different from fixed-shape robots in that the connections among their links and joints don't change. In addition, these modular robot morphologies or configurations should be invariant under different labeling. It is desired for modular robots to be capable of recognizing their current configuration. It is common to build a library of configurations with properties and controllers. In order to reuse these well-developed techniques, the robot has to match itself to an existing one in the library and, if true, map each module to a unique module in the matching configuration. In this work, a general graph model is proposed for modular robots. The cluster can be discov-

ered by leveraging distributed information from each module. Then an efficient algorithm is presented to address the matching and mapping problem in polynomial time. The algorithm is demonstrated on the SMORES-EP system and shown to be efficient to solve this configuration recognition problem.

7.1 Introduction

Modular robots are able to adapt or be adapted to many different functions or activities, handle hardware and software failures, and also be cost-effective to be used in more cost-sensitive tasks [135]. With these advantages, modular robotic systems are promising to do a wide variety of tasks. However, it is also a challenge to come up with planning and control algorithms to handle numerous modules. One key reason is that the number of all possible configurations of the system increases drastically as the number of modules increases. Plus, each module may have multiple DOFs and the configurations in topology become even more complicated. Hence, a general modeling approach is necessary to characterize and differentiate these configurations which are supposed to be invariant of explicit labels.

Graphs have been shown useful to represent modular robot configurations and there are several techniques from graph theory for us to use. In this work, the graph model of a configuration can be captured in a distributed manner, and then an automatic configuration recognition algorithm is proposed. It is useful for modular robots if they can automatically identify their configurations so that there is no need to program this information every time when constructing a new configuration. A library of useful configurations, as well as preprogrammed behaviors, is usually built for modular robots to handle a variety of tasks. Configuration recognition is necessary in order to make use of these well developed techniques, namely matching a new modular robotic configuration to an existing configuration in a library and mapping each module in these two configurations. This process is also helpful in self-repair and rapid manual repair discussed in [105].

As mentioned, the number of all possible configurations of modular robots grows drastically as the number of modules increases, so it is challenging to put forward an efficient algorithm for configuration recognition, which is even more difficult when taking the connec-

tion types between modules into consideration. This chapter presents a new representation for modular robot configurations and a polynomial-time algorithm for configuration recognition which can solve matching and mapping problems simultaneously. In addition, a local algorithm for configuration discovery of modular robots in linear time is introduced and demonstrated on the SMORES-EP platform.

7.2 Related Work

Graphs have been widely used to represent modular robot configurations. Polypod [158] and PolyBot [159] are represented as graphs where modules are nodes and connections are edges. This graph representation is shown to be useful for closed-chain reconfiguration planning [15, 164]. The graph model can also be represented as a vertex-edge incidence matrix (IM) or assembly incidence matrix (AIM) with connecting port information embedded [20]. Similarly, the graph representation can be converted into an adjacency matrix or port-adjacency matrix [105]. Connecting port information can also be embedded in an unlabeled directed graph [17]. Furthermore, Connector-Graph (C-Graph) embeds the orientation information of each connection [49]. In essence, these are all graph-based representations with different information embedded.

It is desired to enable a modular robot to identify its current configuration which is necessary when deploying configuration recognition algorithms in real hardware platforms. Butler *et al.* [13] presented a distributed recognition algorithm for lattice-based systems. Castano and Will [17] proposed a centralized method to build the configuration representation. In contrast, Hou [47] proposed a distributed method to collect the necessary connection information among all modules. Baca *et al.* [5] also presented a real-time distributed algorithm to discover the modules and the topology of the configuration.

Based on these graph representations and the output from the configuration discovery process, a variety of configuration recognition approaches are introduced. Chen and Burdick [20] presented a method to enumerate the non-isomorphic assembly configurations from a set of modules. Park *et al.* [105] compared three methods of matching and mapping, an automorphism grouping method (*nauty*), a spectral decomposition approach, and a heuristic

graph search (3DLL). In particular, the *nauty* method first finds all graph isomorphism configurations with the adjacency matrix of the given robot configuration and then finds the matching configuration by checking the port-adjacency matrix and considering the symmetry property of the modules. However, this method scales badly with the increase of the number of modules. The spectral decomposition method is trying to find the permutation matrix between isomorphic configurations which are represented by port-adjacency matrices. This method requires more computation for structures with symmetry and numerical stability is a concern for large matrices, and the time to compute eigenvectors becomes extremely long at a very large scale. For the 3DDL method, robots are represented by a three-dimensional linked list of module objects (3DDL) and some heuristics are evaluated in order to filter out most of the configurations in the library concerning all possible choices of origin module. However, for different modular robotic systems, different reasonable heuristics may need to be designed and some heuristics can be invariant under some similar configurations so that it cannot find the accurate matching. Also, the 3DLL method is specific to CKBot and cube-oriented modules [105]. An algorithm based on the methods of linear algebra to recognize isomorphism between two configuration graphs of modular robots is discussed in [131]. It is also focusing on the configuration matrix based on the adjacency matrix and the complexity is $\mathcal{O}(n^6 \log n)$ in terms of manipulations of elementary operations. An improved 3DLL approach is presented in [169]. It first finds all graph isomorphism configurations with the adjacency matrix (similar to *nauty*) which may take a lot of time for configurations with a large amount of modules and then compare the configuration's space representation list with those in the library which requires the global position and orientation information with respect to a base module, so it only works for modular robots with fixed discrete states for each DOF.

7.3 Graph Representation and Library Design

A good configuration representation is helpful for configuration recognition. The basic requirement to construct the database is that each configuration should have a unique representation in the database. In particular, a modular robot configuration can be represented as

a graph. Each vertex of the graph represents a module and each edge of the graph represents a connection between two modules. This graph can then be converted into an adjacency matrix and then a port adjacency matrix [105] or a configuration graph matrix [131] which has been discussed in Section 7.2.

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices of G and E is the set of edges of G. Graphs with only one path between each pair of vertices are *trees*. Any acyclic graph is a tree. Some preliminaries on rooted trees are discussed here. Once a tree $G = (V, E)$ is rooted with respect to a vertex $\tau \in V$, the parent of a vertex $v \in V$ is the vertex connected to it on the path to τ which is unique except for τ, and the child of a vertex $v \in V$ is a vertex of which v is the parent. The configuration of a modular robot cluster is represented as a rooted tree and the root has to be selected as the center of its graph [88] defined as the following:

Definition 1. Let T be a tree. The **center** of T is the unique vertex (or unique pair of adjacent vertices) such that removing that vertex (or vertices) from T leaves a collection of components each having less than half the vertices of T.

Given a tree T and a vertex v, $T - v$ is the result of removing v from T. If $T - v$ is connected, v is a leaf; otherwise, $T - v$ consists of several acyclic components, and if $T - v$ has no components of order greater than $n/2$, v is a central vertex. This is the rule to define the root for the graph of a given configuration.

A modular robot module usually has multiple connectors and there may also be multiple ways to connect them. For each connection between two modules, the involved faces and orientations are meant to be considered. A connection in a configuration is defined as the following:

Definition 2. A **connection** between module u's connector U_Con and module v's connector V_Con with orientation Ori is defined as

$$\text{connect}(u, v) = \{\text{Face} : \text{U\_Con}, \text{Face2Con} : \text{V\_Con}, \text{Orientation} : \text{Ori}\}$$

from module u's point of view and

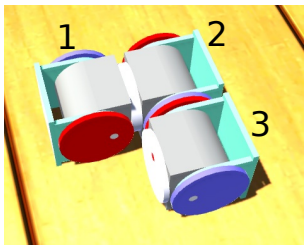
$$\text{connect}(v, u) = \{\text{Face} : \text{V\_Con}, \text{Face2Con} : \text{U\_Con}, \text{Orientation} : \text{Ori}\}$$

from module v's point of view.

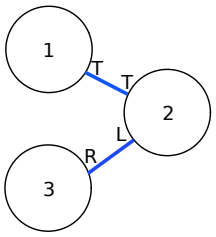
Each connection has three attributes: *Face*, *Face2Con* and *Orientation*, so modular robots can be connected in multiple ways and the number of all possible configurations grows dramatically as the number of modules increases. However, some seemingly different connections can actually be considered equivalent. This is a general definition of *connection* for modular robots and attributes vary with different systems. For example, each SMORES-EP module has four faces (LEFT Face, RIGHT Face, TOP Face, BOTTOM Face) and each face can be configured to connect with any face of other modules. Since there are no angular limits on rotation for LEFT DOD, RIGHT DOF, and PAN DOF, and EP-Faces are hermaphroditic, for connections among LEFT Face, RIGHT Face, and TOP Face, the connections in which only the *Orientation* attribute is different are equivalent. In contrast, for connections between two BOTTOM Faces, the orientation needs to be considered and there are two different orientations (Orientation = $[0, 1]$) in total which is shown in Figure 7.1. In addition, the module is bilaterally symmetric, namely LEFT Face or LEFT DOF is a mirror image of RIGHT Face or RIGHT DOF, so all possible connections between LEFT Face and RIGHT Face are also equivalent, namely all connections with LEFT Face or RIGHT Face as their *Face* or *Face2Con* attribute are equivalent. $\cong$ is used to denote the equivalent relation between two connections.

(a) (b)

Figure 7.1: Connecting two BOTTOM Faces: (a) Orientation is 0; (b) Orientation is 1.



(a)



(b)

Figure 7.2: (a) A three-module configuration. (b) The corresponding graph representation.

A modular robot configuration can be fully determined using a graph $G = (V, E)$, in which V is the set of modules and E is the set of edges containing connection information defined by Definition 2. A three-module SMORES-EP configuration example is shown in Figure 7.2. For this configuration, the connections can be expressed as following:

$$\text{connect}(1, 2) = \{\text{Face} : \text{TOP Face}, \text{Face2Con} : \text{TOP Face}, \text{Orientation} : \textsc{Null}\}$$

$$\text{connect}(2, 3) = \{\text{Face} : \text{RIGHT Face}, \text{Face2Con} : \text{LEFT Face}, \text{Orientation} : \textsc{Null}\}$$

In addition, for this simple configuration, the root module can be defined as Module 2 according to Definition 1.

The library is a collection of modular robot configurations and each configuration has its unique representation. The representation of each configuration contains the corresponding rooted graph G, the information of all connections defined in Definition 2 and all CN values described in Section 7.4.2. Some more properties can also be added to configurations in the library, like robot behavior property, for other design purposes. While, for configuration recognition, they are not necessary.

7.4 Algorithms for Configuration Recognition

This section presents the modular robot configuration recognition algorithm that can solve the matching and mapping problem simultaneously in polynomial time. The algorithm contains three parts:

1. Discover the configuration of a cluster of modular robots;

2. Given a new modular robot configuration, decide the root module;

3. Verify if the configuration can be matched with an existing configuration in the library and, if true, map each module to the module in this known isomorphic configuration.

7.4.1 Configuration Discovery

When a modular robot configuration is constructed, a fully autonomous robotic system has to be able to figure out the configuration by itself, then the graph representation can be discovered to represent its current configuration. This discovery process is affected by the hardware design, especially how a module communicates with its neighbors and the communication protocol. For the SMORES-EP modular robotic system, every module can talk to its neighbors via EP-Faces. Sending and receiving messages share the same route so that each EP-Face is either in sending state or receiving state. In addition, the system requires a central controller to control the whole cluster of modules. Each module can only exchange data with this central controller via Wi-Fi and exchange data with other modules via EP-Face connectors.

A cluster discovery algorithm based on distributed information among modules is presented. The pseudocode is presented in Algorithm 1. The input to this algorithm is a starting module. This module can be selected randomly. The output is all the connections in this configuration. The idea is to traverse every module in the breadth-first search order and record every connection one by one.

Line 9: The algorithm only checks faces that are not verified. Initially, every face of every module is not verified and each face can be verified via two conditions: 1. when a face is not verified in *Line 9*, mark it to be verified; 2. when a face receives messages in *Line 15*, mark it to be verified.

Line 10 — 14: Switch all not verified faces in the configuration to Receiving Mode, except F, to wait for the message sent from F.

Line 15 — 18: If there is a connection that this face F is involved in, the connected

Algorithm 1: Configuration Discovery

Input: Start module S
Output: Configuration connections E

```
1  Create empty set V to store visited modules;
2  Create empty queue Q;
3  Q.enqueue(S);
4  while Q is not empty do
5      Current module C = Q.dequeue();
6      if C ∉ V then
7          V ← V ∪ {C};
8          Create empty set NM to store newly recognized modules;
9          foreach Face F of C that is not verified do
10             foreach module M ∉ V do
11                 foreach Face F′ of M that is not verified do
12                     Switch F′ to Receiving Mode;
13                 end
14             end
15             Switch F to Sending Mode and send message;
16             Switch F to Receiving Mode;
17             Update connections E;
18             Update NM;
19         end
20         Q.enqueue(NM − V);
21     end
22 end
```

face F' should be able to receive the message sent from F. Then after some certain time during which F is already switched to Receiving Mode in *Line 16*, the connected module will control F' to send out feedback message and turn off the communication function for F' so F should receive this feedback message then module C will also turn off the communication function for F. Only MAGNET 1 (there are four magnets in an EP-Face) is active when sending and all magnets are active when in Receiving Mode so that the orientation can be determined by checking which magnet can receive messages. Both messages contain module ID and sending face information. Then both modules will store the connection information. This connected module will then be added to set NM in *Line 18*. If there is no connection, namely after some certain time F doesn't receive any feedback message, module C will also record that there is no connection on F and turn off the communication function, and set

NM will remain the same.

Once this process is done, every module stores its connection information locally and the central computer will request this information from modules one by one. This will then be used to build the graph representation of the configuration.

7.4.2 Root Module

The root module is defined to be the center vertex of the graph of a given modular robot configuration. As discussed in Section 7.3, intuitively this can be done by iteratively removing each vertex in the graph and counting the number of vertices in these generated acyclic components that requires the traversal of these subgraphs. An efficient algorithm using dynamic programming to figure out the root module is presented here. The idea is to compute the order of all acyclic components after removing every vertex in the graph, and then find the one or a pair satisfying the requirement. This problem has been solved in [47] in a distributed way in that every module runs the same code and exchanges connection information with its neighbors. This requires the robots to exchange more data and it is time-consuming when coordinating all the modules and ensuring that they can receive data properly. This is especially difficult for a system like SMORES-EP in which the sending mode and receiving mode between connectors share the same route. Hence, a centralized algorithm that uses local information gathered in configuration discovery is presented.

Based on the gathered local information, the graph of a configuration $G = (V, E)$ can be built and rooted with respect to a random module v_0. The parent and the children of any vertex $v \in V$ are then determined. The set of connectors is denoted as $\mathcal{C}$ and $\mathrm{CN}^v(c)$ is defined to denote the total number of modules connected to v via its connector $c \in \mathcal{C}$ (as in [47]) which is the number of vertices of the component of $T - v$ corresponding to connector c (for SMORES-EP modules, $\mathcal{C} = \{\text{LEFT Face}, \text{RIGHT Face}, \text{TOP Face}, \text{BOTTOM Face}\}$). The number of modules n satisfies

$$n = \sum_{c \in \mathcal{C}} \mathrm{CN}^v(c) + 1, \ \forall v \in V \tag{7.1}$$

and a configuration graph G can be rooted with respect to all n vertices but $\mathrm{CN}^v(c)$ is invariant under the selection of the root.

For any vertex $v \in V$ that is not a leaf with respect to root τ, its child connected via its connector c is denoted as $\hat{v}^c$, the mating connector of $\hat{v}^c$ as $\hat{c}'$, the set of its children as $\mathcal{N}(v,\tau)$, and the set of connectors connected with its children as $\mathcal{C}_d(v) \subseteq \mathcal{C}$. If $v \in V$ has both a parent and some children, and $\mathcal{N}(v,\tau)$ and $\mathcal{C}_d(v)$ are known, then

$$\mathrm{CN}^v(c) = \sum_{\hat{c} \in \mathcal{C} - \hat{c}'} \mathrm{CN}^{\hat{v}^c}(\hat{c}),\ c \in \mathcal{C}_d(v) \text{ and } \hat{v}^c \in \mathcal{N}(v,\tau) \tag{7.2a}$$

$$\mathrm{CN}^v(c) = n - 1 - \sum_{c' \in \mathcal{C}_d(v)} \mathrm{CN}^v(c'),\ c \notin \mathcal{C}_d(v) \tag{7.2b}$$

Eq. (7.2a) and Eq. (7.2b) construct the recursive solution to figure out all $\mathrm{CN}^v(c)$. According to Definition 1, root module τ has to satisfy the following condition:

$$\mathrm{CN}^\tau(c) \leq \frac{1}{2}n,\ \forall c \in \mathcal{C} \tag{7.3}$$

The descendants of $v \in V$ with respect to root τ is denoted as $\mathrm{desc}(v,\tau)$. With the above

Algorithm 2: Root Module Search

Input: Graph representation $G = (V, E)$
Output: Root module τ

```
Root G = (V, E) with respect to a module v_0 with height of H and the height of
  v ∈ V is h(v);
Initialize CN^v(c) for v ∈ V and c ∈ C to be zero;
Initialize h ← 1;
while h ≤ H do
    foreach module v with h(v) = h do
        foreach module v̂^c ∈ N(v, τ̂) do
            CN^v(c) = Σ_{ĉ ∈ C − ĉ'} CN^{v̂^c}(ĉ) ∀c ∈ C_d(v);
            CN^{v̂^c}(ĉ') = n − 1 − Σ_{ĉ ∈ C_d(v̂)} CN^{v̂^c}(ĉ);
        end
    end
    h ← h + 1;
end
τ ← v ∈ V such that CN^v(c) ≤ (1/2)n ∀ c ∈ C
```

recursive solution, solving $\mathrm{CN}^{v_1}(c_1)$ and solving $\mathrm{CN}^{v_2}(c_2)$ have overlapping subproblems if $v_2 \in \mathrm{desc}(v_1, \tau)$. They both need to solve $\mathrm{CN}^{u}(c)$, where $u \in \mathrm{desc}(v_2, \tau)$ and $c \in \mathcal{C}_d(u)$. A bottom-up algorithm is constructed. The pseudocode is presented in Algorithm 2. The algorithm starts from vertices whose height are one. If $v \in V$ is a leaf, then $\mathcal{C}_d(v) = \emptyset$ and $\mathrm{CN}^{v}(c) = 0$ except when c is the connector connected with its parent. For $v \in V$ with $h(v) > 0$, $\mathcal{N}(v, v_0) \neq \emptyset$ and $\mathcal{C}_d(v) \neq \emptyset$. $\mathrm{CN}^{v}(c)$ when $c \in \mathcal{C}_d(v)$ can be computed using Eq. (7.2a) and also compute $\mathrm{CN}^{\hat{v}^c}(\hat{c}')$ where $h(\hat{v}^c) = h(v) - 1$ using Eq. (7.2b). In this iteration, $\mathrm{CN}^{v}(c)$ where $c \notin \mathcal{C}_d(v)$ is not computed which will be computed when visiting its parent vertex. After this iteration, move to the modules with higher height and the algorithm ends until the predefined root v_0 is visited. It is clear that the root module can be found in time $\mathcal{O}(|V|)$.

7.4.3 Matching and Mapping

The configuration graph $G = (V, E)$ can be rooted with respect to the root module searched by Algorithm 2. Then given two configurations, the objective is to verify if they are isomorphic in terms of modular robotic system topology and, if true, map each module from one configuration to that in another configuration. Intuitively, this can be solved by traverse the tree from the root to the leaves and check all the edges and the corresponding $\mathrm{CN}^{v}(c)$ values and, if the connections are equivalent and $\mathrm{CN}^{v}(c)$ are equal, then these two connections share the same topology. However, since modular robots are usually symmetric in their geometry, for example, for SMORES-EP system, the left side is a mirror image of the right side for each module, there will be multiple candidates for some connection when trying to find the equivalent one in the other configuration. It is a heavy computation burden to track all the options and the number of cases to track can quickly grow.

If two modular robot configurations $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, then each module $v_1 \in V_1$ must be able to be mapped to a unique module $v_2 \in V_2$ who shares the same topology and the number of mapped pairs of modules is the number of modules in G_1 or G_2. Hence, if there exists this bijective mapping $f : V_1 \to V_2$, then G_1 and G_2 are isomorphic and f is the mapping that we are looking for. For each module

$v_1 \in V_1$, there may be multiple modules in G_2 that share the same topology with v_1 in G_1. The idea of the algorithm is to try to find some possible mappings which result in multiple common subgraphs of both configurations, and, if these two configurations are isomorphic, then there must be one common subgraph containing all the modules that is also the isomorphism mapping.

Given two modular robot configurations $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, there may be some common parts between them. Two definitions are introduced:

Definition 3. Given two modular robot configurations $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a **common subconfiguration** is a set of connected graphs $\{G'_1, G'_2\}$ where $G'_1 = (V'_1, E'_1) \subseteq G_1$, $G'_2 = (V'_2, E'_2) \subseteq G_2$ such that G'_1 and G'_2 are isomorphic. The corresponding bijective **common subconfiguration mapping** is defined as $f' : V'_1 \to V'_2$.

Definition 4. Given the condition that module $v_1 \in V_1$ of $G_1 = (V_1, E_1)$ must be mapped to module $v_2 \in V_2$ of $G_2 = (V_2, E_2)$, the common subconfiguration with maximum common connections is called **maximum common subconfiguration with respect to** v_1 **and** v_2 denoted as $\text{MCS}(v_1, v_2)$ with mapping $f : \widehat{V}_1 \to \widehat{V}_2$ where $\widehat{V}_1 \subseteq V_1$ and $\widehat{V}_2 \subseteq V_2$. If a module pair (v'_1, v'_2) satisfies $v'_1 \in \widehat{V}_1$, $v'_2 \in \widehat{V}_2$ and $f(v'_1) = v'_2$, then $(v'_1, v'_2) \in \text{MCS}(v_1, v_2)$ under $f : \widehat{V}_1 \to \widehat{V}_2$.

Given two graphs G_1 and G_2 rooted with respect to $\tau_1 \in V_1$ and $\tau_2 \in V_2$ respectively,

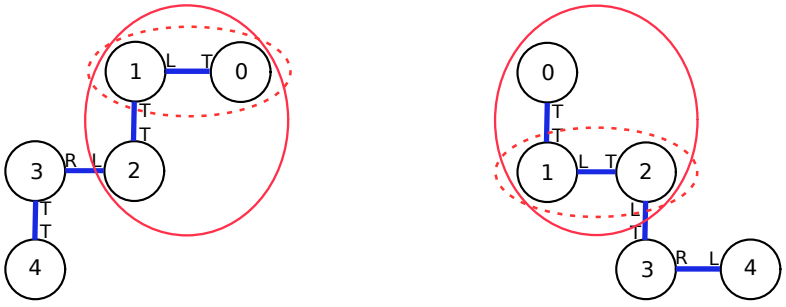


Figure 7.3: Given two SMORES configurations, an example of common subconfiguration between them is circled by "- -" with mapping $1 \to 1$ and $0 \to 2$. The set containing the subgraphs of (a) and (b) circled by "—" is $\text{MCS}(1, 1)$ with mapping $1 \to 1$, $2 \to 0$, and $0 \to 2$.

for module $v_1 \in V_1$ and $v_2 \in V_2$, a common subconfiguration $\{G_1', G_2'\}$ can be constructed where $V_1' = \{v_1, \hat{v}_1^{c_1}\}$, $V_2' = \{v_2, \hat{v}_2^{c_2}\}$ under a subconfiguration mapping $f' : V_1' \to V_2'$ such that $f'(v_1) = v_2$ and $f'(\hat{v}_1^{c_1}) = \hat{v}_2^{c_2}$ if and only if $\text{connect}(v_1, \hat{v}_1^{c_1}) \cong \text{connect}(v_2, \hat{v}_2^{c_2})$ which is called the **feasibility rule**. A stronger feasibility rule can be designed by adding $\text{CN}^{v_1}(c_1) = \text{CN}^{v_2}(c_2)$ then the rule becomes a sufficient condition which is expressed as a form of a function $F((v_1, c_1), (v_2, c_2))$. c_1 is not necessarily to be equal to c_2 because of the symmetry property of modules.

Theorem 1. *Given* $\text{MCS}(v_1, v_2)$ *under mapping* $f : V_1 \to V_2$*, for any module pair* $(v_1', v_2') \in \text{MCS}(v_1, v_2)$ *under* $f : V_1 \to V_2$, $\text{MCS}(v_1', v_2')$ *is equal to* $\text{MCS}(v_1, v_2)$.

According to Theorem 1 and previous analysis, if $F((v_1, c_1), (v_2, c_2))$ is true, then $(\hat{v}_1^{c_1}, \hat{v}_2^{c_2}) \in \text{MCS}(v_1, v_2)$ under $f : V_1 \to V_2$ and $\text{MCS}(\hat{v}_1^{c_1}, \hat{v}_2^{c_2})$ is equal to $\text{MCS}(v_1, v_2)$. Hence, whether computing $\text{MCS}(v_1, v_2)$ is equivalent to computing $\text{MCS}(\hat{v}_1^{c_1}, \hat{v}_2^{c_2})$ depends only on the corresponding feasibility rule. If two modular robot configurations $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, τ_1 and τ_2 are the only root module of G_1 and G_2 respectively, then a common subconfiguration $\{G_1, G_2\}$ under mapping $f : V_1 \to V_2$ must exist and this common subconfiguration is actually $\text{MCS}(\tau_1, \tau_2)$. There may be multiple options of $f : V_1 \to V_2$ because of the symmetry property of the modules. The problem to compute $\text{MCS}(\tau_1, \tau_2)$ can be converted into the problem to compute $\text{MCS}(\hat{\tau}_1^{c_1}, \hat{\tau}_2^{c_2})$ just by checking $F((\tau_1, c_1), (\tau_2, c_2))$. This leads to the recursive solution to compute $\text{MCS}(\tau_1, \tau_2)$ and the subproblem is to solve $\text{MCS}(\hat{\tau}_1^{c_1}, \hat{\tau}_2^{c_2})$ where $c_1 \in \mathcal{C}_d(\tau_1)$ and $c_2 \in \mathcal{C}_d(\tau_2)$. Whichever of these MCSs is equivalent to $\text{MCS}(\tau_1, \tau_2)$ is the solution and the corresponding mapping $f : V_1 \to V_2$ is generated accordingly. For some configurations that may have a pair of root modules while each configuration is only rooted with respect to one root in the library, just need to root this configuration with respect to both of the root modules and compare both of them with that in the library. With this, a bottom-up algorithm to check the isomorphism and map modules simultaneously is developed.

The algorithm takes two rooted configuration graphs as input. Some preliminary checks can be added here, like the number of root modules, the height of trees, the number of

modules in each level, and so on. If they fail these tests, they cannot be isomorphic because there can be two root modules at most and, if two graphs are isomorphic, then they should have the same root modules. This bottom-up algorithm is started from the modules that are leaves of two given configurations and move gradually to their roots. During the process, all possible MCS which may be equivalent to MCS(τ_1, τ_2) are computed and their corresponding mappings are generated. In the end, the one that contains all the modules is the solution and the common subconfiguration mapping tells us how to map each module. The pseudocode is presented in Algorithm 3. For any vertex $v \in V$ with its depth $d(v) > 0$, its parent connected via its connector c is denoted as $\tilde{v}^c$ and the mating connector of $\tilde{v}^c$ as $\tilde{c}'$.

Line 8 — 9: It is possible that (v_1, v_2) or $(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ has been added to some MCSs with respect to some pair of modules. $\mathcal{MCS}_{\mathrm{P}}$ is the set of all MCSs containing $(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ with corresponding mapping f_p . Similarly $\mathcal{MCS}_{\mathrm{C}}$ is the set of all MCSs containing (v_1, v_2) with corresponding mapping f_c.

Line 10 — 28: If $F((\tilde{v}_1^{c_1}, \tilde{c}_1'), (\tilde{v}_2^{c_2}, \tilde{c}_2'))$ is true, then MCS(v_1, v_2) is equal to MCS$(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$. When $\mathcal{MCS}_{\mathrm{P}}$ is empty but $\mathcal{MCS}_{\mathrm{C}}$ is not empty, any MCS$(u_1, u_2) \in \mathcal{MCS}_{\mathrm{C}}$ is equal to MCS(v_1, v_2) from Theorem 1. Hence, as long as $\tilde{v}_1^{c_1} \notin U_1 \wedge \tilde{v}_2^{c_2} \notin U_2$, each MCS(u_1, u_2) is also equal to MCS$(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ and u_1 and u_2 are updated accordingly. Then $\mathcal{MCS}$ is also updated which is to remove the old MCS(u_1, u_2) if existing and add the new MCS(u_1, u_2). Similar to the previous case, when $\mathcal{MCS}_{\mathrm{P}}$ is not empty and $\mathcal{MCS}_{\mathrm{C}}$ is empty, then, as long as $v_1 \notin P_1 \wedge v_2 \notin P_2$, MCS$(p_1, p_2)$ is equal to MCS(v_1, v_2) and p_1 and p_2 are also updated. In addition, it is possible that $v_1 \in P_1 \wedge v_2 \notin P_2$ or $v_1 \notin P_1 \wedge v_2 \in P_2$, then there is no MCS $\in \mathcal{MCS}$ equal to MCS(v_1, v_2) or MCS$(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$. So MCS$(v_1, v_2)$ or MCS$(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ is a new possible MCS which may be equal to MCS(τ_1, τ_2). Thus, add MCS(v_1, v_2) with mapping $f : \{v_1, \tilde{v}_1^{c_1}\} \to \{v_2, \tilde{v}_2^{c_2}\}$ to $\mathcal{MCS}$.

Line 29 — 31: If both $\mathcal{MCS}_{\mathrm{P}}$ and $\mathcal{MCS}_{\mathrm{C}}$ are empty, then MCS(v_1, v_2) or MCS$(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ is a new possible MCS which may be equal to MCS(τ_1, τ_2). Thus, add MCS(v_1, v_2) with mapping $f : \{v_1, \tilde{v}_1^{c_1}\} \to \{v_2, \tilde{v}_2^{c_2}\}$ to $\mathcal{MCS}$.

Line 32 — 51: This is the last case when $F((\tilde{v}_1^{c_1}, \tilde{c}_1'), (\tilde{v}_2^{c_2}, \tilde{c}_2'))$ is true that both $\mathcal{MCS}_{\mathrm{P}}$

Algorithm 3: Matching and Mapping

Input: $G_1 = (V_1, E_1)$ with root τ_1 and $G_2 = (V_2, E_2)$ with root τ_2 where $h(G_1) = h(G_2) = H$
Output: $\text{MCS}(\tau_1, \tau_2)$ with $f : \widehat{V_1} \to \widehat{V_2}$

Create a set $\mathcal{MCS} = \{\text{MCS}(\textsc{Null}, \textsc{Null})\}$ with a trivial mapping $f : \emptyset \to \emptyset$;
Initialize $\mathbf{h} \leftarrow H$;
while $\mathbf{h} > 0$ **do**
  foreach $v_1 \in V_1$ *with* $d(v_1) = \mathbf{h}$ **do**
    $\mathcal{MCS}' \leftarrow \mathcal{MCS}$;
    foreach $v_2 \in V_2$ *with* $d(v_2) = \mathbf{h}$ **do**
      if $F((\tilde{v}_1^{c_1}, \tilde{c}_1'), (\tilde{v}_2^{c_2}, \tilde{c}_2'))$ **then**
        $\mathcal{MCS}_\text{P} = \{\text{MCS}(p_1, p_2) \in \mathcal{MCS}' \; f_p : P_1 \to P_2 \mid (\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2}) \in \text{MCS}(p_1, p_2)\}$;
        $\mathcal{MCS}_\text{C} = \{\text{MCS}(u_1, u_2) \in \mathcal{MCS}' \; f_c : U_1 \to U_2 \mid (v_1, v_2) \in \text{MCS}(u_1, u_2)\}$;
        if $\mathcal{MCS}_\text{P} = \emptyset \wedge \mathcal{MCS}_\text{C} \neq \emptyset$ **then**
          foreach $\text{MCS}(u_1, u_2) \in \mathcal{MCS}_\text{C}$ **do**
            if $\tilde{v}_1^{c_1} \notin U_1 \wedge \tilde{v}_2^{c_2} \notin U_2$ **then**
              Add $(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ to $\text{MCS}(u_1, u_2)$ such that $f_c(\tilde{v}_1^{c_1}) = \tilde{v}_2^{c_2}$;
              $u_1 \leftarrow \tilde{v}_1^{c_1}$ and $u_2 \leftarrow \tilde{v}_2^{c_2}$;
              Update $\mathcal{MCS}$;
            end
          end
        else if $\mathcal{MCS}_\text{P} \neq \emptyset \wedge \mathcal{MCS}_\text{C} = \emptyset$ **then**
          foreach $\text{MCS}(p_1, p_2) \in \mathcal{MCS}_\text{P}$ **do**
            if $v_1 \notin P_1 \wedge v_2 \notin P_2$ **then**
              Add (v_1, v_2) to $\text{MCS}(p_1, p_2)$ such that $f_p(v_1) = v_2$;
              $p_1 \leftarrow v_1$ and $p_2 \leftarrow v_2$;
              Update $\mathcal{MCS}$;
            else if $(v_1 \in P_1 \wedge v_2 \notin P_2) \vee (v_1 \notin P_1 \wedge v_2 \in P_2)$ **then**
              Construct $\text{MCS}(v_1, v_2)$ with $f : \{v_1, \tilde{v}_1^{c_1}\} \to \{v_2, \tilde{v}_2^{c_2}\}$ such that $f(v_1) = v_2$ and $f(\tilde{v}_1^{c_1}) = \tilde{v}_2^{c_2}$;
              Add $\text{MCS}(v_1, v_2)$ to $\mathcal{MCS}$;
            end
          end
        else if $\mathcal{MCS}_\text{P} = \emptyset \wedge \mathcal{MCS}_\text{C} = \emptyset$ **then**
          Construct $\text{MCS}(v_1, v_2)$ with $f : \{v_1, \tilde{v}_1^{c_1}\} \to \{v_2, \tilde{v}_2^{c_2}\}$ such that $f(v_1) = v_2$ and $f(\tilde{v}_1^{c_1}) = \tilde{v}_2^{c_2}$;
          Add $\text{MCS}(v_1, v_2)$ to $\mathcal{MCS}$;
        else
          $\mathcal{MCS}'_\text{P} = \{\text{MCS}(p_1, p_2) \in \mathcal{MCS}_\text{P} \mid v_1 \notin P_1 \wedge v_2 \notin P_2\}$;
          $\mathcal{MCS}'_\text{C} = \{\text{MCS}(u_1, u_2) \in \mathcal{MCS}_\text{C} \mid \tilde{v}_1^{c_1} \notin U_1 \wedge \tilde{v}_2^{c_2} \notin U_2\}$;
          if $\mathcal{MCS}'_\text{P} = \emptyset \wedge \mathcal{MCS}'_\text{C} \neq \emptyset$ **then**
            foreach $\text{MCS}(u_1, u_2) \in \mathcal{MCS}'_\text{C}$ **do**
              Add $(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ to $\text{MCS}(u_1, u_2)$ such that $f_c(\tilde{v}_1^{c_1}) = \tilde{v}_2^{c_2}$;
              $u_1 \leftarrow \tilde{v}_1^{c_1}$ and $u_2 \leftarrow \tilde{v}_2^{c_2}$;
              Update $\mathcal{MCS}$;
            end
          else if $\mathcal{MCS}'_\text{P} \neq \emptyset \wedge \mathcal{MCS}'_\text{C} = \emptyset$ **then**
            foreach $\text{MCS}(p_1, p_2) \in \mathcal{MCS}'_\text{P}$ **do**
              Add (v_1, v_2) to $\text{MCS}(p_1, p_2)$ such that $f_p(v_1) = v_2$;
              $p_1 \leftarrow v_1$ and $p_2 \leftarrow v_2$;
              Update $\mathcal{MCS}$;
            end
          else if $\mathcal{MCS}'_\text{P} \neq \emptyset \wedge \mathcal{MCS}'_\text{C} \neq \emptyset$ **then**
            $\text{MCS}(p_1, p_2) \leftarrow \text{MCS}(u_1, u_2) \cup \text{MCS}(p_1, p_2)$, $\forall (\text{MCS}(p_1, p_2), \text{MCS}(u_1, u_2)) \in \mathcal{MCS}'_\text{P} \times \mathcal{MCS}'_\text{C}$;
            Update $\mathcal{MCS}$;
          end
        end
      end
    end
  end
  $\mathbf{h} \leftarrow \mathbf{h} - 1$;
end

and $\mathcal{MCS}_{\mathrm{C}}$ are not empty. $\mathrm{MCS}(u_1, u_2) \in \mathcal{MCS}_{\mathrm{C}}$ cannot be equal to any $\mathrm{MCS}(p_1, p_2) \in \mathcal{MCS}_{\mathrm{P}}$ for the reason that the algorithm starts from leaves of two given trees and multiple modules may share a parent but multiple modules cannot have the same children. In *Line 33* and *Line 34* the algorithm checks if (v_1, v_2) can be added to any $\mathrm{MCS}(p_1, p_2)$ or $(\tilde{v}_1^{c_1}, \tilde{v}_2^{c_2})$ can be added to any $\mathrm{MCS}(u_1, u_2)$. $\mathcal{MCS}$ is updated according to different conditions and the special one is that when both $\mathcal{MCS}'_{\mathrm{P}}$ and $\mathcal{MCS}'_{\mathrm{C}}$ are not empty, $\mathrm{MCS}(u_1, u_2)$ and $\mathrm{MCS}(p_1, p_2)$ have to be merged with all possible combinations.

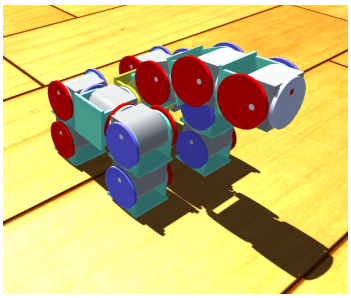
Line 55: Update $\mathbf{h}$ and, in the next iteration, the one-level higher modules will be compared to update $\mathcal{MCS}$. The last group of modules that the algorithm compares are those with depth equal to one.

In the end, there will be multiple $\mathrm{MCS}(\tau_1, \tau_2) \in \mathcal{MCS}$ and multiple $\mathrm{MCS}(q_1, q_2) \in \mathcal{MCS}$ such that $(\tau_1, \tau_2) \in \mathrm{MCS}(q_1, q_2)$. Among all these candidates, the one covering maximum number of modules is the solution of $\mathrm{MCS}(\tau_1, \tau_2)$ with corresponding mapping $f : \widehat{V}_1 \to \widehat{V}_2$. If the roots of both two given configurations are unique, then they are isomorphic if $\widehat{V}_1 = V_1$ (obviously $\widehat{V}_2 = V_2$) and not isomorphic if $\widehat{V}_1 \neq V_1$ (namely $\widehat{V}_2 \neq V_2$).

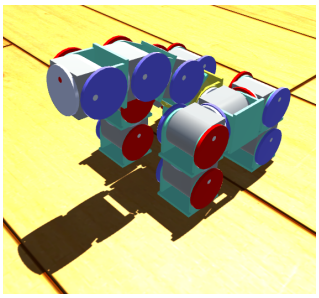
This algorithm can be improved to be more efficient for matching and mapping task. When checking $v_1 \in V_1$ with $d(v_1) = \mathbf{h}$, if $\forall v_2 \in V_2$ with $d(v_2) = \mathbf{h}$, $F((\tilde{v}_1^{c_1}, \tilde{c}'_1), (\tilde{v}_2^{c_2}, \tilde{c}'_2))$ is not true, then these two configurations cannot be isomorphic and the algorithm can stop or continue with next candidate. This algorithm can solve the matching and mapping problem in time $\mathcal{O}(|E_1|^2)$ or $\mathcal{O}(|V_1|^2)$ for the worst case. In reality, the number of connectors a module has is usually small, so the time for a large number of modules should be much smaller than the worst case.

7.5 Test Scenario

The integral algorithm is implemented in Python. The SMORES-EP configuration used in this experiment is a walker with a manipulator shown in Figure 7.4. The corresponding graph representations of them are shown in Figure 7.5. This is a special configuration that has a lot of symmetric parts. For example, the left legs and the right legs are symmetric and the front legs and the back legs are also symmetric.

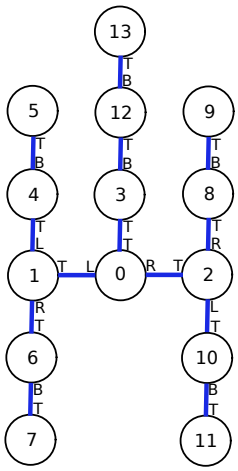


(a)

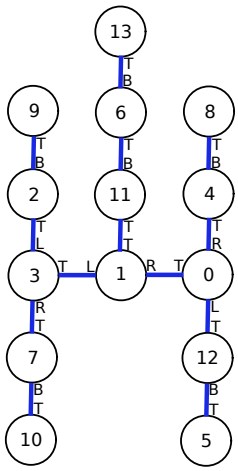


(b)

Figure 7.4: Walker configurations with different labels: (a) the configuration in the library; (b) the new configuration to recognize.



(a)



(b)

Figure 7.5: Walker configuration graphs: (a) the configuration in the library; (b) the new design.

First run Algorithm 1 to discover all the connections in this configuration shown in Figure 7.4b. The sequence to discover all the connections is connect$(0, 1) \rightarrow$ connect$(0, 4) \rightarrow$ connect$(0, 12) \rightarrow$ connect$(1, 11) \rightarrow$ connect$(1, 3) \rightarrow$ connect$(4, 8) \rightarrow$ connect$(12, 5) \rightarrow$ connect$(11, 6) \rightarrow$ connect$(3, 7) \rightarrow$ connect$(3, 2) \rightarrow$ connect$(6,\ 13) \rightarrow$ connect$(2, 9) \rightarrow$ connect$(7, 10)$.

Then the graph $G = (V, E)$ can be built based on these connections. With $G = (V, E)$, the root and all $\text{CN}^v(c)$ for $v \in V$ and $c \in \mathcal{C}$ can be computed by Algorithm 2 and the root module is Module 1. Some preliminary tests can be implemented to filter most of the configurations in the library and the one shown in Figure 7.4a is the closest one. The

Table 7.1: Isomorphic mappings.

$10 \to 5$	$10 \to 5$	$9 \to 5$	$9 \to 5$	$8 \to 5$	$8 \to 5$	$5 \to 5$	$5 \to 5$
$7 \to 4$	$7 \to 4$	$2 \to 4$	$2 \to 4$	$4 \to 4$	$4 \to 4$	$12 \to 4$	$12 \to 4$
$3 \to 1$	$3 \to 1$	$3 \to 1$	$3 \to 1$	$0 \to 1$	$0 \to 1$	$1 \to 1$	$0 \to 1$
$9 \to 7$	$9 \to 7$	$10 \to 7$	$10 \to 7$	$5 \to 7$	$5 \to 7$	$8 \to 7$	$8 \to 7$
$2 \to 6$	$2 \to 6$	$7 \to 6$	$7 \to 6$	$12 \to 6$	$12 \to 6$	$4 \to 6$	$4 \to 6$
$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 0$	$1 \to 0$
$8 \to 9$	$5 \to 9$	$8 \to 9$	$5 \to 9$	$10 \to 9$	$9 \to 9$	$10 \to 9$	$9 \to 9$
$4 \to 8$	$12 \to 8$	$4 \to 8$	$12 \to 8$	$7 \to 8$	$2 \to 8$	$7 \to 8$	$2 \to 8$
$0 \to 2$	$0 \to 2$	$0 \to 2$	$0 \to 2$	$3 \to 2$	$3 \to 2$	$3 \to 2$	$3 \to 2$
$5 \to 11$	$8 \to 11$	$5 \to 11$	$8 \to 11$	$9 \to 11$	$10 \to 11$	$9 \to 11$	$10 \to 11$
$12 \to 10$	$4 \to 10$	$12 \to 10$	$4 \to 10$	$2 \to 10$	$7 \to 10$	$2 \to 10$	$7 \to 10$
$13 \to 13$	$13 \to 13$	$13 \to 13$	$13 \to 13$	$13 \to 13$	$13 \to 13$	$13 \to 13$	$13 \to 13$
$6 \to 12$	$6 \to 12$	$6 \to 12$	$6 \to 12$	$6 \to 12$	$6 \to 12$	$6 \to 12$	$6 \to 12$
$11 \to 3$	$11 \to 3$	$11 \to 3$	$11 \to 3$	$11 \to 3$	$11 \to 3$	$11 \to 3$	$11 \to 3$

graph representation $\overline{G} = (\overline{V}, \overline{E})$ (Figure 7.5a) rooted with respect to the corresponding root module (Module 0), all connection information, and all CN values are stored in the library. Then run Algorithm 3 to match this new configuration $G = (V, E)$ to $\overline{G} = (\overline{V}, \overline{E})$ and map each module $v \in V$ to $\bar{v} \in \overline{V}$. Since there are many symmetric parts, the mapping between G and $\overline{G}$ is not unique. By the configuration recognition algorithm, all isomorphic mappings $f : V \to \overline{V}$ can be obtained. For this example, there are eight different mappings in total shown in Table 7.1.

7.6 Conclusion

This chapter introduces the graph model of modular robots and develops an efficient algorithm for modular robots to do configuration recognition automatically. A new configuration can be discovered by the use of local communication among modules and a graph representation can be generated. Then its root module(s) and all CN values are computed using dynamic programming. Each configuration in the library has a unique representation containing its rooted graph with respect to its root module, all connections, and all CN values. Matching and mapping problem is solved by searching MCS and, if this new configuration is isomorphic to some configuration in the library, all the mapping results will be computed. The algorithm can be adapted to other modular systems easily. as long as local communi-

cation among modules is supported for configuration discovery and equivalent connections can be defined for matching and mapping.

Chapter 8

Morphology Transformation

This chapter presents the morphology transformation strategy for the SMORES-EP system that can also be extended to other similar self-reconfigurable modular robots. This enables the self-adaption of these systems to different scenarios. This chapter excerpts heavily from [73] and [78]. Credit is due to Michael Whitzer, Qian Lin, and Hyun Kim, who contributed significantly to this work.

The capability to do morphology transformation distinguishes modular robots from other robotic systems. This allows modular robots to handle a much wider range of tasks and they can reconfigure themselves as needed. SMORES-EP is a hybrid modular robotic system that can form kinematic chains to address a variety of manipulation and locomotion scenarios, yet can reconfigure using a mobile reconfiguration strategy. There are two types of self-adaption activities: 1. self-assembly that is to allow multiple separated modules to form target kinematic chains; 2. self-reconfiguration that is to transform a morphology into a different one. For self-assembly, a desired kinematic topology can be mapped onto a planar pattern with the optimal module assignment based on the modules' locations, then the mobile reconfiguration assembly process can be executed in parallel. For self-reconfiguration, the initial configuration and the goal configuration have to be compared, and then find the required sequence of reconfiguration actions. The reconfiguration actions can be executed in a distributed and parallel manner so that each module can efficiently finish its reconfiguration

task which results in a global reconfiguration of the overall morphology.

8.1 Introduction

Self-reconfigurable modular robots are able to adapt their morphologies to many different activities, handle hardware and software failures by reconfiguring themselves, which relies on the *self-reconfiguration planning*. This self-reconfiguration ability enables a modular robotic system to change the arrangement of modules from one arbitrary configuration to another. While the reconfiguration planning problem is well defined, the problem is usually difficult to solve because of the physical constraints particular to each self-reconfigurable robotic system. For example, the modules are usually designed to be simple to manufacture and low-cost so that each module only has a limited number of actuators and connectors. This results in limited motion ability and often complex system constraints. In addition, the number of possible arrangements for a cluster of modules grows exponentially with the number of modules which makes the planning problem intractable to find optimal solutions with naïve brute-force techniques.

As introduced in Part I, the majority of self-reconfigurable robots are typically divided into three main types: chain-type, lattice-type, and mobile-type with some models that are hybrid among these types. Lattice-type self-reconfigurable robots sit nominally on a lattice and reconfigure between neighboring lattice positions. There are already many reconfiguration planning algorithms for lattice-type robots. However, it is difficult for lattice-type modular robots to generate some dynamic locomotion and generic manipulation. In contrast, chain-type self-reconfigurable robots are particularly well suited for locomotion and manipulation. Numerous chain-type modular robotic systems have been developed in the past few years. Self-reconfiguration with this type of robot is often a difficult and time-consuming task in which closed chains have to be formed [135]. The third type, mobile-type robots, use the environment to move between modules as they reconfigure. The system are usually composed of a large number of identical robots that can move around on the flat ground and connect together to form different planar shapes.

Self-assembly is related to self-reconfiguration for modular robots which is most related

to the mobile-style reconfiguration as pieces are assembled as separate units. This swarming behavior is common in nature that groups of individuals can join together to overcome the limited capability of individuals, especially for insects who often need to collaborate in large groups to accomplish tasks. This collective has inspired similar approaches for robotic systems, such as modular robots that are composed of numerous simple building blocks, or modules, that can be joined into different connected morphologies depending on task requirements. Determining the motion sequence of a cluster of modules to form a goal morphology is called *self-assembly planning*.

A particularly useful self-assembly example of collaborative behavior can be shown for systems that can achieve mobile reconfiguration. Every individual module can locomote around an area, however, it may discover that it cannot cross a gap that is larger than one module. Instead, multiple modules can join to form a snake configuration making it long enough to cross over. Other applications include reaching tall spaces or rapid simultaneous exploration. All of these behaviors are similar to the behaviors shown in ants.

The goal is to allow modular robots to self-assemble or self-reconfigure into desired morphologies to perform complex tasks. There are several challenges needed to be addressed:

1. Efficiency is important, especially for modular robotic systems that include a large number of modules;

2. Many physical constraints have to be considered for real hardware applications;

3. Accurate docking is required.

SMORES-EP is a hybrid modular robotic system that can achieve all three types of motions for reconfiguration. As mentioned, each module is a four-DOF (*LEFT DOF*, *RIGHT DOF*, *PAN DOF*, and *TILT DOF*) system with four connectors (*LEFT Face* or **L**, *RIGHT Face* or **R**, *TOP Face* or **T** and *BOTTOM Face* or **B**). There are seventeen ways to connect two modules to achieve different kinematics functionality. This chapter focuses on mobile-style reconfiguration that is shown to be a reliable and efficient strategy for modular robots. Robot morphologies are modeled as graphs introduced in Chapter 7. The self-reconfiguration

planner can compare and decompose the initial configuration and the goal configuration efficiently so that a sequence of reconfiguration actions can be generated in such a way that modules can act in a reasonable order to achieve the goal configuration efficiently. For self-assembly, given the current locations of all modules, every individual is mapped to a module in the target configuration in an optimal way by solving a task assignment problem. Then the assembly actions can be executed in parallel for efficiency while satisfying physical hardware constraints. This framework can also speed up the self-reconfiguration process by executing the sequential reconfiguration actions in parallel. The effectiveness and robustness of the framework in this chapter are demonstrated on the physical hardware. These proposed algorithms can be easily extended to other mobile-type modular robots.

8.2 Related Work

Lattice-type modular robots usually offer simpler reconfiguration, as the motions of modules are constrained in a discrete set of locations. Self-reconfiguration planning for these robots includes work from [14, 30, 99, 138]. Chain-type modular robots can be more versatile in that modules are able to do motions in continuous space. However, it is usually difficult for these robots to do reconfiguration due to the difficulty in control. Closed chains are formed during the reconfiguration process, such as [15, 100, 164]. These works only consider single-DOF modules with uniform connections. More complex chain-style reconfiguration tasks are addressed in [4, 48, 49]. While lattice-style reconfiguration and chain-style reconfiguration have been studied in depth, the mobile form of reconfiguration has not been.

Self-assembly is related to self-reconfiguration for modular robots. Stochastic self-assembly algorithms are for modules that are not self-actuated but use external actuation, such as fluid flow [63, 90, 156]. Static planar structures can be assembled by building blocks which are supplied by mobile robots [153]. Self-assembly solutions for mobile-type modular robots typically use modules that do not have non-holonomic constraints and aim for planar structures [69, 117, 123]. An approach to solve configuration formation is presented in [27] but in sequential manner which makes the formation process slow. Assembling structures in 3-dimensional space is shown in [142, 154] but does not deal with the physical constraints

of land-mobile platforms. The self-assembly work in this chapter differs from structural self-assembly in that the assembly goal is to build a movable kinematic topology, such as a multi-limbed form. The target morphologies are similar to those built by chain-type modular robots. For example, reconfiguration for PolyBot in a kinematic topology was presented in [164]. The proposed parallel self-assembly framework can also be used to make the sequential reconfiguration actions more efficient by executing these actions in parallel.

8.3 Self-reconfiguration Planning

The self-reconfiguration planning problem can be stated as: Given an arbitrary initial configuration and goal configuration, find the actions required for the system to transform the initial configuration to the goal configuration. Different atomic reconfiguration actions can be defined for different modular robotic systems. Usually there are two atomic reconfiguration actions: *Docking* and *Undocking*. *Docking* means connecting two connectors and *Undocking* means disconnecting an existing connection. Different modular robots have different procedures to execute these two atomic actions. In particular, for the SMORES-EP modular robotic system, a *Docking* action requires two modules to move their involved connectors to be in proximity, align these two faces, and activate all corresponding magnets. Similarly, an *Undocking* action requires two modules to deactivate all magnets of both involved connectors.

An outline of the reconfiguration planning algorithm follows. The first step is to find the root module of the initial configuration as in Section 7.4.2, and then figure out the root module of the goal configuration. From here, a novel way to decompose the initial configuration and the goal configuration into multiple subconfigurations can be applied. These subconfigurations can then be mapped between the initial and the goal configuration. This mapping is computed by iteratively adding virtual modules and virtual connections to these subconfigurations. After the connection and disconnection actions are determined, the locomotion plans for modules to move to their required positions for docking can then be implemented with standard 2D path planning approaches. Commonly used nomenclature is listed in the following:

G_i	Initial Configuration
G_g	Goal Configuration
$\overline{G}_i$	Initial Subconfiguration in MCS
$\overline{G}_g$	Goal Subconfiguration in MCS
$\widehat{G}_i$	Initial Subconfiguration not in MCS
$\widehat{G}_g$	Goal Subconfiguration not in MCS
G'_i	Initial Configuration after Virtual Module Operation
G'_g	Goal Configuration after Virtual Module Operation
τ_i	Initial Configuration Root Module
τ_g	Goal Configuration Root Module
τ_i^α	αth Subconfiguration Root of G_i
τ_g^β	βth Subconfiguration Root of G_g
$\mathcal{M}$	Virtual Module

8.3.1 Configuration Decomposition

Given the initial and the goal configuration, first decompose them into multiple subconfigurations. For efficiency, the goal is to find the decomposition that shares the most connections (edges in the graph) between the initial and the goal configuration. Reconfiguration actions (*Docking* and *Undocking*) are usually hard to execute and also time-consuming. Hence, one goal of the algorithm is to minimize the number of Docking and Undocking actions. In addition, it is hard to change the height of a vertex in the graph, for example, moving a module which is a leaf vertex of a configuration to a position close to the root requires more actions, so minimizing these changes will make the physical reconfiguration more efficient as well.

Given any two modular robot configurations $G_i = (V_i, E_i)$ and $G_g = (V_g, E_g)$, their root modules τ_i and τ_g can be computed in $\mathcal{O}(|V_i|)$ and $\mathcal{O}(|V_g|)$ respectively, then $\text{MCS}(\tau_i, \tau_g)$ under mapping $f : \overline{V}_i \to \overline{V}_g$ where $\overline{V}_i \subseteq V_i$ and $\overline{V}_g \subseteq V_g$ can be computed efficiently. These two subconfigurations $\overline{G}_i = (\overline{V}_i, \overline{E}_i)$ and $\overline{G}_g = (\overline{V}_g, \overline{E}_g)$ contained in $\text{MCS}(\tau_i, \tau_g)$ are isomorphic so that there is no need to reconfigure these modules. If subtracting subconfigurations $\overline{G}_i = (\overline{V}_i, \overline{E}_i)$ from $G_i = (V_i, E_i)$ without keeping boundaries, a graph $\widehat{G}_i = (\widehat{V}_i, \widehat{E}_i)$ composed of multiple unconnected subgraphs is generated. Similar operations can be applied to the goal configuration $G_g = (V_g, E_g)$ to generate $\widehat{G}_g = (\widehat{V}_g, \widehat{E}_g)$.

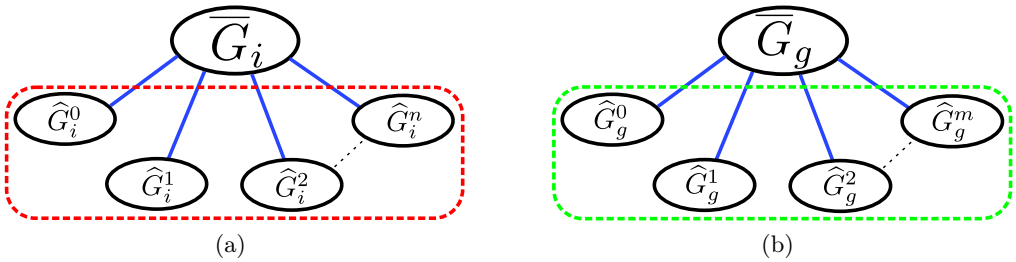


Figure 8.1: (a) Configuration decomposition for $G_i = (V_i, E_i)$ and the subconfiguration encircled by "- -" is $\widehat{G}_i = (\widehat{V}_i, \widehat{E}_i)$. (b) Configuration decomposition for $G_g = (V_g, E_g)$ and the subconfiguration encircled by "- -" is $\widehat{G}_g = (\widehat{V}_g, \widehat{E}_g)$.

The process is shown in Figure 8.1 where $\widehat{G}_i = \left\{\widehat{G}_i^\alpha = (\widehat{V}_i^\alpha, \widehat{E}_i^\alpha) | \alpha = 0, 1, 2, \cdots, n\right\}$ and $\widehat{G}_g = \left\{\widehat{G}_g^\beta = (\widehat{V}_g^\beta, \widehat{E}_g^\beta) | \beta = 0, 1, 2, \cdots, m\right\}$. This process is defined as a *configuration decomposition* for $G_i = (V_i, E_i)$ and $G_g = (V_g, E_g)$ with respect to the root module pair τ_i and τ_g written as $\mathcal{CD}(G_i, \tau_i, G_g, \tau_g)$. This configuration decomposition can be finished in time $\mathcal{O}(|V_i|^2)$ or $\mathcal{O}(|E_i|^2)$ and, in reality, a SMORES-EP module has only four connectors so the time for a large number of modules should be much smaller than the worst case as discussed in Section 7.4.3. Then the problem is to do reconfiguration planning from $\widehat{G}_i = (\widehat{V}_i, \widehat{E}_i)$ to $\widehat{G}_g = (\widehat{V}_g, \widehat{E}_g)$.

8.3.2 Module Mapping

Applying configuration decomposition for the initial and the goal configuration, $G_i = (V_i, E_i)$ and $G_g = (V_g, E_g)$, modules involved in $\text{MCS}(\tau_i, \tau_g)$ are mapped under $f : \overline{V}_i \to \overline{V}_g$ where $\overline{V}_i \subseteq V_i$ and $\overline{V}_g \subseteq V_g$, so $\widehat{G}_i = (\widehat{V}_i, \widehat{E}_i)$ and $\widehat{G}_g = (\widehat{V}_g, \widehat{E}_g)$ can be generated respectively, both of which are composed of multiple subconfigurations. The connection between $\overline{G}_i = (\overline{V}_i, \overline{E}_i)$ and subconfiguration $\widehat{G}_i^\alpha = (\widehat{V}_i^\alpha, \widehat{E}_i^\alpha)$ in $\widehat{G}_i = (\widehat{V}_i, \widehat{E}_i)$ for $\alpha = 0, 1, \cdots, n$ is denoted as $\text{connect}(u, \tau_i^\alpha)$ where $u \in \overline{V}_i$ and $\tau_i^\alpha \in \widehat{V}_i^\alpha$. The vertex τ_i^α is called the *subconfiguration root* for $G_i = (V_i, E_i)$ and there are n subconfiguration roots in total. Similarly there are m subconfiguration roots for $G_g = (V_g, E_g)$ denoted as τ_g^β where $\beta = 0, 1, \cdots, m$.

Then replace $\overline{G}_i = (\overline{V}_i, \overline{E}_i)$ with a virtual module $\mathcal{M}$ and replace $\text{connect}(u, \tau_i^\alpha)$ with a

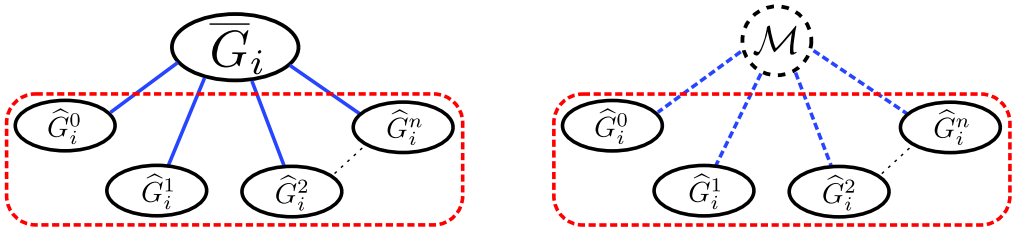


Figure 8.2: Replace $\overline{G}_i = (\overline{V}_i, \overline{E}_i)$ with virtual module $\mathcal{M}$ and replace the connection between $\overline{G}_i = (\overline{V}_i, \overline{E}_i)$ and every $\widehat{G}_i^\alpha = (\widehat{V}_i^\alpha, \widehat{E}_i^\alpha)$ with a virtual connection.

virtual connection connect$(\mathcal{M}, \tau_i^\alpha)$ defined as

$$\text{connect}(\mathcal{M}, v) = \{\text{Face} : \textsc{Null}, \text{Face2Con} : \textsc{Null}, \text{Orientation} : \textsc{Null}\} \tag{8.1}$$

in which v represents a module. All virtual connections are equivalent. The new corresponding modular robot configuration with a virtual module and some virtual connections is written as $G_i' = (V_i', E_i')$ where $V_i' = V_i \setminus \overline{V}_i \cup \{\mathcal{M}\}$ as shown in Figure 8.2. This operation is called the *Virtual Module Operation*.

These procedures can be applied to the goal configuration $G_g = (V_g, E_g)$ and the corresponding $G_g' = (V_g', E_g')$ is generated shown in Figure 8.3. Applying configuration decomposition on $G_i' = (V_i', E_i')$ and $G_g' = (V_g', E_g')$ with respect to virtual module $\mathcal{M}_i$ and $\mathcal{M}_g$, MCS$(\mathcal{M}_i, \mathcal{M}_g)$ under mapping $f : \overline{V}'_i \to \overline{V}'_g$ where $\overline{V}'_i \subseteq V_i'$ and $\overline{V}'_g \subseteq V_g'$ can be computed. The solution to MCS$(\mathcal{M}_i, \mathcal{M}_g)$ may not be unique. In addition to these two virtual modules,

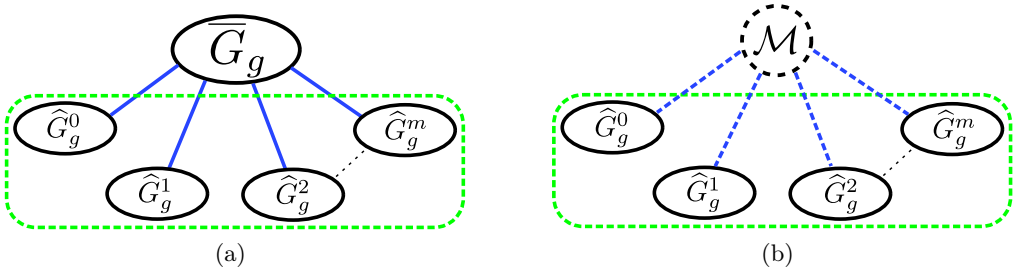


Figure 8.3: Replace $\overline{G}_g = (\overline{V}_g, \overline{E}_g)$ with virtual module $\mathcal{M}$ and replace the connection between $\overline{G}_g = (\overline{V}_g, \overline{E}_g)$ and every $\widehat{G}_g^\alpha = (\widehat{V}_g^\alpha, \widehat{E}_g^\alpha)$ with a virtual connection.

each vertex $u \in \overline{V}'_i$ is mapped to a unique module $v \in \overline{V}'_g$. After configuration decomposition $\mathcal{CD}(G'_i, \mathcal{M}_i, G'_g, \mathcal{M}_g)$, $\widehat{G}'_i = (\widehat{V}'_i, \widehat{E}'_i)$ and $\widehat{G}'_g = (\widehat{V}'_g, \widehat{E}'_g)$ are generated respectively. Repeat the virtual module operation for $\widehat{G}'_i = (\widehat{V}'_i, \widehat{E}'_i)$ and $\widehat{G}'_g = (\widehat{V}'_g, \widehat{E}'_g)$, and two new modular robot configurations with virtual modules and virtual connections are generated.

Mapping is completed by repeating this process until every module in V_i has been mapped with a unique module in V_g. If assuming that configuration decomposition is applied N times, then N mappings $f_1, f_2, \cdots, f_N$ are computed in order. This mapping process maintains the vertex height between configurations as much as possible and also keeps most of the common topology connections so that fewer reconfiguration actions are needed. For the worst case, the mapping process has to do configuration decomposition $\lceil |V_i|/2 \rceil + 1$ times (at least two modules can be mapped after each configuration decomposition except for the first and the last configuration decomposition and $\lceil x \rceil$ maps x to the least integer greater than or equal to x), so the time complexity is $\mathcal{O}(|V_i|^3)$. Again, for a large number of modules, in reality, the mapping process should be much faster than the worst case.

8.3.3 Reconfiguration Actions

Once the mapping process is done, the corresponding reconfiguration actions can be determined. Assume there are N mappings $f_t : {}^tV_i \rightarrow {}^tV_g$, $t = 1, 2, \cdots, N$ computed in order, then a mapping $f : V_i \rightarrow V_g$ that maps all modules from the initial configuration $G_i = (V_i, E_i)$ to the goal configuration $G_g = (V_g, E_g)$ can be obtained by the combination of these mappings while excluding virtual module mapping ($\mathcal{M}_i \rightarrow \mathcal{M}_g$). This mapping is one-to-one and onto and the inverse of the mapping is $f^{-1} : V_g \rightarrow V_i$. Thus, the reconfiguration actions can be computed by iterating modules in $G_i = (V_i, E_i)$ from leaves to the root.

Recall that for a modular robot configuration $G = (V, E)$ rooted at τ, for any vertex $v \in V$ with depth $d(v) > 0$, its parent connected via its connector c is denoted as $\tilde{v}^c$ and the mating connector of $\tilde{v}^c$ as $\tilde{c}$. Given the mapping $f : V_i \rightarrow V_g$, each $v_i \in V_i$ is mapped to a unique $v_g \in V_g$, and $\tilde{v}_i^{c_i}$ and $\tilde{v}_g^{c_g}$ are their parents respectively. Similarly, with the inverse mapping $f^{-1} : V_g \rightarrow V_i$, $\tilde{v}_g^{c_g} \in V_g$ is also mapped to a unique $v'_i \in V_i$.

If module pair (v_i, v_g) and $(\tilde{v}_i^{c_i}, \tilde{v}_g^{c_g})$ are in any MCS during the module mapping process, then connect$(v_i, \tilde{v}_i^{c_i}) \cong$ connect$(v_g, \tilde{v}_g^{c_g})$ and there is no need to reconfigure. Otherwise, the reconfiguration action is undocking v_i from $\tilde{v}_i^{c_i}$ by removing connect$(v_i, \tilde{v}_i^{c_i})$ and docking v_i with v_i' by constructing connect(v_i, v_i') which should be equivalent to connect$(v_g, \tilde{v}_g^{c_g})$ in $G_g(V_g, E_g)$.

Once all modules except subconfiguration roots for $G_i = (V_i, E_i)$ and modules in MCS(τ_i, τ_g) are visited, $\widehat{G}_i = (\widehat{V}_i, \widehat{E}_i)$ has reconfigured into $\widehat{G}_g = (\widehat{V}_g, \widehat{E}_g)$ by executing reconfiguration actions from leaves to subconfiguration roots. This enables us to pick one solution to MCS$(\mathcal{M}_i, \mathcal{M}_g)$ in the module mapping process freely since the module should be easy to maneuver when a reconfiguration action is applied. Then execute Algorithm 3 — the *Matching and Mapping* algorithm — to check if the new configuration is isomorphic to the goal configuration. If not, continue iterating unvisited modules and executing reconfiguration actions. This process can be done in time $\mathcal{O}(|V_i|)$.

8.3.4 Hardware Execution

To implement the reconfiguration plan described above with SMORES-EP, modules must undock from their initial positions in the current configuration, safely navigate them to their final positions in the goal configuration, and then dock to the appropriate modules.

The environment in which the modules will reconfigure can be described with a discrete representation. The graph representation of the environment and the reconfiguration plan can then be used to sequentially generate trajectories that safely navigate the modules to their new reconfigured positions in the goal configuration. These trajectories can be generated through graph search techniques such as A* [43].

When generating the trajectory for each module, the modules not involved in the current reconfiguration action will be represented as static obstacles in the discrete environment. It may be the case that the current reconfiguration action requires the motion of other modules to create appropriate free space. Given the full discrete environment and system state knowledge, a state machine can be used to identify when additional free space is required, and initiate the motions of the occluding modules to enable the current reconfiguration

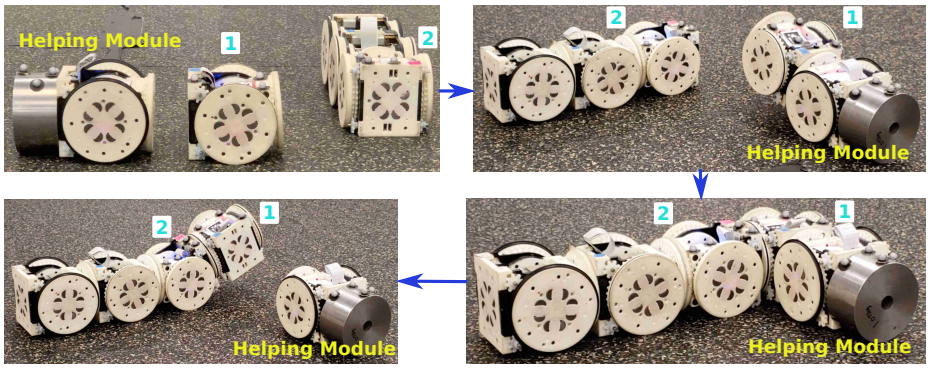


Figure 8.4: A helping module docks with Module 1 BOTTOM Face and lifts it up so that LEFT Face of Module 1 can be aligned with BOTTOM Face of Module 2, then carry Module 1 to the location to finish the docking action.

action.

Since locomotion on the ground is achieved by rotating the LEFT and RIGHT Faces, aligning the orientation of these faces with a stationary mating face could be problematic since their orientation is coupled to translation. When a mobile module is docking to a stationary one, if the face on the stationary module is a LEFT, RIGHT, or TOP Face, the stationary face can rotate to align the orientation appropriately. There are two special docking action cases in which this cannot be done: 1. Dock either a LEFT Face or a RIGHT Face with a mating BOTTOM Face; 2. Dock BOTTOM Face with a mating BOTTOM Face with Orientation attribute being 1. In these cases, the BOTTOM Face cannot rotate as the SMORES-EP modules have fixed BOTTOM faces. For these two cases, a helping module is utilized. As mentioned, a helping module is also a SMORES-EP module with some payload attached to its BOTTOM Face. The helping module can dock with and lift the mobile module so that the face is no longer coupled with the ground and can be orientated appropriately. A proof-of-concept demo of this behavior is shown in Figure 8.4. The detailed usage of helping modules is already discussed in Chapter 6.

8.4 Parallel Self-assembly Planning

Assume there is a team of modules $M = \{m_1, m_2, \cdots, m_n\}$ in the Euclidean space $\mathbb{R}^2$. The state of a module $m_i \in M$ is defined as $p_i = [x_i, y_i, \theta_i]^\intercal$ where $o_i = [x_i, y_i]^\intercal$ is the location

of the center of m_i and θ_i is the orientation of m_i. Then the distance between module m_i and m_j can be derived as $\|o_i - o_j\|$. Every SMORES-EP module is a cube with a side length of w. The assembly goal is a SMORES-EP tree topology configuration $G = (V, E)$ where $|V| = n$. Not all kinematic topology can be built by self-assembly process. Only the kinematic topology that can be unfolded onto a plane can be achieved by a bunch of separated modules on the ground.

Definition 5. The target kinematic topology that can be self-assembled by separated modules is a modular robot configuration $G = (V, E)$ that can be fully unfolded to a plane satisfying:

1. G is a connected graph;
2. The Euclidean distance between two adjacent modules is w;
3. The center of every module occupies a unique location.

We are not interested in all topologies that satisfy the constraints in Definition 5. Some topologies are less useful for executing tasks. For example, the connection in Figure 7.1b shows the two BOTTOM Faces are connected with *Orientation* 1. This arrangement unnecessarily constrains the relative orientation between two modules so it is not considered in this work. In addition, the target topology is also meant to satisfy hardware constraints, such as the connector and actuator limitations when lifting many modules.

The modules are at arbitrary locations with constraint that the distance between any pair of modules m_i and m_j denoted as d_{ij} is greater than w. The kinematic topology self-assembly problem is stated: Given a target kinematic topology $G = (V, E)$ and a team of n modules $M = \{m_1, m_2, \cdots, m_n\}$ where $n = |V|$, find a sequence of collision-free assembly actions to form the target kinematic topology.

8.4.1 Task Assignment

In order to self-assemble n separated modules into a target kinematic topology, each module has to be mapped to a role in the target. This problem is modeled as a task assignment prob-

lem, finding the optimal assignment solution among $n!$ different assignments with respect to some cost function.

Given a target kinematic topology $G = (V, E)$, first check if it satisfies the requirements in Definition 5 in order to be self-assembled and, if so, fully unfold it onto the ground. The root module τ of G can be computed by Algorithm 2 in linear time. Then the state of each module $v_i \in V$ with respect to this root module τ denoted as $\bar{p}_i$ after fully unfolding G can be computed in breadth-first search order starting from τ. From Section 6.2.2, the state of a module is fully determined if the state of its parent module and the involved connection are known. If module v_i is the parent of module v_j, then, in breadth-first search order, when visiting v_j, the state of v_i with respect to τ should already be known that is $\bar{p}_i = \left[\bar{x}_i, \bar{y}_i, \bar{\theta}_i\right]^\intercal$. The state of m_j with respect to m_i denoted as $\bar{p}_{ji}$ is determined by the involved connectors. Then the state of v_j with respect to τ is

$$\bar{p}_j = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \bar{p}_{ji} + \bar{p}_i \tag{8.2}$$

in which $R = \begin{bmatrix} \cos\bar{\theta}_i & -\sin\bar{\theta}_i \\ \sin\bar{\theta}_i & \cos\bar{\theta}_i \end{bmatrix}$.

Given a set of n modules M, each $m \in M$ needs to be mapped by a module $v \in V$ in an optimal way. In this task assignment problem, the objective is to minimize the total distance that all modules have to travel in order to assemble G. First, the center location of all modules can be defined as $o_c = [x_c, y_c]^\intercal$ where $x_c = \sum_{i=1}^{n} x_i/n$ and $y_c = \sum_{i=1}^{n} y_i/n$. Then the root module is selected as

$$m_\tau = \arg\min_{m_i \in M} \|o_i - o_c\| \tag{8.3}$$

The state of every module $m_i \in M$ with respect to m_τ denoted as $\tilde{p}_i$ can be computed simply with a rigid body transformation. Obviously $\tilde{p}_\tau = [0, 0, 0]^\intercal$ that is the state of m_τ with respect to itself. Recall that $\bar{o}_i$ is the location of the center of v_i with respect to

$\tau \in V$. Given m_τ is mapped to τ, namely $\|\tilde{o}_\tau - \bar{o}_\tau\| = 0$, the distance between every pair of $m_i \in M \setminus \{m_\tau\}$ and $v_j \in V \setminus \{\tau\}$ is simply $\|\tilde{o}_i - \bar{o}_j\|$ which is the cost of the task — moving to location of v_j — for module m_i. Other factors can also be included in the cost besides distance, such as the orientation. The optimal task assignment problem can be solved by Kuhn-Munkres algorithm or other concurrent assignment and planning of trajectories algorithms in polynomial time [146]. The output is a one-to-one and onto mapping $f : V \to M$ that is used later by a motion planner to generate the assembly sequence.

8.4.2 Parallel Assembly Actions

With mapping $f : V \to M$, the assembly sequence from the root to the leaves of G can be computed. In each step, the modules in M mapped to the modules in the target kinematic topology G at the same depth can be executed in a parallel manner. Let $d(G)$ be the depth of the rooted graph G. An assembly action is a tuple of the form (m_i, c_i, m_j, c_j) which means connect m_i's connector c_i with m_j's connector c_j. The parallel assembly algorithm is shown in Algorithm 4. In every iteration, first fetch all modules in the current depth from the target topology, then, with the optimal mapping $f : V \to M$, the corresponding moving

Algorithm 4: Parallel Assembly

Input: Target kinematic topology $G = (V, E)$ with root τ and depth $d(G)$
$f : V \to M$
Output: Parallel Assembly Sequence $\mathcal{A}$

```
d ← 1;
while d ≤ d(G) do
    Create empty action queue A;
    V̄ ← {v ∈ V | d(v) = d};
    for v ∈ V̄ do
        m ← f(v);
        m̃^c ← f(ṽ^c);
        A.enqueue((m, c, m̃^c, c̃′));
    end
    A.enqueue(A);
    d ← d + 1;
end
```

modules (e.g. m) are determined as well as their parent modules (e.g. $\tilde{m}^c$). Then a group of assembly actions A composed of $(m, c, \tilde{m}^c, \tilde{c}')$ can be derived.

For each group of assembly actions $A \in \mathcal{A}$, all actions can be executed in parallel except when multiple modules are docking with different connectors of the root module. The group of assembly actions A with the root module involved is separated into two subgroups: one group contains the actions for LEFT Face and RIGHT Face of the root module which can be executed first. The other group contains the actions for TOP Face and BOTTOM Face of the root module which are executed later. This is because the root module is not fixed to the ground and it is hard to ensure all modules to be attached can approach the root module simultaneously. For each assembly action $a = (m_i, c_i, m_j, c_j) \in \mathcal{A}$, the docking strategy from Chapter 6 can be applied. A square grid environment is generated once the root module is determined, with the root module at the center. The grid serves as a routing graph for navigating modules. Similar to the self-reconfiguration process, when c_i is either LEFT Face or RIGHT Face, a helping module is required to finish this assembly action. In this process, add a new assembly action $(m_{\mathrm{H}}, \mathbf{T}, m_i, \bar{c}_i)$ where m_{H} is a helping module and $\bar{c}_i$ is LEFT Face if c_i is RIGHT Face, or the vice versa.

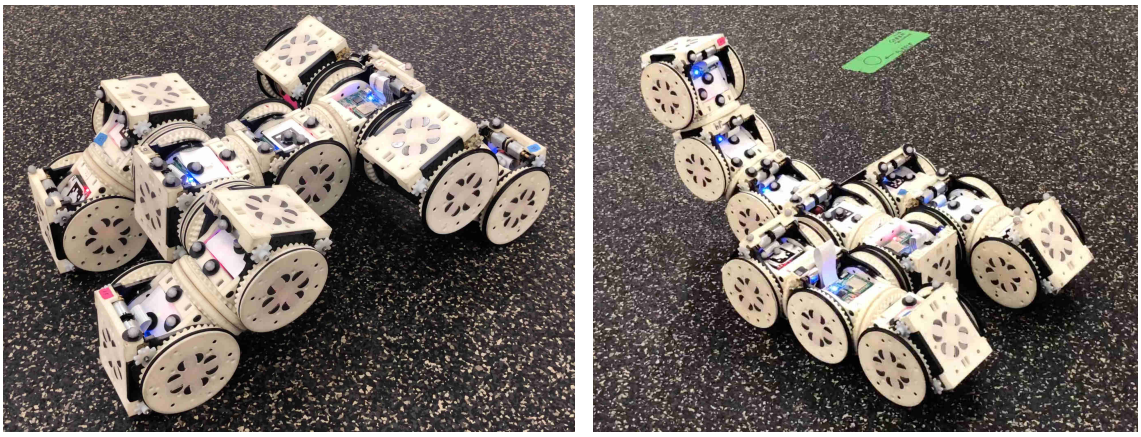
8.5 Experiments

The algorithms presented in this chapter are demonstrated with the SMORES-EP hardware platform. Several self-reconfiguration tasks and self-assembly tasks are shown in this section.

8.5.1 Self-reconfiguration

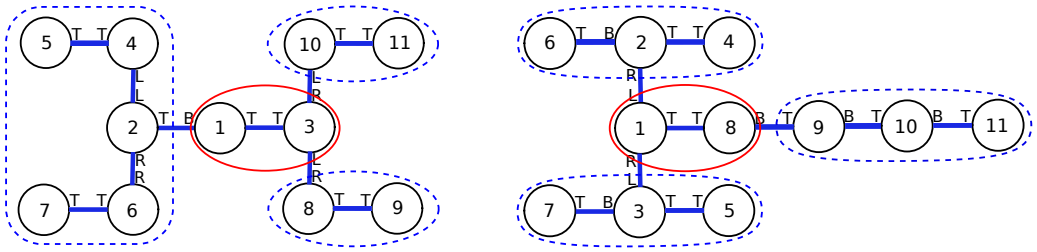
Task 1: Walker → Mobile Manipulator

The task is to reconfigure a cluster of SMORES-EP modules from a walker (Figure 8.5a) into a mobile vehicle with an arm (Figure 8.5b) with eleven modules. The initial and the goal graph representation are shown in Figure 8.6 where τ_i is Module 1 and τ_g is Module 1 respectively. For $G_i = (V_i, E_i)$ and $G_g = (V_g, E_g)$, $\mathrm{MCS}(\tau_i, \tau_g)$ only contains two modules under mapping $1 \to 1$ and $3 \to 8$. The result of the virtual module operation is shown in Figure 8.7. In addition to some equivalent virtual connections, there are two common



(a) (b)

Figure 8.5: Reconfigure a walker configuration (a) into a mobile vehicle with an arm configuration (b) in which eleven SMORES-EP modules are involved.

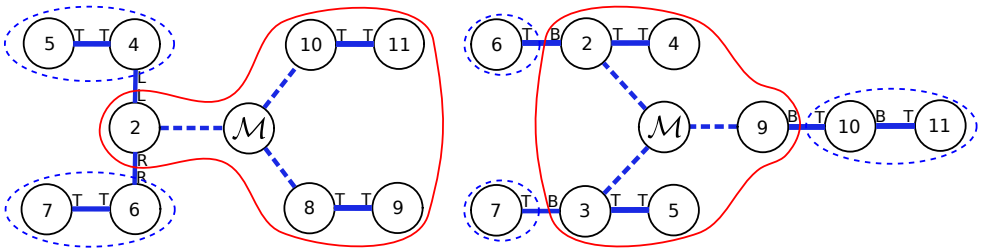


(a) (b)

Figure 8.6: The graph representation of the walker configuration (a) and the graph representation of the mobile manipulator configuration (b) are shown. MCS$(1,1)$ is encircled by "—" under mapping $1 \to 1$ and $3 \to 8$. After removing MCS$(1,1)$, there are three unconnected subgraphs in both the current initial configuration and the current goal configuration which are encircled by "- -".

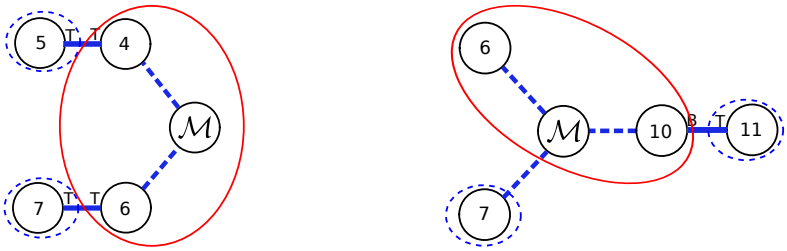
connections in MCS$(\mathcal{M}, \mathcal{M})$. The rest of module mapping process is shown in Figure 8.8 and Figure 8.9. There are only virtual connections in MCS$(\mathcal{M}, \mathcal{M})$, each of which requires reconfiguration actions. The final mapping $f : V_i \to V_g$ is $1 \to 1$, $3 \to 8$, $9 \to 5$, $8 \to 3$, $2 \to 9$, $11 \to 4$, $10 \to 2$, $4 \to 6$, $6 \to 10$, $5 \to 7$, and $7 \to 11$ and the corresponding reconfiguration actions are shown in Table 8.1.

The hardware execution of this plan is shown in Figure 8.10. First, Module 6 and Module 7 have to move away so that Module 5 can move to dock with Module 8. Then Module 7 moves to dock with BOTTOM Face of Module 6, and Module 4 undocks from Module 2 and



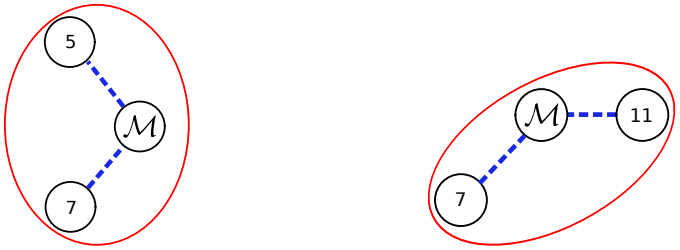
(a) (b)

Figure 8.7: The new graph representation of the walker configuration (a) and the new graph representation of the mobile manipulator configuration (b) are shown. MCS($\mathcal{M}, \mathcal{M}$) is encircled by "—" under mapping $\mathcal{M} \to \mathcal{M}$, $9 \to 5$, $8 \to 3$, $2 \to 9$, $11 \to 4$, and $10 \to 2$. After removing MCS($\mathcal{M}, \mathcal{M}$), there are two unconnected subgraphs in the current initial configuration and three unconnected subgraphs in the current goal configuration which are encircled by "- -".



(a) (b)

Figure 8.8: MCS($\mathcal{M}, \mathcal{M}$) is encircled by "—" under mapping $\mathcal{M} \to \mathcal{M}$, $4 \to 6$, and $6 \to 10$. After removing MCS($\mathcal{M}, \mathcal{M}$), there are two unconnected subgraphs in both the current initial configuration and the current goal configuration which are encircled by "- -"



(a) (b)

Figure 8.9: MCS($\mathcal{M}, \mathcal{M}$) is encircled by "—" under mapping $\mathcal{M} \to \mathcal{M}$, $5 \to 7$, and $7 \to 11$.

Table 8.1: Reconfiguration actions for Task 1.

Action	ID	Face	ID	Face	Orientation
Undock	5	TOP Face	4	TOP Face	NULL
Dock	5	TOP Face	8	BOTTOM Face	NULL
Undock	7	TOP Face	6	TOP Face	NULL
Dock	7	TOP Face	6	BOTTOM Face	NULL
Undock	4	LEFT Face	2	LEFT Face	NULL
Dock	4	TOP Face	10	BOTTOM Face	NULL
Undock	6	RIGHT Face	2	RIGHT Face	NULL
Dock	6	TOP Face	2	BOTTOM Face	NULL

Table 8.2: The vertex height of modules before and after the reconfiguration process.

v	1	2	3	4	5	6	7	8	9	10	11
$h(v)$ in G_i	3	2	2	1	0	1	0	1	0	1	0
$h(v)$ in G_g	3	2	4	0	0	1	0	1	0	1	0

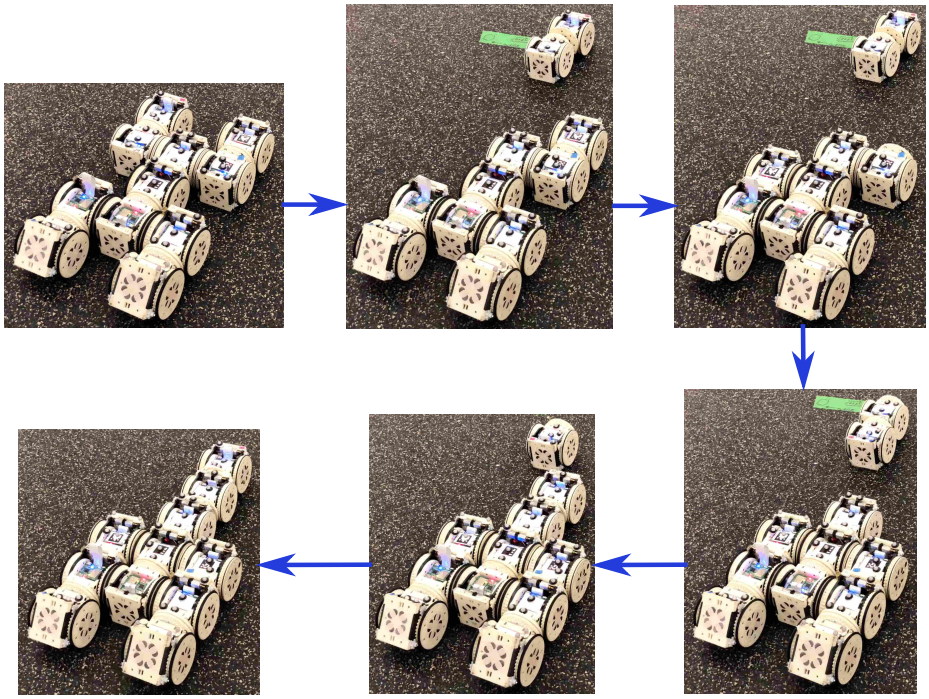


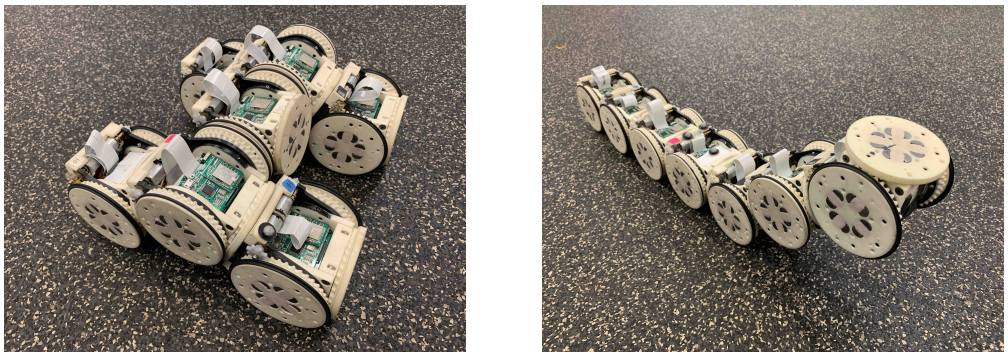
Figure 8.10: SMORES-EP hardware reconfiguration from a walker to a mobile vehicle with an arm.

moves to dock with Module 10. Finally, Module 6 moves to dock with Module 2 followed by Module 7. Now $\widehat{G}_g = (\widehat{V}_g, \widehat{E}_g)$ is formed and run the Matching and Mapping algorithm (Algorithm 3) which shows that this configuration is isomorphic to $G_g = (V_g, E_g)$ and no further reconfiguration actions are needed. Table 8.2 shows how the vertex height of each module changes after the reconfiguration actions and, for those modules that need to execute actions, their heights have no change except for Module 4.

These sequential reconfiguration actions can be executed in parallel by formulating all docking actions as a self-assembly problem. First execute all undocking actions in Table 8.1 so that these modules are free to move. Then run the parallel algorithm (Algorithm 4) in which the target topology is the goal configuration, and the mapping is simply $f^{-1} : V_g \to V_i$ derived from the reconfiguration planner. The algorithm starts from modules in the goal configuration with depth being 1, but ignores existing connections which are not constructed by docking actions in Table 8.1. By the mapping, the depth of Module 4 and Module 5 in the goal configuration are both 2, the depth of Module 6 is 3, and the depth of Module 7 is 4. Hence, the output of the assembly process is first $(5, \mathbf{T}, 8, \mathbf{B})$ and $(4, \mathbf{T}, 10, \mathbf{B})$ in parallel, then execute $(6, \mathbf{T}, 2, \mathbf{B})$, finally execute $(7, \mathbf{T}, 6, \mathbf{B})$.

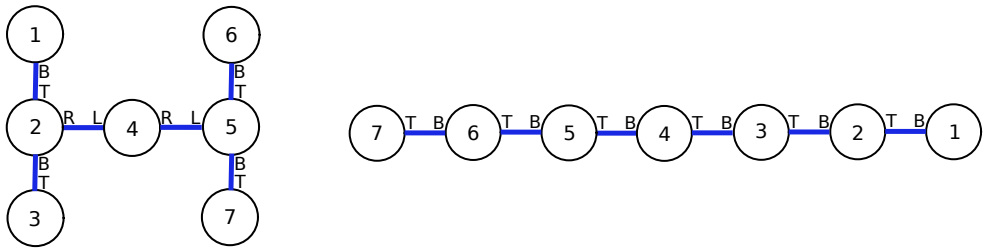
Task 2: Driver $\to$ Snake

This task is to reconfigure a cluster of SMORES-EP modules from a driver (Figure 8.11a) into a snake (Figure 8.11b) with seven modules. The initial and the goal graph representation are shown in Figure 8.12 where τ_i is Module 4 and τ_g is Module 4 respectively. MCS(τ_i, τ_g) is empty. A virtual module $\mathcal{M}$ and a virtual connection $(\mathcal{M}, \tau_i)$ are added to G_i and a similar operation is applied to G_g. Then MCS$(\mathcal{M}, \mathcal{M})$ is under mapping $\mathcal{M} \to \mathcal{M}$ and $4 \to 4$ which can be removed for further configuration decomposition. The final mapping between these two configurations $f : V_i \to V_g$ is $1 \to 7$, $2 \to 5$, $3 \to 6$, $4 \to 4$, $5 \to 3$, $6 \to 2$, and $7 \to 1$ and the corresponding reconfiguration actions are shown in Table 8.3.



(a) (b)

Figure 8.11: Reconfigure a driver configuration (a) into a snake configuration (b) with seven SMORES-EP modules involved.



(a) (b)

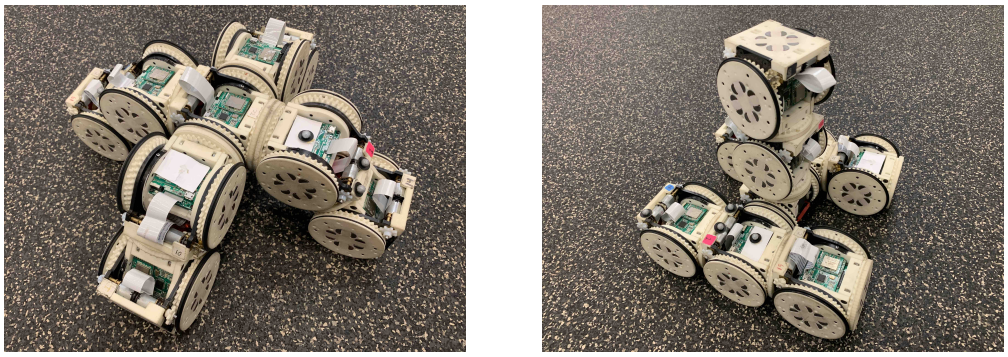
Figure 8.12: (a) The graph representation of the driver configuration. (b) The graph representation of the snake configuration.

Table 8.3: Reconfiguration actions for Task 2.

Action	ID	Face	ID	Face	Orientation
Undock	1	BOTTOM Face	2	TOP Face	NULL
Dock	1	TOP Face	3	BOTTOM Face	NULL
Undock	7	TOP Face	5	BOTTOM Face	NULL
Dock	7	BOTTOM Face	6	TOP Face	NULL
Undock	2	RIGHT Face	4	LEFT Face	NULL
Dock	2	TOP Face	4	BOTTOM Face	NULL
Undock	5	LEFT Face	4	RIGHT Face	NULL
Dock	5	BOTTOM Face	4	TOP Face	NULL

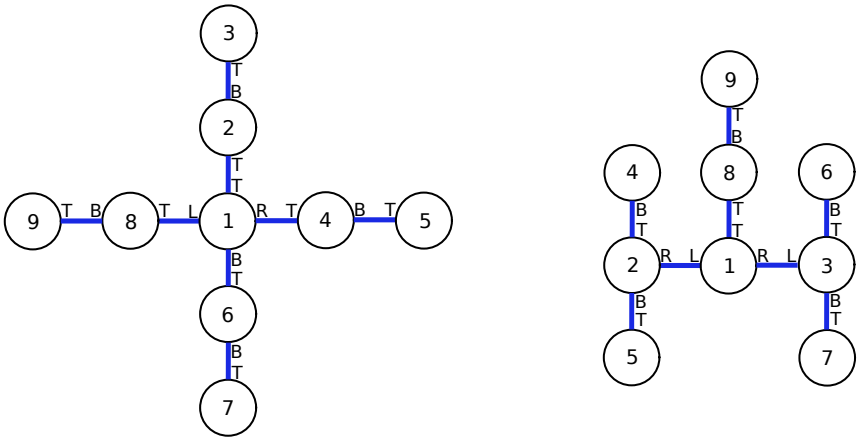
Task 3: Omni-Driver → Mobile Observer

This task is to reconfigure a cluster of SMORES-EP modules from an omni-driver (Figure 8.13a) into a mobile observer (Figure 8.13b) with nine modules. The initial and the goal



(a) (b)

Figure 8.13: Reconfigure a omni-driver configuration (a) into a mobile observer configuration (b) with nine SMORES-EP modules involved.



(a) (b)

Figure 8.14: (a) The graph representation of the omni-driver configuration. (b) The graph representation of the mobile observer configuration.

Table 8.4: Reconfiguration actions for Task 3.

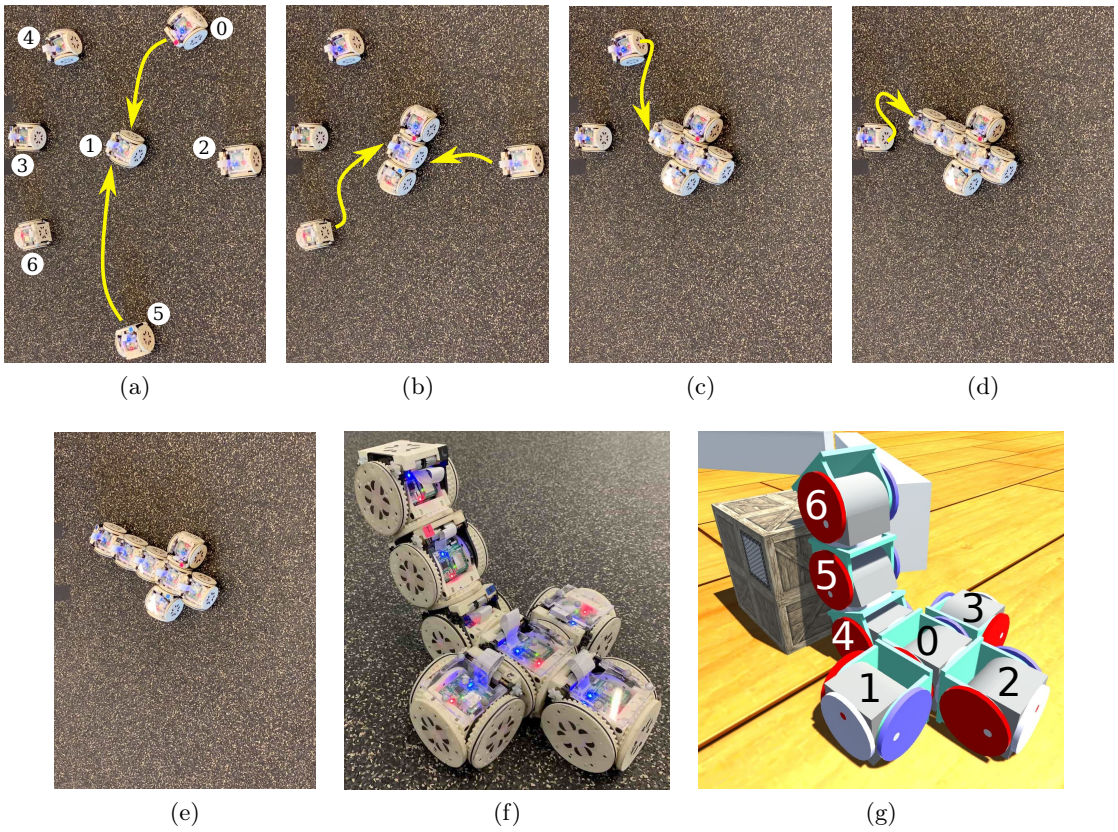
Action	ID	Face	ID	Face	Orientation
Undock	7	TOP Face	6	BOTTOM Face	NULL
Dock	7	BOTTOM Face	8	TOP Face	NULL
Undock	8	TOP Face	1	LEFT Face	NULL
Dock	8	LEFT Face	1	RIGHT Face	NULL
Undock	4	TOP Face	1	RIGHT Face	NULL
Dock	4	RIGHT Face	1	LEFT Face	NULL
Undock	6	TOP Face	1	BOTTOM Face	NULL
Dock	6	BOTTOM Face	4	TOP Face	NULL

graph representation are shown in Figure 8.14 where τ_i is Module 1 and τ_g is Module 1 respectively. MCS(τ_i, τ_g) is under mapping $1 \rightarrow 1$, $2 \rightarrow 8$, and $3 \rightarrow 9$ which can be maintained during the reconfiguration process. The final mapping for other modules is $4 \rightarrow 2$, $5 \rightarrow 5$, $6 \rightarrow 4$, $7 \rightarrow 6$, $8 \rightarrow 3$, and $9 \rightarrow 7$, and the corresponding reconfiguration actions are shown in Table 8.4.

8.5.2 Self-assembly

Task 1: Mobile Manipulator

The first task is to form a mobile vehicle with an arm that can reach higher locations as shown in Figure 8.15g. There are seven modules involved with Module 1 selected as the



(a) (b) (c) (d)

(e) (f) (g)

Figure 8.15: SMORES-EP hardware mobile manipulator self-assembly: (a) Execute actions $(0, \mathbf{B}, 1, \mathbf{L})$ and $(5, \mathbf{B}, 1, \mathbf{R})$; (b) Execute actions $(2, \mathbf{B}, 1, \mathbf{T})$ and $(6, \mathbf{T}, 1, \mathbf{B})$; (c) Execute action $(4, \mathbf{T}, 6, \mathbf{B})$; (d) Execute action $(3, \mathbf{T}, 4, \mathbf{B})$. (e) — (f) The final assembly. (g) The target kinematic topology.

Table 8.5: Initial locations of all modules in Task 1.

Module	x (m)	y (m)	θ (rad)
Module 0	0.017	0.357	1.142
Module 1	0	0	0
Module 2	0.305	0.129	0.641
Module 3	-0.318	-0.132	0.454
Module 4	-0.318	0.158	0.823
Module 5	0.264	-0.448	-0.763
Module 6	-0.172	-0.380	-2.431

root module m_τ which is the closest to the center (Figure 8.15a). The initial locations of all modules with respect to the root module are shown in Table 8.5. In the target topology (Figure 8.15g), the root module τ is computed as Module 0. Hence Module 0 in the target topology $G = (V, E)$ is mapped to Module 1 in the set of modules M on the ground. The mapping $f : V \to M$ that is $0 \to 1$, $1 \to 5$, $2 \to 2$, $3 \to 0$, $4 \to 6$, $5 \to 4$, and $6 \to 3$ is then derived by Kuhn-Munkres algorithm.

The assembly sequence starts from all vertices $v \in V$ with depth of one in the target topology which include Module 0, Module 2, Module 5, and Module 6. Because the root module is involved in this step, the assembly actions are separated into two subgroups.

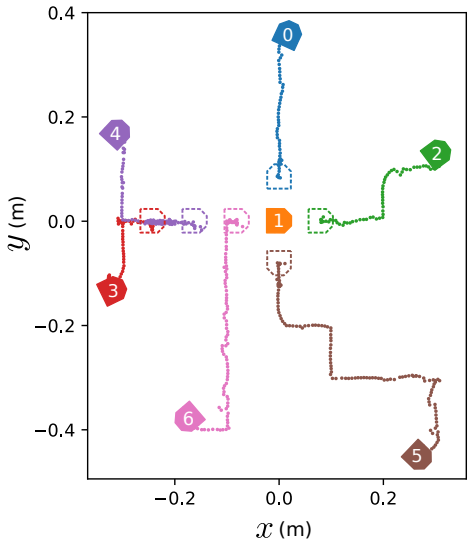
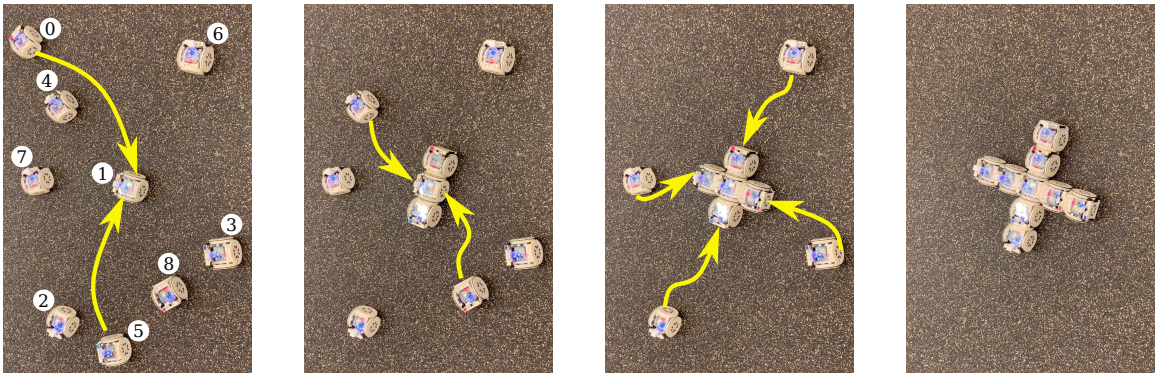


Figure 8.16: The actual path of each module for Task 1.

Module 0 and Module 5 start moving first to dock with LEFT Face and RIGHT Face of Module 1 respectively (Figure 8.15a), then Module 2 and Module 6 begin the docking process with TOP Face and BOTTOM Face of the root module (Figure 8.15b). In this way, even Module 1 can be moved slightly after docking with Module 0 and Module 5, Module 2 and Module 6 can still dock with Module 1 successfully. Lastly, Module 4 and Module 3 execute the assembly actions (Figure 8.15c and Figure 8.15d). It takes 130 seconds to finish the whole assembly process with paths shown in Figure 8.16.

Task 2: Holonomic Vehicle

The second task assembles nine modules into a holonomic vehicle in order to move as in Figure 8.17f. Based on the initial locations of all modules, the root module m_τ is then selected as Module 1. Then the pose of every module with respect to m_τ is computed and



(a) (b) (c) (d)

(e) (f)

Figure 8.17: SMORES-EP hardware holonomic vehicle self-assembly: (a) Execute assembly actions $(0, \mathbf{T}, 1, \mathbf{L})$ and $(5, \mathbf{T}, 1, \mathbf{R})$; (b) Execute assembly actions $(4, \mathbf{T}, 1, \mathbf{B})$ and $(8, \mathbf{T}, 1, \mathbf{T})$; (c) Execute assembly actions $(3, \mathbf{T}, 8, \mathbf{B})$, $(2, \mathbf{T}, 5, \mathbf{B})$, $(7, \mathbf{T}, 4, \mathbf{B})$, and $(6, \mathbf{T}, 0, \mathbf{B})$. (d) — (e) The final assembly. (f) The target kinematic topology.

Table 8.6: Initial locations of all modules in Task 2.

Module	x(m)	y(m)	θ(rad)
Module 0	-0.421	0.388	-0.760
Module 1	0.	0.	0.
Module 2	-0.110	-0.469	1.445
Module 3	0.386	-0.143	0.509
Module 4	-0.343	0.082	-2.961
Module 5	0.118	-0.472	1.700
Module 6	0.215	0.428	-2.058
Module 7	-0.342	-0.044	-1.249
Module 8	0.270	-0.317	2.416

shown in Table 8.6. The root module of the target topology $G = (V, E)$ in Figure 8.17f is Module 0. Given Module 0 in the target topology is mapped to Module 1 in this set of modules M on the ground, the optimal mapping $f : V \to M$ is derived as $0 \to 1$, $1 \to 0$, $2 \to 8$, $3 \to 5$, $4 \to 4$, $5 \to 6$, $6 \to 3$, $7 \to 2$, and $8 \to 7$. The assembly process is shown in Figure 8.17a — Figure 8.17c and the final assembly is shown in Figure 8.17d and Figure 8.17e. The behavior that the group of assembly actions is separated into two when the root module is involved can be seen to ensure the success of the docking process. Then the rest of the assembly actions can be executed in parallel. Module 2, 3, 6, and 7 begin moving at the same time to locations close to their destinations, adjust their poses, and finally approach to

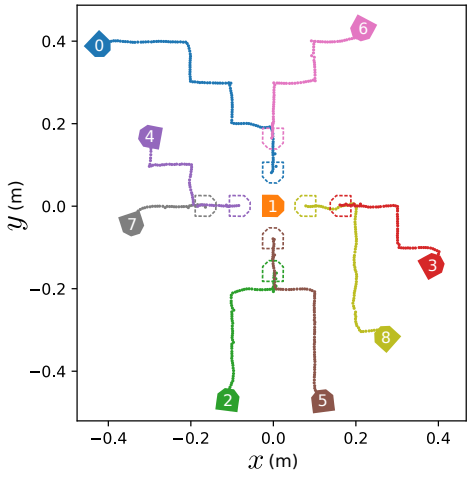
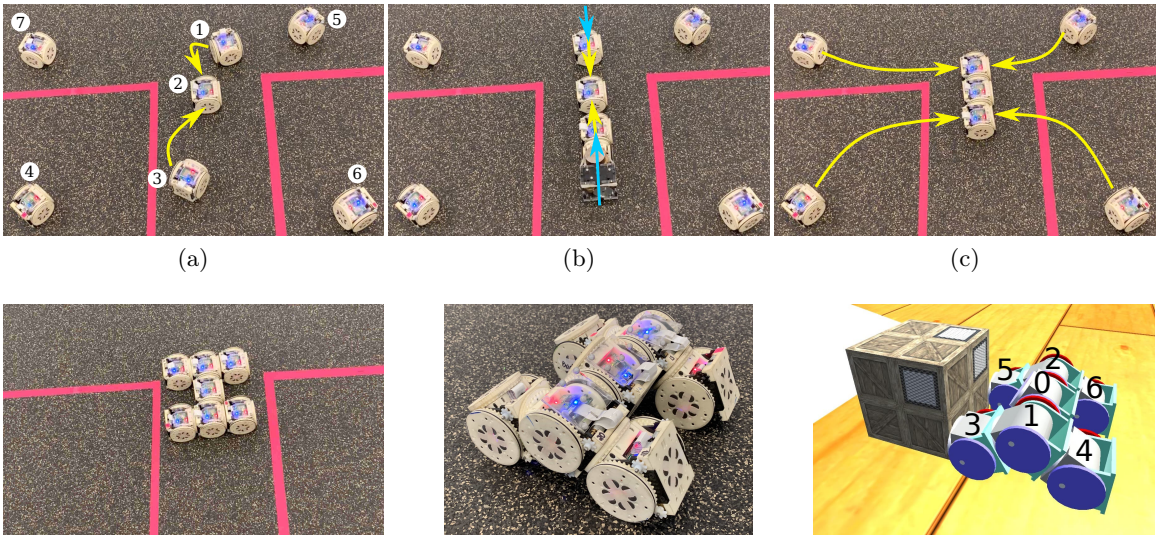


Figure 8.18: The actual path of each module for Task 2.

execute docking actions. It takes 103 seconds in total to finish the whole assembly process. With the proposed planner and controller, the recorded actual path of every module in the experiment is illustrated in Figure 8.18.

Task 3: RC Car

The last task assembles seven modules into a vehicle in order to push heavy items shown in Figure 8.19f. The root module m_τ is selected as Module 2 that is the closest module to the center of the cluster. Then the initial locations of all modules with respect to m_τ are shown in Table 8.7. The root module τ of the target topology $G = (V, E)$ is Module 0, and given τ is mapped to m_τ, the mapping $f : V \to M$ that is $0 \to 2$, $1 \to 1$, $2 \to 3$, $3 \to 5$, $4 \to 7$, $5 \to 6$, and $6 \to 4$ is derived. The assembly actions are shown in Figure 8.19a — Figure 8.19c. For the first step, Module 1 needs to dock its RIGHT Face with Module 2 LEFT Face and Module 3 needs to dock its LEFT Face with Module 2 RIGHT Face. These two assembly actions cannot be executed directly since the LEFT Face and RIGHT Face cannot be aligned directly. This step needs the help of a helping module shown in Figure 8.19b. In



(a) (b) (c)

(d) (e) (f)

Figure 8.19: SMORES-EP hardware RC car self-assembly: (a) Execute actions $(1, \mathbf{R}, 2, \mathbf{L})$ and $(3, \mathbf{L}, 2, \mathbf{R})$; (b) A helping module is used to execute the current docking action; (c) Execute actions $(4, \mathbf{T}, 3, \mathbf{B})$, $(7, \mathbf{T}, 1, \mathbf{B})$, $(6, \mathbf{B}, 3, \mathbf{T})$, and $(5, \mathbf{B}, 1, \mathbf{T})$. (d) — (e) The final assembly. (f) The target kinematic topology.

Table 8.7: Initial locations of all modules in Task 3.

Module	x(m)	y(m)	θ(rad)
Module 1	0.070	0.155	-1.352
Module 2	0	0	0
Module 3	-0.066	-0.250	0.997
Module 4	-0.299	0.170	0.811
Module 5	0.311	0.197	-2.539
Module 6	0.330	-0.373	-2.728
Module 7	-0.487	0.218	-0.436

this experiment, there is only one helping module. Similar to common docking actions, the helping module first navigates to a location close to Module 3, then adjusts its pose to align its TOP Face, and approaches Module 3 RIGHT Face for docking. Then it lifts Module 3 so that Module 3 can adjust its LEFT Face. Finally, the helping module delivers Module 3 to the desired location for docking with Module 2 RIGHT Face. This process repeats for Module 1 RIGHT Face. After both Module 1 and Module 3 are docked with the root module (Module 2), the rest of the four modules can execute assembly actions in parallel. It takes 260 seconds in total to finish this assembly task, though 210 seconds are consumed by the helping module. More helping modules working in parallel would decrease the duration. The final assembly result is shown in Figure 8.19d and Figure 8.19e. The recorded path in the experiment is shown in Figure 8.20.

8.6 Conclusion

This chapter addresses the self-adaption planners for modular robots using mobile-style reconfiguration, including self-reconfiguration and self-assembly. These planners are based on the graph representation of modular robots. For self-reconfiguration, an efficient algorithm is developed to do configuration decomposition iteratively by adding virtual modules and virtual connections so that each module in the initial configuration is mapped to a module in the goal configuration with which reconfiguration actions can be computed. For self-assembly, given a target kinematic topology, modules are mapped by those in the target

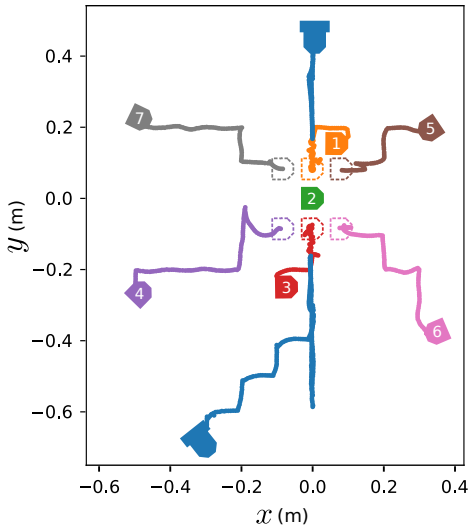


Figure 8.20: The actual path of each module in Task 3. The blue blocks without number labeled represent the helping modules.

configuration in an optimal way and then the assembly actions can be computed and executed in parallel. This parallel assembly planner can also speed up the self-reconfiguration process by formulating all necessary docking actions as a self-assembly problem.

Chapter 9

Manipulation Planning

This chapter presents the manipulation planning for modular robots in chain-type structures, such as SMORES-EP and CKBot. Once a suitable morphology is chosen for a scenario, a generic approach for control and motion planning is needed in order to handle various morphologies and environments. This chapter excerpts heavily from [74] and [80].

Motion planning in high-dimensional space is a challenging task. In order to perform dexterous manipulation in an unstructured environment, a robot with many DOFs is usually necessary, which also complicates its motion planning problem. Real-time control brings about more difficulties in which robots have to maintain stability while moving toward the target. Redundant systems are common in modular robots that consist of multiple modules and are able to transform into different configurations with respect to different needs. Different from robots with fixed geometry or configurations, the kinematics model of a modular robotic system can alter as the robot reconfigures itself, and developing a generic control and motion planning approach for such systems is difficult, especially when multiple motion goals are coupled. A new manipulation planning framework is developed in this chapter. The problem is formulated as a sequential linearly constrained quadratic program (QP) that can be solved efficiently. Some constraints can be incorporated into this QP, including a novel way to approximate environmental obstacles. This solution can be used directly for real-time applications or as an off-line planning tool, and it is validated

and demonstrated on the CKBot and SMORES-EP modular robot platforms.

9.1 Introduction

Manipulation tasks are common in robotics applications. In unstructured, cluttered environments, these tasks are usually executed by redundant robots to reach larger workspaces while avoiding obstacles and other constraints. This results in motion planning in high-dimensional space.

The motion planning problem is usually solved by some well developed framework (e.g. MoveIt! [23]) containing three components: a *path planner*, a *trajectory generator*, and a *tracking controller*. The path planner is responsible for generating collision-free paths. The trajectory generator smooths the computed paths and generates trajectories that can be parameterized by time while satisfying motion constraints, such as maximum velocities and accelerations. The tracking controller guarantees the motion of the robot when executing the generated trajectories. This type of framework has shown successful applications in many scenarios but rarely achieves real-time performance for all three components in high dimensions. Some approaches combine path planning with trajectory optimization that can directly construct trajectories resulting from optimization over a variety of criteria. These approaches are related to optimal control of robotic systems.

Modular robots are designed to be versatile and adaptable with respect to different tasks, environments, functions, or activities. A single module in a modular robotic system usually has one or more DOFs. Combining many modules to form versatile systems results in robots requiring representations with high dimensions. This dimensionality makes control and motion planning difficult. That the system is not a single structure but can take a very large number of configurations (typically exponential in the number of modules) requires an approach that can be applied to arbitrary configurations. For example, a modular robot configuration built with PolyBot modules [159] is shown in Figure 9.1 which has multiple serial kinematic chains. This is different from common multi-limb systems with a single base. These systems can be modeled such that chains are decoupled.

In a modular robotic system, modules usually are approximated by simple shapes such

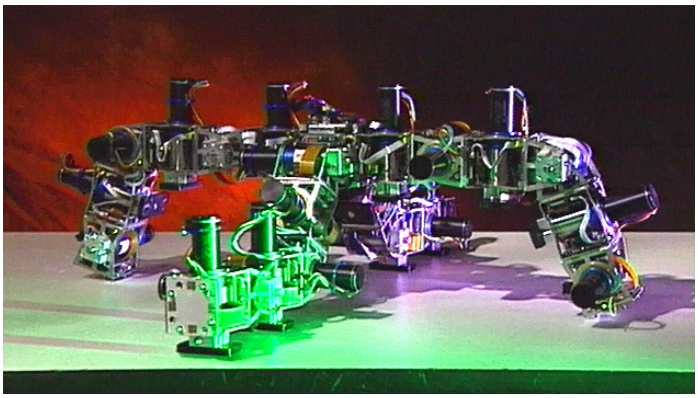


Figure 9.1: A modular robot configuration built by PolyBot modules is composed of multiple chains [159].

as a cube or a sphere. This is often useful for reconfiguration but can make manipulation more complex. Rather than two long fingers in a parallel jaw gripper, to obtain similar geometry, a modular robot may require many modules to form those long fingers. This results in grasping type of motions in which two or more chains can behave as a multi-arm system to clamp an object [126]. Hence, in order to grasp some object, motion planning for multiple (potentially high DOF) chains are necessary. In addition, their motions are strongly coupled. This fact leads to a more complicated control and planning problem.

This chapter presents a new approach for real-time manipulation planning and control as well as a novel way to approximate environmental obstacles, and apply it to two modular robotic systems. In order to solve the problem in general, a universal kinematics model is required for arbitrary configurations. This approach concomitantly can be easily extended, such as a dual-arm system or a modular robot configuration that has multiple chains. A quadratic programming approach that can be solved efficiently for real-time applications is proposed. This requires that system control stability, hardware motion constraints (joint limits and actuation limits), and collision avoidance can be incorporated into this quadratic program (QP) as linear constraints. One advantage of this approach is that the large variety of configurations and kinematic structures found from modular robotic systems can be represented easily as linear constraints, including situations where multiple portions of the robot may have different simultaneous goals. This approach can also be used as an off-line

trajectory planner by simple Euler integration. The framework is tested and evaluated on CKBot and SMORES-EP in the end.

9.2 Related Work

High-dimensional motion planning and control have been studied over several decades. This section reviews several types of approaches from previous work and some special approaches for modular robots.

9.2.1 Motion Planning for Manipulation

Artificial potential field manipulation planning methods can avoid searching in high-dimensional configuration space, planning in operational space directly [61]. Robots can avoid collision in real time, but may get stuck at local minima. Analytical navigation functions that have a unique minimum at the goal configuration avoiding local minima are shown in [110]. However, it is usually computationally expensive to build such a navigation function in general. A Monte Carlo technique was applied to escape local minima of the potential by executing Brownian motions [7].

Sampling-based approaches have been used widely for high-dimensional motion planning problems. The probabilistic roadmap (PRM) has been demonstrated on planar articulated robots with many DOFs [3, 59]. Expansive configuration space was proposed to resolve the narrow passage issue which is a common problem for sampling-based planners [50]. Rapidly-exploring Random Trees (RRT) approach was later presented in [67] to deal with nonholonomic constraints. An optimal sampling-based planner (RRT*) is introduced in [58] with less efficiency. These approaches require post-processing to generate smooth trajectories in order to be executable for real tasks.

Search-based planners rely on the discretization of the space. However, these approaches are generally not suitable for high-dimensional problems. For example, naïve A* can rarely scale to large complicated planning problems. In order to increase the efficiency of these approaches, a number of suboptimal heuristic searches have been proposed [33, 70, 71]. These methods are promising but are currently computationally inefficient when solving

motion planning problems in high-dimensional space.

Once a feasible path is found, a trajectory generator is needed to smooth and shorten the computed path with time parameterization. Trajectories are modeled as elastic bands that need to maintain equilibrium states under internal contraction forces and external repulsion forces [11, 108]. Obstacles in the workspace are considered directly which is also beneficial for real-time trajectory modification while a precomputed path is necessary.

Another class of planners is related to optimal control. Rather than separate the planning process into two phases (path planning and trajectory planning), trajectories are constructed directly by these frameworks which optimize over a variety of criteria. A global time-optimal trajectory generator is introduced in [130]. It combines a grid search with a local optimization to obtain the global optimal solution. This approach requires the representations of obstacle regions in configuration space which is difficult to derive for high-dimensional problems. CHOMP formulates the cost to be the combination of trajectory smoothness and obstacle avoidance, and gradients for these two terms are needed [109, 170]. This approach uses pre-computed signed distance fields for collision checking. A similar idea is used in ITOMP that can also deal with dynamic environments [104]. In contrast, STOMP [55] can also handle more general objective functions for which gradients are not available by using trajectory samples, but can be difficult to determine the number of samples. A sequential convex optimization approach is presented in [122] which adds new constraints and costs during the motion so as to tackle a larger range of motion. Collision is detected by checking the intersection of the swept-out volume of the robot in an interval and obstacles, and a collision-avoidance penalty gradient can be incorporated into the optimization problem to ensure safety. These works mainly focus on single-high-DOF-arm manipulation tasks. Given the trajectories of end-effectors, an optimal control framework is formulated to solve whole-body manipulation tasks [128]. A repulsive velocity can be applied to any rigid body whenever it collides with any obstacle based on a physical simulator. Reachable sets are used for safe and real-time trajectory design in [46], but the reachability analysis has to be offline. Some optimal controllers handle the obstacles by mixed integer programming which

is known to be an NP-hard problem [121].

The approach presented here is also related to optimal control and differs from these previous works in two ways: 1. the way in which the motion planning problem is formulated and 2. the simple model that approximates the environmental obstacles. Multiple motion goals are incorporated into the objective function in the form of feedback controllers to guarantee the trajectory tracking performance or efficient search for navigation, and the output can be applied to the system directly to achieve real-time performance. During the motion, both the objective function and constraints may be updated according to the current scenario which allows the approach to tackle a wider range of tasks. For collision avoidance, a new way to simplify obstacles dynamically during the motion is presented, and the collision avoidance constraint can be modeled as linear constraints in order to solve the optimization problem efficiently. A collision-avoidance penalty is added to the objective function when any rigid body is near any obstacle in the form of the projected motion from the rigid body to this obstacle. This approach is well suited for real-time applications since its output can be applied on robotic systems directly in real-time, or can be used as an off-line trajectory planner by integrating the output over time.

9.2.2 Modular Robot Control and Planning

Modular robots are inherently systems with many DOFs. They are usually composed of a large number of modules with each module has one or more DOFs. This chapter addresses the manipulation tasks of modular robots that form configurations in tree topologies. That is they are constructed from multiple serial chain configurations without forming loops. Work related to manipulation of modular robot systems includes inverse kinematics for highly redundant chains using PolyBot [2, 159], and constrained optimization techniques with non-linear constraints [31]. Due to complicated constraints in these approaches, real-time applications for large systems cannot be guaranteed and numerical issues have to be addressed when solving the optimization problem in the presence of obstacles. Another set of related work includes controller design for modular robots, such as an adaptive control approach using a neural network architecture [89], a virtual decomposition control approach [168], a

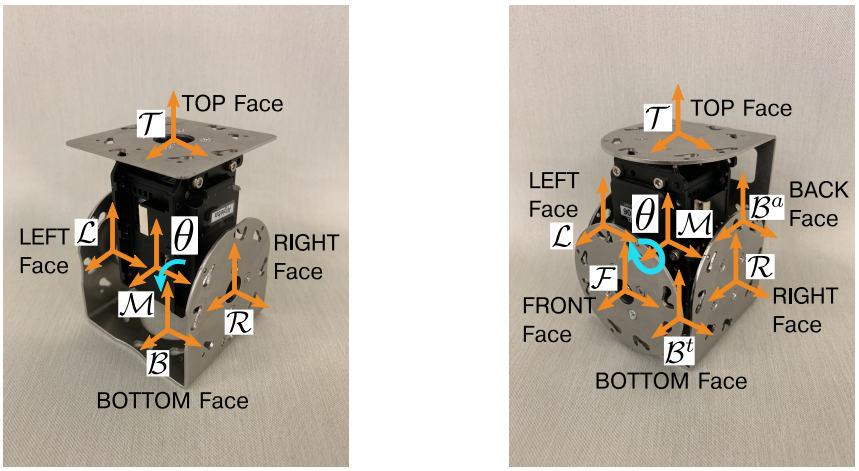
distributed control method with torque sensing [84], and a centralized controller [36]. These approaches consider the control problem in a free environment and require extra motion planning to fully control the system in a complex environment.

9.3 Kinematics For Modular Robots

In this section, a general kinematics model for modular robots is derived. For other manipulators, a similar technique can be applied to derive necessary models in order to utilize this planning framework.

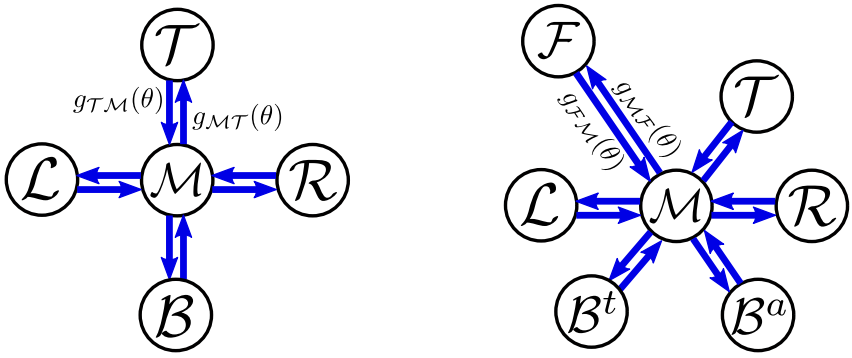
9.3.1 Kinematics Graph

The representation of a modular robot configuration is already discussed in Chapter 7 which is an undirected graph $G = (V, E)$. Each vertex $v \in V$ represents a module and each edge $e \in E$ represents the connection between two modules. In order to further describe the kinematics model of a modular robot configuration, a *module graph* is used to model a module's topology which includes all connectors and joints. A **module graph** is a directed graph $G_m = (V_m, E_m)$: each vertex is a rigid body in the module which is either a connector or the module body, and each edge represents how two adjacent rigid bodies are connected. The transformations among all rigid bodies are determined by the joint set and geometry. For example, a CKBot UBar module in Figure 9.2a is a single-DOF module as well as four connectors (TOP Face, BOTTOM Face, LEFT Face, and RIGHT Face). A frame is attached to each rigid body ($\mathcal{T}$ for TOP Face, $\mathcal{B}$ for BOTTOM Face, $\mathcal{L}$ for LEFT Face, $\mathcal{R}$ for RIGHT Face, and $\mathcal{M}$ for the module body). For simplicity, $\mathcal{M}$ is attached to the center of the joint rotation axis and the frame for every connector is attached to the center of it, and when the module joints are in their zero positions, all rigid bodies are in the same orientation and the translation offsets among them are determined by the module geometry. Let $\mathcal{B}$ be fixed in $\mathcal{M}$, then the homogeneous transformations among $\mathcal{M}$, $\mathcal{L}$, and $\mathcal{R}$ are invariant of the joint parameter θ because they are rigidly connected. Only the homogeneous transformation between $\mathcal{M}$ and $\mathcal{T}$ is related to θ. This relationship can be fully represented in a directed graph shown in Figure 9.3a. The edge direction denotes the direction of the corresponding



(a) (b)

Figure 9.2: (a) A CKBot UBar module has one DOF and four connectors. (b) A CKBot CR module has one DOF and six connectors.

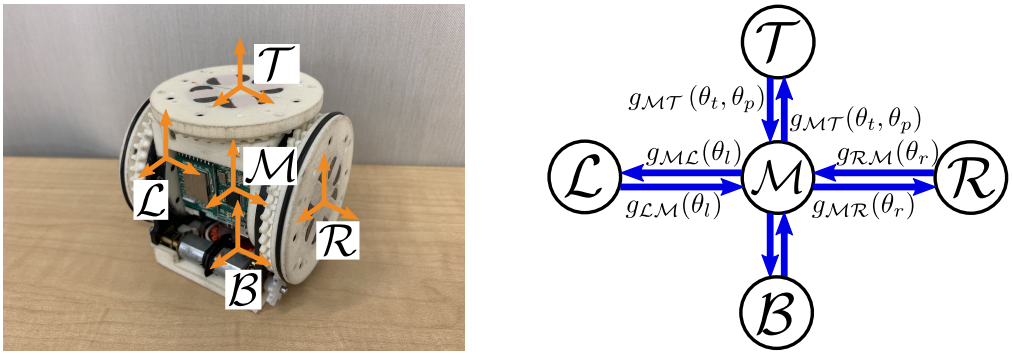


(a) (b)

Figure 9.3: (a) The module graph of a CKBot UBar module in which $g_{\mathcal{MB}}$, $g_{\mathcal{BM}}$, $g_{\mathcal{ML}}$, $g_{\mathcal{LM}}$, $g_{\mathcal{MR}}$, and $g_{\mathcal{RM}}$ are invariant of θ. (b) The module graph of a CKBot CR module in which $g_{\mathcal{MB}^a}$, $g_{\mathcal{B}^a\mathcal{M}}$, $g_{\mathcal{MB}^t}$, $g_{\mathcal{B}^t\mathcal{M}}$, $g_{\mathcal{MT}}$, $g_{\mathcal{TM}}$, $g_{\mathcal{ML}}$, $g_{\mathcal{LM}}$, $g_{\mathcal{MR}}$, and $g_{\mathcal{RM}}$ are invariant of θ.

forward kinematics. $\mathbf{G}_m$ is the set of unique module graphs G_m for a modular robotic system since some systems have more than one type of module (e.g., CKBot in Figure 9.2).

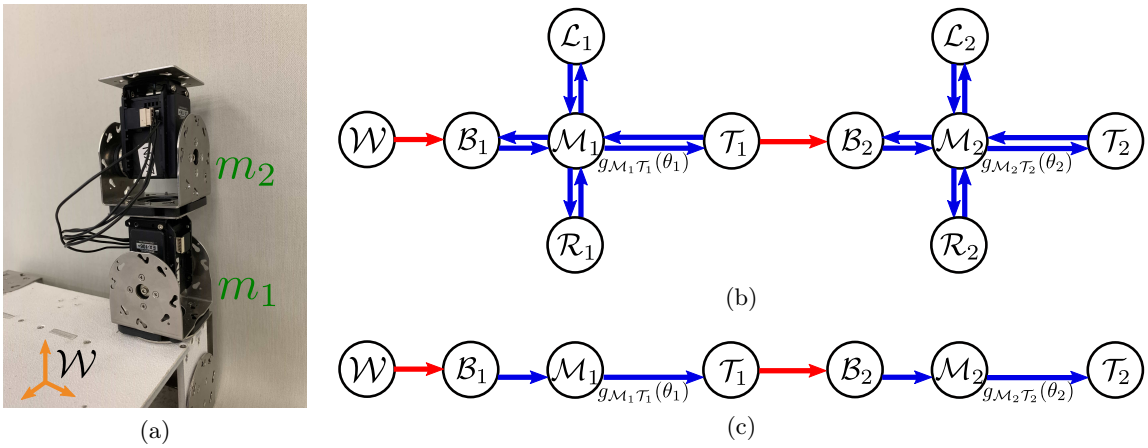
In general, given a module m with connector set C and joint set Θ, a frame $\mathcal{C}$ is attached to each connector $c \in C$ and frame $\mathcal{M}$ is attached to the module body. Let mapping $g_{\mathcal{F}_1\mathcal{F}_2} : Q \to SE(3)$ describe the forward kinematics from $\mathcal{F}_1$ to $\mathcal{F}_2$ in joint space Q, then $\forall c \in C$, $g_{\mathcal{MC}}$ and $g_{\mathcal{CM}}$ can be defined with respect to Θ. The results for CKBot CR modules and SMORES-EP modules are shown in Figure 9.3b and Figure 9.4. With a module graph



(a) (b)

Figure 9.4: (a) A SMORES-EP module has four DOFs and four connectors. The frames of all rigid bodies are shown and $\mathcal{B}$ is fixed in $\mathcal{M}$. (b) The module graph of a SMORES-EP module in which $g_{\mathcal{MB}}$ and $g_{\mathcal{BM}}$ are invariant of $\Theta = (\theta_l, \theta_r, \theta_p, \theta_t)$.

model, it is easy to obtain the **kinematics graph** $G_K = (V_K, E_K)$ for a modular robot configuration which is constructed by composing the modules by connecting connectors. A directed edge is used to denote each connection and the transformation between the two mating connectors is fixed since they are rigidly connected. Using this kinematics graph, a *kinematic chain* from frame $\mathcal{F}_1$ to frame $\mathcal{F}_2$ can be derived by following the shortest path $G_K : \mathcal{F}_1 \rightsquigarrow \mathcal{F}_2$. This creates a graph with no loops. A simple configuration built by two CKBot UBar modules is shown in Figure 9.5a. Frame $\mathcal{W}$ is the world frame and module m_1 is fixed to it via its BOTTOM Face. The kinematics graph for this configuration is shown



(a) (b) (c)

Figure 9.5: (a) A configuration by two CKBot UBar modules. (b) The kinematics graph model of the configuration. (c) The kinematic chain from $\mathcal{W}$ to $\mathcal{T}_2$.

in Figure 9.5b and the kinematic chain from $\mathcal{W}$ to $\mathcal{T}_2$ shown in Figure 9.5c. All the edges have fixed homogeneous transformations except for edge $(\mathcal{M}_1, \mathcal{T}_1)$ and edge $(\mathcal{M}_2, \mathcal{T}_2)$, and it can be concluded that the forward kinematics mapping is $g_{\mathcal{W}\mathcal{T}_2} : \mathbb{T}^2 \to SE(3)$ where $\mathbb{T}^p$ represents the p-torus. However, it can be seen that all the edges in the shortest path from $\mathcal{W}$ to $\mathcal{L}_1$ have fixed homogeneous transformations, so $\mathcal{L}_1$ is fixed in $\mathcal{W}$.

Similar to the configuration discovery algorithm (Algorithm 1), the kinematics graph can be built by visiting modules in breadth-first-search order. The given configuration is traversed from the module fixed to the world frame $\mathcal{W}$. When visiting a new module m, denoting its parent via its connector c as $\tilde{m}$ and the mating connector of $\tilde{m}$ as $\tilde{c}$, record the fixed homogeneous transformation $g_{\mathcal{C}\tilde{\mathcal{C}}}$ in which frame $\mathcal{C}$ and frame $\widetilde{\mathcal{C}}$ are attached to c and $\tilde{c}$ respectively. Not until all modules are visited, is the $G_K = (V_K, E_K)$ of the given configuration constructed. With this structure, there is no need for the case-by-case derivation of the kinematics as long as the kinematics for each type of module and connection are defined.

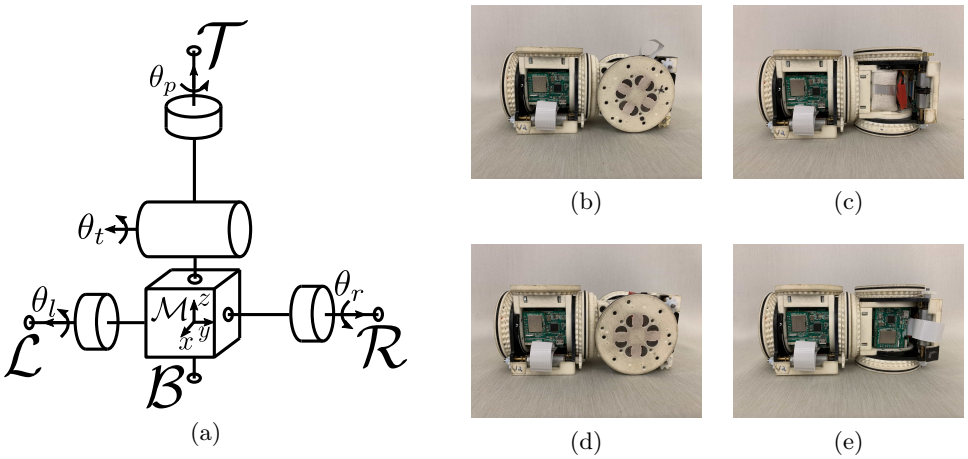
9.3.2 Kinematics for Modules

Recall that given a module m with connector set C and joint set Θ, a frame $\mathcal{C}$ is attached to each connector $c \in C$ and frame $\mathcal{M}$ is attached to the module body. For a joint $\theta \in \Theta$, a twist $\hat{\xi}_\theta \in se(3)$ can be defined with respect to $\mathcal{M}$ in which $\xi_\theta = (v_\theta, \omega_\theta) \in \mathbb{R}^6$ is the twist coordinates for $\hat{\xi}_\theta$[1], and $\boldsymbol{\xi}$ is the set of the twist associated with each joint. For homogeneous transformation $g_{\mathcal{M}\mathcal{C}}$, it is straightforward to have

$$g_{\mathcal{M}\mathcal{C}} = g_{\mathcal{M}\mathcal{C}}(\Theta^{\mathcal{C}}) = \prod_i \exp(\hat{\xi}_{\Theta_i^{\mathcal{C}}} \Theta_i^{\mathcal{C}})\ g_{\mathcal{M}\mathcal{C}}(0) \tag{9.1}$$

in which $\Theta^{\mathcal{C}}$ denotes the parameter vector in the joint space of the kinematic chain from $\mathcal{M}$ to $\mathcal{C}$. If no joints are involved in the kinematic chain from $\mathcal{M}$ to $\mathcal{C}$, then $\mathcal{C}$ is fixed in $\mathcal{M}$ and $g_{\mathcal{M}\mathcal{C}}$ is a constant determined by the geometry of the module. $g_{\mathcal{C}\mathcal{M}}$ is just the inverse of $g_{\mathcal{M}\mathcal{C}}$.

[1]Refer to Chapter 2 in [97] for background.



(a) (b) (c) (d) (e)

Figure 9.6: (a) Kinematics for SMORES-EP modules. (b) — (e) Four cases to connect $\mathcal{R}$ and $\mathcal{T}$.

9.3.3 Kinematics for Chains

A kinematic chain from frame $\mathcal{S}$ to $\mathcal{F}$ can be obtained as $G_K : \mathcal{S} \rightsquigarrow \mathcal{F}$ where $\mathcal{S}$ and $\mathcal{F}$ are two vertices of G_K. In this kinematic chain, all homogeneous transformations between connectors (e.g., $g_{\mathcal{T}_1\mathcal{B}_2}$ in Figure 9.5c) are fixed and can be easily computed. The relative orientation between connectors is determined by examining the connector design. For example, there are four cases for connecting SMORES-EP modules shown in Figure 9.6b — Figure 9.6e due to the arrangement of the magnets on the connector. The homogeneous transformation $g_{\mathcal{SF}}$ can be computed by multiplying the homogeneous transformation of each edge of path $G_K : \mathcal{S} \rightsquigarrow \mathcal{F}$ in order. In particular, let $\mathcal{S}$ be world frame $\mathcal{W}$, if module $m_1, m_2, \cdots, m_N$ are involved in this chain, then the position of the origin of $\mathcal{F}$ in $\mathcal{W}$ is given by

$$p_{\mathcal{F}}^{\mathcal{W}} = g_{\mathcal{WF}} \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^{\mathsf{T}} \tag{9.2}$$

and the instantaneous spatial velocity of $\mathcal{F}$ is given by the twist

$$\widehat{V}_{\mathcal{WF}}^{s} = \sum_{i=1}^{N} \sum_{j=1}^{N_i} \left(\frac{\partial g_{\mathcal{WF}}}{\partial \theta_{ij}} g_{\mathcal{WF}}^{-1} \right) \dot{\theta}_{ij} \tag{9.3}$$

in which θ_{ij} is the jth joint parameter of module m_i involved in this chain and the number of joints of module m_i involved in this chain is N_i. Rewrite Eq. (9.3) in twist coordinates as

$$V^s_{\mathcal{WF}} = J^s_{\mathcal{WF}} \dot{\Theta}^{\mathcal{WF}} \tag{9.4}$$

in which

$$\Theta^{\mathcal{WF}} = [\theta_{11} \cdots \theta_{1N_1} \ \theta_{21} \cdots \theta_{2N_1} \ \cdots \ \theta_{N1} \cdots \theta_{NN_n}]^\intercal \tag{9.5}$$

$$J^s_{\mathcal{WF}} = \begin{bmatrix} J_1 & J_2 & \cdots & J_N \end{bmatrix} \tag{9.6}$$

$$J_i = \begin{bmatrix} \left(\frac{\partial g_{\mathcal{WF}}}{\partial \theta_{i1}} g^{-1}_{\mathcal{WF}}\right)^\vee & \left(\frac{\partial g_{\mathcal{WF}}}{\partial \theta_{i2}} g^{-1}_{\mathcal{WF}}\right)^\vee & \cdots & \left(\frac{\partial g_{\mathcal{WF}}}{\partial \theta_{iN_i}} g^{-1}_{\mathcal{WF}}\right)^\vee \end{bmatrix} \tag{9.7}$$

and $J^s_{\mathcal{WF}}$ is the *spatial chain Jacobian*.

Define the twist of the jth joint of module m_i with respect to $\mathcal{W}$ as ξ'_{ij} that is

$$\xi'_{ij} = \left(\frac{\partial g_{\mathcal{WF}}}{\partial \theta_{ij}} g^{-1}_{\mathcal{WF}}\right)^\vee = \mathrm{Ad}_{g_{\mathcal{W}\mathcal{M}_i}} \xi_{ij}$$

in which $\mathrm{Ad}_{g_{\mathcal{W}\mathcal{M}_i}}$ is the adjoint transformation[2] and ξ_{ij} is defined in Section 9.3.2 for each joint in a module with respect to its module body frame. Then J_i becomes

$$J_i = \begin{bmatrix} \xi'_{i1} & \xi'_{i2} & \cdots & \xi'_{iN_i} \end{bmatrix} \tag{9.8}$$

With this spatial chain Jacobian, the velocity of the origin of frame $\mathcal{F}$ is

$$v^s_{\mathcal{F}} = \widehat{V}^s_{\mathcal{WF}} p^{\mathcal{W}}_{\mathcal{F}} = \left(J^s_{\mathcal{WF}} \dot{\Theta}^{\mathcal{WF}}\right)^\wedge p^{\mathcal{W}}_{\mathcal{F}} \tag{9.9}$$

For a module m_i in the kinematic chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}$ ($\mathcal{M}_i$ is a vertex in the corresponding path), a sub-kinematic chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{M}_i$ can be defined with joint parameter vector $\Theta^{\mathcal{W}\mathcal{M}_i} = \left[\theta_{11}, \theta_{12}, \cdots, \theta_{\bar{i}\bar{j}_i}\right]^\intercal$ where $\theta_{\bar{i}\bar{j}_i}$ is the parameter of the $\bar{j}_i$th joint of module $m_{\bar{i}}$. For example, take the sub-kinematic chain from $\mathcal{W}$ to $\mathcal{M}_2$ in Figure 9.5c, then $i = 2$, $\bar{i} = 1$,

[2]Refer to Chapter 2 in [97] for adjoint transformation definition.

$\bar{j}_i = 1$, since there is only one joint between $\mathcal{W}$ and $\mathcal{M}_2$ which is the 1st joint of module m_1. Then the *spatial module Jacobian* $J^s_{\mathcal{W}\mathcal{M}_i}$ or $J^s_{\mathcal{M}_i}$ for simplicity can be defined as

$$J^s_{\mathcal{M}_i} = \begin{bmatrix} \xi'_{11} & \xi'_{12} & \cdots & \xi'_{\bar{i}\bar{j}_i} \end{bmatrix} \tag{9.10}$$

and the velocity of the origin of $\mathcal{M}_i$ is

$$v^s_{\mathcal{M}_i} = \left(J^s_{\mathcal{M}_i} \dot{\Theta}^{\mathcal{W}\mathcal{M}_i} \right)^\wedge p^{\mathcal{W}}_{\mathcal{M}_i} \tag{9.11}$$

By replacing all twists associated with joints after the $\bar{j}_i$th joint of module $m_{\bar{i}}$ in the spatial chain Jacobian of chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}$ with 6×1 zero vectors, the spatial module Jacobian can also be written as

$$J^s_{\mathcal{M}_i} = \begin{bmatrix} \xi'_{11} & \xi'_{12} & \cdots & \xi'_{\bar{i}\bar{j}_i} & 0_{6\times1} & \cdots & 0_{6\times1} \end{bmatrix} \tag{9.12}$$

then the velocity of the origin of $\mathcal{M}_i$ is represented as

$$v^s_{\mathcal{M}_i} = \left(J^s_{\mathcal{M}_i} \dot{\Theta}^{\mathcal{W}\mathcal{F}} \right)^\wedge p^{\mathcal{W}}_{\mathcal{M}_i} \tag{9.13}$$

9.4 Control and Motion Planning

9.4.1 Control

Given the kinematic chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}$, the goal of the control task is to move $p^{\mathcal{W}}_{\mathcal{F}}$ (or $p_{\mathcal{F}}$ for simplicity) — the position of $\mathcal{F}$ — to follow a desired trajectory.

Let $\tilde{p}_{\mathcal{F}} = \tilde{p}_{\mathcal{F}}(t)$ be the desired trajectory for the robot to track and $\tilde{v}^s_{\mathcal{F}}$ (or $\tilde{v}_{\mathcal{F}}$ for simplicity) is the derivative of $\tilde{p}_{\mathcal{F}}$, and the error and its derivative are defined as

$$e = \tilde{p}_{\mathcal{F}} - p_{\mathcal{F}} \tag{9.14}$$

$$\dot{e} = \dot{\tilde{p}}_{\mathcal{F}} - \dot{p}_{\mathcal{F}} = \tilde{v}_{\mathcal{F}} - v_{\mathcal{F}} \tag{9.15}$$

The error e can converge exponentially to zero as long as it satisfies

$$\dot{e} + Ke = 0 \tag{9.16}$$

in which K is positive definite. Substitute e and $\dot{e}$:

$$\tilde{v}^s_{\mathcal{F}} - v^s_{\mathcal{F}} + K(\tilde{p}_{\mathcal{F}} - p_{\mathcal{F}}) = 0 \tag{9.17}$$

With Eq. (9.9), Eq. (9.17) can be rewritten as

$$(J^s_{\mathcal{WF}}\dot{\Theta}^{\mathcal{WF}})^{\wedge} p_{\mathcal{F}} = \tilde{v}^s_{\mathcal{F}} + K(\tilde{p}_{\mathcal{F}} - p_{\mathcal{F}}) \tag{9.18}$$

Eq. (9.18) is the control law to control the position of frame $\mathcal{F}$, namely $\dot{\Theta}^{\mathcal{WF}}$ (or $\dot{\Theta}^{\mathcal{F}}$ for simplicity) — the velocities of all involved joints that satisfy this equation — can move $p_{\mathcal{F}}$ to $\tilde{p}_{\mathcal{F}}$ in exponential time.

Suppose there are α motion goals $\tilde{p}_{\mathcal{F}_1}, \tilde{p}_{\mathcal{F}_2}, \cdots, \tilde{p}_{\mathcal{F}_\alpha}$, then the control law for all motion goals can be written as

$$\mathbf{JP} = \widetilde{\mathbf{V}} + \mathbf{K}(\widetilde{\mathbf{P}} - \mathbf{P}) \tag{9.19}$$

which is the stack of Eq. (9.18) for each motion goal. This makes the control problem for multiple motion goals easier without considering the fact that some motion goals may be coupled. That is, some kinematic chains may share DOFs. The work needs to do is only building an Eq. (9.18) for each individual motion goal and then stacking them as linear constraints. Building a specific model for different combinations of motion goals is not necessary.

Recall that a modular robotic system is usually redundant so that there can be an infinite number of solutions to Eq. (9.19). This problem is formulated as a quadratic program:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\dot{\Theta}^{\mathsf{T}}\dot{\Theta} \\ \text{subject to} \quad & \mathbf{JP} = \widetilde{\mathbf{V}} + \mathbf{K}(\widetilde{\mathbf{P}} - \mathbf{P}) \end{aligned} \tag{9.20}$$

where Θ is the set of joint parameters in kinematic chains $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_1$, $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_2$, $\cdots$, $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_\alpha$. Then solving (9.20) yields the minimum norm solution of joint velocities at every moment.

The joint position and velocity limits can be added to the quadratic program as inequality constraints as follows:

$$\frac{\Theta_{\min} - \Theta}{\Delta t} \leq \dot{\Theta} \leq \frac{\Theta_{\max} - \Theta}{\Delta t} \tag{9.21}$$

$$\dot{\Theta}_{\min} \leq \dot{\Theta} \leq \dot{\Theta}_{\max} \tag{9.22}$$

in which Δt is the time duration for the current step. Due to these two constraints, K cannot be too aggressive or solutions may not be obtained.

This optimization approach is helpful for many types of motion tasks. The controller can be used to move $p_{\mathcal{F}}$ to a desired position $\tilde{p}_{\mathcal{F}}$ by setting $\tilde{v}^s_{\mathcal{F}} = 0$, and it can also control $p_{\mathcal{F}}$ to move at a desired velocity by increasing $\tilde{p}_{\mathcal{F}}$ by $\tilde{v}_{\mathcal{F}} \Delta t$ for every time step.

9.4.2 Motion Planning

The goal of the motion planning task is to enable a cluster of modules to navigate collision-freely in an environment with obstacles.

Frame Boundaries

The cluster of modules can be kept in any polyhedral region in space which is defined by the boundaries of the environment. For a module m_i in the kinematic chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}$, let $\hat{s}_{ij}$ be the unit direction vector from $p^{\mathcal{W}}_{\mathcal{M}_i}$ (or $p_{\mathcal{M}_i}$ for simplicity) — the origin of $\mathcal{M}_i$ in world frame $\mathcal{W}$ — to the jth face of the environment polyhedron perpendicular with distance d_{ij}, then if enforcing the constraint

$$v^s_{\mathcal{M}_i} \bullet \hat{s}_{ij} = (J^s_{\mathcal{M}_i} \dot{\Theta})^{\wedge} p_{\mathcal{M}_i} \bullet \hat{s}_{ij} \leq d_{ij} \tag{9.23}$$

for every side of the environment polyhedron, $p_{\mathcal{M}_i}$ will never cross the boundary of the environment as long as this kinematic chain follows the velocity for much less than 1 second.

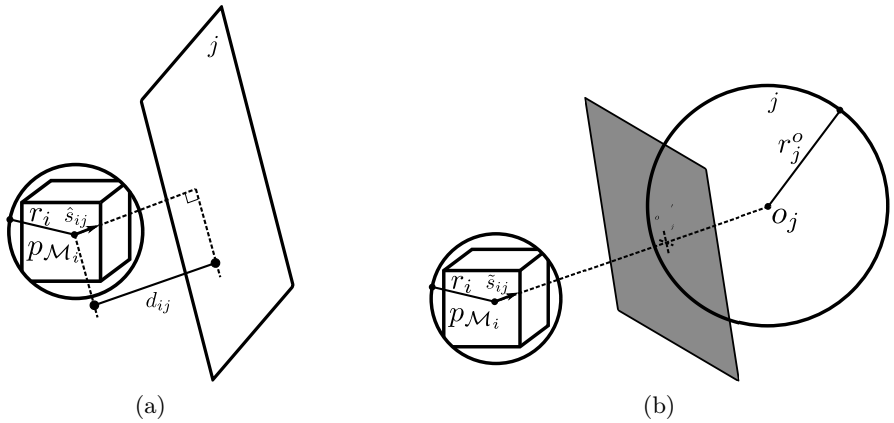


Figure 9.7: (a) Environment boundary. (b) Sphere obstacle avoidance.

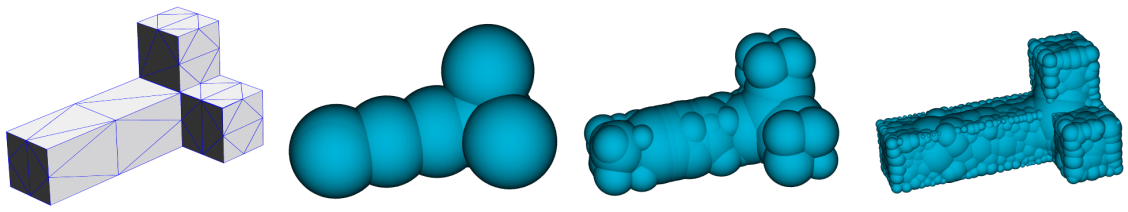
Using a sphere with radius r_i to approximate the geometry size of module m_i, then the constraint

$$v^s_{\mathcal{M}_i} \bullet \hat{s}_{ij} = (J^s_{\mathcal{M}_i}\dot{\Theta})^\wedge p_{\mathcal{M}_i} \bullet \hat{s}_{ij} \leq d_{ij} - r_i \tag{9.24}$$

will ensure that the module body will always be inside the environment boundaries (Figure 9.7a). Thus, applying constraint (9.24) to all modules in the kinematic chain will ensure the chain will stay inside the environment.

Obstacle Avoidance

It is hard to represent the collision-free space analytically in joint space due to the high DOFs of modular robotic systems. Here an alternative is proposed. The obstacles can be approximated by a set of spheres using a sphere-tree construction algorithm [10]. Similar ideas have been explored in [31, 167]. There are two issues using this idea. This collision-avoidance constraint is modeled as the condition that the distance between every sphere approximating the robot and every sphere approximating the obstacles is greater than the sum of their radius. This leads to quadratic constraints which are not suitable for real-time applications of large systems due to numerical issues. In addition, in order to approximate obstacles with decent accuracy, many spheres have to be generated. For example, a block object shown in Figure 9.8a is constructed by multiple spheres. A more accurate approximation of this object requires more spheres (Figure 9.8b — Figure 9.8d). An advantage of this



(a) (b) (c) (d)

Figure 9.8: A block obstacle (a) is approximated with 3 levels of spheres. (b) 8 spheres in level 1. (c) 64 spheres in level 2. (d) 470 spheres in level 3.

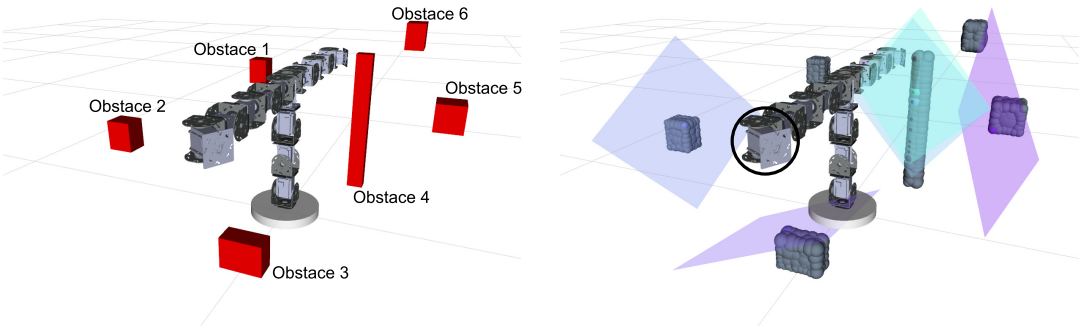
approach is that obstacles automatically have some level of buffer that can further guarantee motion safety. However, using a small number of spheres to approximate an obstacle can lead to too conservative planning space not finding collision-free paths when they exist. On the other hand, if there are a large number of spheres, there will also be a large number of constraints that can prohibit the optimization problem from being solved efficiently without numerical issues.

In this chapter, the obstacle avoidance requirement is modeled as linear constraints which are efficient to solve stably. For a module m_i in the kinematic chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}$, let $\tilde{s}_{ij}$ be the unit direction vector from $p_{\mathcal{M}_i}$ to the center of the jth obstacle sphere o_j in world frame $\mathcal{W}$ with radius r_j^o. Imaging a plane P_{ij} with $\tilde{s}_{ij}$ as its normal vector and o'_j being the point of tangency to this sphere, then if enforcing the constraint

$$v^s_{\mathcal{M}_i} \bullet \tilde{s}_{ij} = (J^s_{\mathcal{M}_i}\dot{\Theta})^\wedge p_{\mathcal{M}_i} \bullet \tilde{s}_{ij} \leq \|o'_j - p_{\mathcal{M}_i}\| - r_i \tag{9.25}$$

in which $o'_j = o_j - r_j^o s_{ij}$ for every obstacle sphere, $p_{\mathcal{M}_i}$ will never touch an obstacle (Figure 9.7b). In order to enable the system to safely navigate the environment, this constraint has to be applied for every module.

In order to resolve the difficulty that there can be a large number of obstacle spheres leading to a large number of constraints, a novel way is presented to significantly simplify these constraints as a pre-processing step for optimization. As mentioned, for module m_i and the jth obstacle sphere, compute an obstacle plane P_{ij} to build a linear constraint. If another obstacle sphere and module m_i are perfectly separated by P_{ij}, then this obstacle



(a) (b)

Figure 9.9: (a) A configuration built by 14 CKBot UBar modules is placed in a cluttered environment with 6 obstacles. (b) Apply the sphere-tree construction algorithm on all obstacles and the total number of obstacle spheres is 373. For the module inside the circle at its current state, only 5 highlighted obstacle spheres are necessary for collision avoidance and their obstacle planes are shown.

sphere can be ignored for m_i at the current planning step, because as long as module m_i never intersects with obstacle plane P_{ij}, m_i cannot touch this obstacle sphere. By this simple rule, the set of obstacle spheres can be refined by iteratively applying an erase-remove idiom technique efficiently for real-time performance. For example, a robot configuration built by fourteen CKBot UBar modules is placed in a cluttered environment where there are six obstacles (Figure 9.9a). After applying the sphere-tree construction algorithm, 373 obstacle spheres are derived in total shown in Figure 9.9b. For the module circled in the figure, after refining the set of all obstacle spheres, only five obstacle spheres need to be considered. Their obstacle planes are shown in Figure 9.9b. In the current step, Obstacle 1 and Obstacle 6 are behind other obstacles so they can be ignored. And for the rest of the obstacles, only five spheres are needed to approximate these obstacle-avoidance constraints.

9.4.3 Integrated Control and Motion Planning

With the control law in Section 9.4.1 and the motion constraints in Section 9.4.2, the control and motion planning problem for multiple kinematic chains $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_i, i = 1, 2, \cdots, \alpha$

can be formalized as the following quadratic program with linear constraints:

$$\begin{aligned}
\text{minimize} \quad & \tfrac{1}{2}\dot{\Theta}^{\intercal}\dot{\Theta} \\
\text{subject to} \quad & \mathbf{JP} = \widetilde{\mathbf{V}} + \mathbf{K}(\widetilde{\mathbf{P}} - \mathbf{P}) \\
& \tfrac{\Theta_{\min} - \Theta}{\Delta t} \leq \dot{\Theta} \leq \tfrac{\Theta_{\max} - \Theta}{\Delta t} \\
& \dot{\Theta}_{\min} \leq \dot{\Theta} \leq \dot{\Theta}_{\max} \\
& (J^s_{\mathcal{M}_i}\dot{\Theta})^{\wedge} p_{\mathcal{M}_i} \bullet \hat{s}_{ij} \leq d_{ij} - r_i \quad \forall (\mathcal{M}_i, f_j) \in V_K \times F \\
& (J^s_{\mathcal{M}_i}\dot{\Theta})^{\wedge} p_{\mathcal{M}_i} \bullet \tilde{s}_{ik} \leq \|o'_k - p_{\mathcal{M}_i}\| - r_i \quad \forall (\mathcal{M}_i, S_k) \in V_K \times \mathbf{S}_i
\end{aligned} \tag{9.26}$$

in which F is the set of all faces of the environment polyhedron and f_j is the jth face, $\mathbf{S}$ is the set of all spheres approximating the environmental obstacles, $\mathbf{S}_i \subseteq \mathbf{S}$ is the current set of obstacle spheres under consideration for module m_i, and S_k is the kth sphere in $\mathbf{S}_i$. By solving this quadratic program, the minimum norm solution that satisfies the hardware limits, control requirement, and motion constraints can be obtained for the current time step given the current state of every kinematic chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_i$ where $i = 1, 2, \cdots, \alpha$, the desired velocity, and the position of the origin of each frame $\mathcal{F}_i$.

This formulation can be used for motion tasks with simple constraints (e.g., when obstacles are far from robots and motion goals). The equality constraint enables the motion goal to be achieved very fast with suitable gains. However, this can also cause difficulties for optimization. For complicated scenarios, a sequential convex optimization formulation in which the objective function and constraints are updated when encountering obstacles is proposed. Initially the objective function is in the following form:

$$f(\dot{\Theta}) = \|\dot{\Theta}\|^2 + \lambda \|\mathbf{JP} - (\widetilde{\mathbf{V}} + \mathbf{K}(\widetilde{\mathbf{P}} - \mathbf{P})\|^2 \tag{9.27}$$

in which λ is a weight to address the significance of the feedback controller. In order to avoid entering space that is hard to maintain safety, aggressive motions toward obstacles are penalized by adding $\|v^s_{\mathcal{M}_i} \bullet s_{ij}\|^2$ to the objective function when the distance between module body frame $\mathcal{M}_i$ of the robot and the jth obstacle is less than $d_{\min}$, and the objective

function becomes

$$f(\dot{\Theta}) = \|\dot{\Theta}\|^2 + \lambda\|\mathbf{JP} - (\widetilde{\mathbf{V}} + \mathbf{K}(\widetilde{\mathbf{P}} - \mathbf{P})\|^2 + \mu_{ij}\|v^s_{\mathcal{M}_i} \bullet s_{ij}\|^2 \tag{9.28}$$

in which μ_{ij} is also a weight. The new penalizing term means minimizing the motion of this module m_i toward the jth obstacle sphere. It is necessary to check every module to update the objective function and this can be computed easily by sphere-to-sphere distance.

If module m_i makes contact with the jth obstacle sphere, this module will be forced to move away by defining a repulsive velocity as the normal to the obstacle plane. This can be done by adding a hard inequality constraint

$$v^s_{\mathcal{M}_i} \bullet s_{ij} = \left(J^s_{\mathcal{M}_i}\dot{\Theta}\right)^{\wedge} p_{\mathcal{M}_i} \bullet s_{ij} \leq \gamma_{ij} \tag{9.29}$$

in which γ_{ij} is bounded between $-\|v^s_{\mathcal{M}_i}\|$ and 0. This constraint can force module body frame $\mathcal{M}_i$ to move away from the jth obstacle sphere. Note that the previously added penalizing term for this module $\mu_{ij}\|v^s_{\mathcal{M}_i} \bullet s_{ij}\|^2$ is removed from the objective function. The larger γ_{ij} is, the faster the module $\mathcal{M}_i$ moves away from the jth obstacle.

9.4.4 Iterative Algorithm for Manipulation Planning

The set of module graph $\mathbf{G}_m$ described in Section 9.3.1 and the twist set $\boldsymbol{\xi}$ described in Section 9.3.2 associated with all the joints in different type of modules are computed and stored. For a modular robot configuration G, assuming the base module $\bar{m}$ and how it is attached to the world frame $\mathcal{W}$ as well as the motion goals $\tilde{p}_{\mathcal{F}_1}$, $\tilde{p}_{\mathcal{F}_2}$, $\cdots$, $\tilde{p}_{\mathcal{F}_\alpha}$ for frame $\mathcal{F}_1$, $\mathcal{F}_2$, $\cdots$, $\mathcal{F}_\alpha$ respectively are known, the set of all faces of the environment polyhedron is F and the set of all spheres approximating environmental obstacles is $\mathbf{S}$, the full algorithm framework is shown in Algorithm 5 with following functions:

- `BFS`$(G, \mathbf{G}_m, \bar{m})$: Traverse a modular robotic configuration G in breadth-first-search order starting from $\bar{m}$ to construct the kinematics graph $G_K = (V_K, E_K)$;
- `GetChain`$(G_K, \mathcal{F})$: Return the kinematic chain from $\mathcal{W}$ to $\mathcal{F}$ in G_K;

Algorithm 5: Control and Motion Planning

Input: $\boldsymbol{\xi}$, $\mathbf{G}_m$, $\bar{m}$, $\mathcal{F}_1$, $\mathcal{F}_2$, $\cdots$, $\mathcal{F}_\alpha$, $\{\tilde{p}_{\mathcal{F}_i}(t) | 0 \leq t \leq T, i = 1, 2, \cdots, \alpha\}$, F, $\mathbf{S}$
Output: result

$G_K \leftarrow$ `BFS`$(G, \mathbf{G}_m, \bar{m})$;
$G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_i \leftarrow$ `GetChain`$(G_K, \mathcal{F}_i)$, $i \in [1, \alpha]$;
Initialize Θ;
Initialize $\mathbf{K}$ and Δt;
$t \leftarrow 0$;
while $\sum_{i=1}^{\alpha} \|p_{\mathcal{F}_i} - \tilde{p}_{\mathcal{F}_i}(T)\| \geq \epsilon$ **do**
    Compute $\hat{s}_{ij}$ $\forall (\mathcal{M}_i, f_j) \in V_K \times F$;
    Compute $\mathbf{S}_i \subseteq \mathbf{S}$ $\forall \mathcal{M}_i \in V_K$ and $\tilde{s}_{ik}$ $\forall S_k \in \mathbf{S}_i$;
    $\dot{\Theta} \leftarrow$ `SolveQP`$(G_K, \mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_\alpha, \tilde{\mathbf{P}}(t), \tilde{\mathbf{V}}(t), \mathbf{P}, \mathbf{K}, \Delta t)$;
    if $\dot{\Theta} =$ NULL **then**
        return result $\leftarrow$ FALSE;
    end
    Publish $\dot{\Theta}$ to the system;
    $t \leftarrow t + \Delta t$;
end
return result $\leftarrow$ TRUE;

- `SolveQP`$(G_K, \mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_\alpha, \widetilde{\mathbf{P}}(t), \widetilde{\mathbf{V}}(t), \mathbf{P}, \mathbf{K}, \Delta t)$: Construct and try to solve the quadratic program described in Section 9.4.3. If failed to solve this program, then return NULL as an invalid solution.

After initializing all the parameters, compute the unit direction vector $\hat{s}_{ij}$ between every $\mathcal{M}_i \in V_K$ and every face of the environment $f_j \in F$, compute $\mathbf{S}_i \subseteq \mathbf{S}$ for every $\mathcal{M}_i \in V_K$ and the corresponding unit direction vector $\tilde{s}_{ik}$ between $\mathcal{M}_i$ and every obstacle sphere $S_k \in \mathbf{S}_i$. If there is no valid solution, the program should stop, or the program will continue until every $p_{\mathcal{F}_i}$ is close enough to the destination $\tilde{p}_{\mathcal{F}_i}(T)$. If the trajectory $\tilde{p}_{\mathcal{F}_i}(t)$ is not specified and only $\tilde{p}_{\mathcal{F}_i}(T)$ where $T \to \infty$ is given, then this algorithm can automatically find a trajectory for modules to navigate the environment. The output from the planner can be applied directly online (e.g., running on robot modules) to achieve real-time performance, or can be integrated over time (one-step Euler integration) to generate the trajectory for each module or joint.

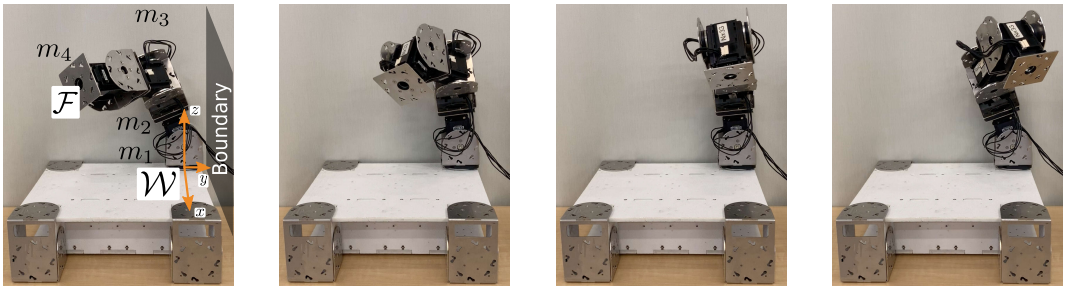
9.5 Experiments

Several experiments on two hardware platforms are performed to verify the approach. Here, it is shown that the framework is able to execute a motion task with guaranteed control performance on real hardware while satisfying all hardware constraints, frame boundary constraint, and obstacle avoidance. The framework is also tested in a complex scenario showing its ability for online trajectory optimization for navigation tasks.

9.5.1 Real-Time Control

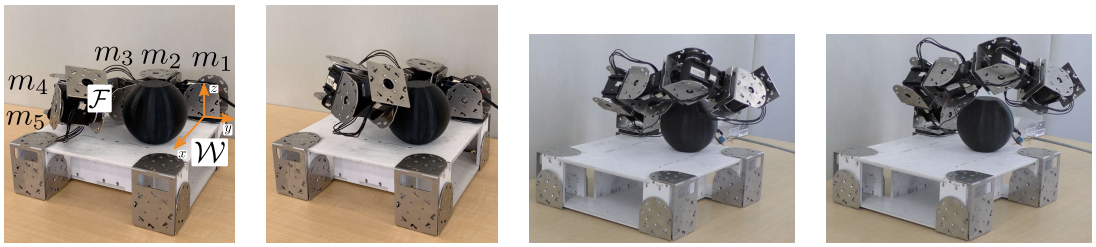
CKBot Chain

A configuration with four CKBot UBar modules is shown in Figure 9.10a. The base module $\bar{m} = m_1$ is attached to the world frame $\mathcal{W}$ and frame $\mathcal{F}$ is attached to connector $\mathcal{T}$ of module m_4. A virtual frame boundary is next to the right side of the base. The task is to control $p_{\mathcal{F}}$ to follow a given trajectory to the position shown in Figure 9.10d. Another experiment setup with five CKBot UBar modules is shown in Figure 9.11a. The black sphere is an obstacle, the base module $\bar{m} = m_1$ and frame $\mathcal{F}$ is attached to connector $\mathcal{T}$ of module m_5. Two tasks are executed: control $p_{\mathcal{F}}$ to follow a given trajectory and control $p_{\mathcal{F}}$ to approach a specified destination with the final position of $p_{\mathcal{F}}$ as shown in Figure 9.11d. The control loop runs at 20 Hz with gain $K = \text{diag}(1, 1, 1)$. Figure 9.12 and Figure 9.14a shows $p_{\mathcal{F}}(t)$



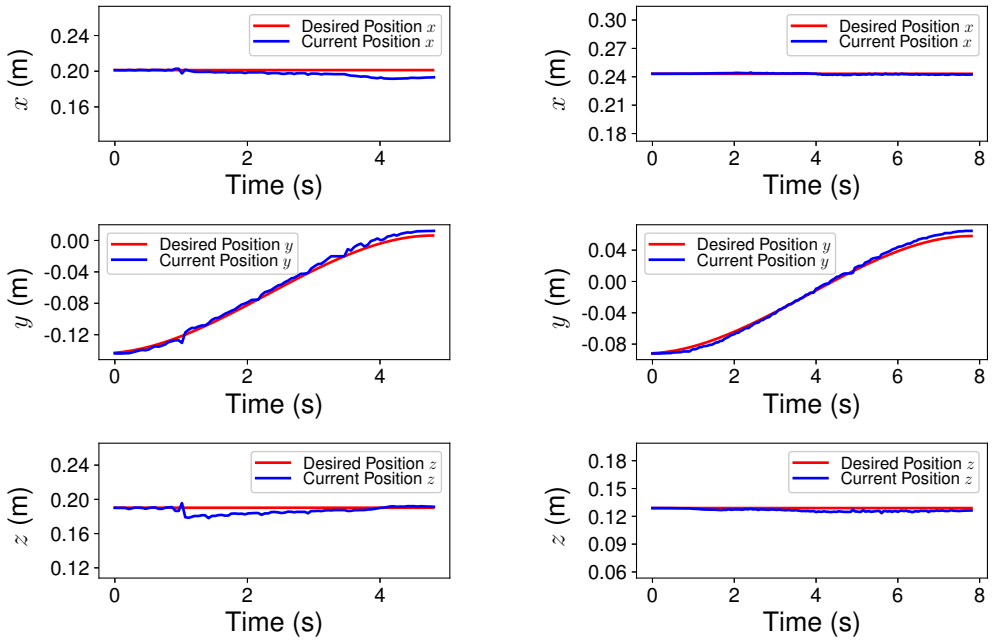
(a) (b) (c) (d)

Figure 9.10: Control $p_{\mathcal{F}}$ to follow a given trajectory along $+y$-axis of $\mathcal{W}$ by 15 cm from the initial pose (a) to the final pose (d). All the modules have to be on the left side of the boundary. m_1, m_2, and m_3 have to approach the boundary first (b) and then move away from the boundary (c) to finish the task.



(a) (b) (c) (d)

Figure 9.11: Control $p_{\mathcal{F}}$ from its initial pose (a) to its final pose (d) by both following a given trajectory along $+y$-axis of $\mathcal{W}$ by 15 cm and navigating to the destination directly. The modules have to move around the sphere obstacle while executing these two tasks.



(a) (b)

Figure 9.12: The motion of $p_{\mathcal{F}}$: (a) the four-module task; (b) the five-module trajectory following task.

and $\tilde{p}_{\mathcal{F}}$ of these three tests demonstrating the tracking and the navigation performance. The velocity commands for all modules in these two five-module demonstrations are shown in Figure 9.13 and all commands are within the constraints of each module. Modules move more aggressively at the beginning when executing the destination navigation task in order to quickly approach the destination.

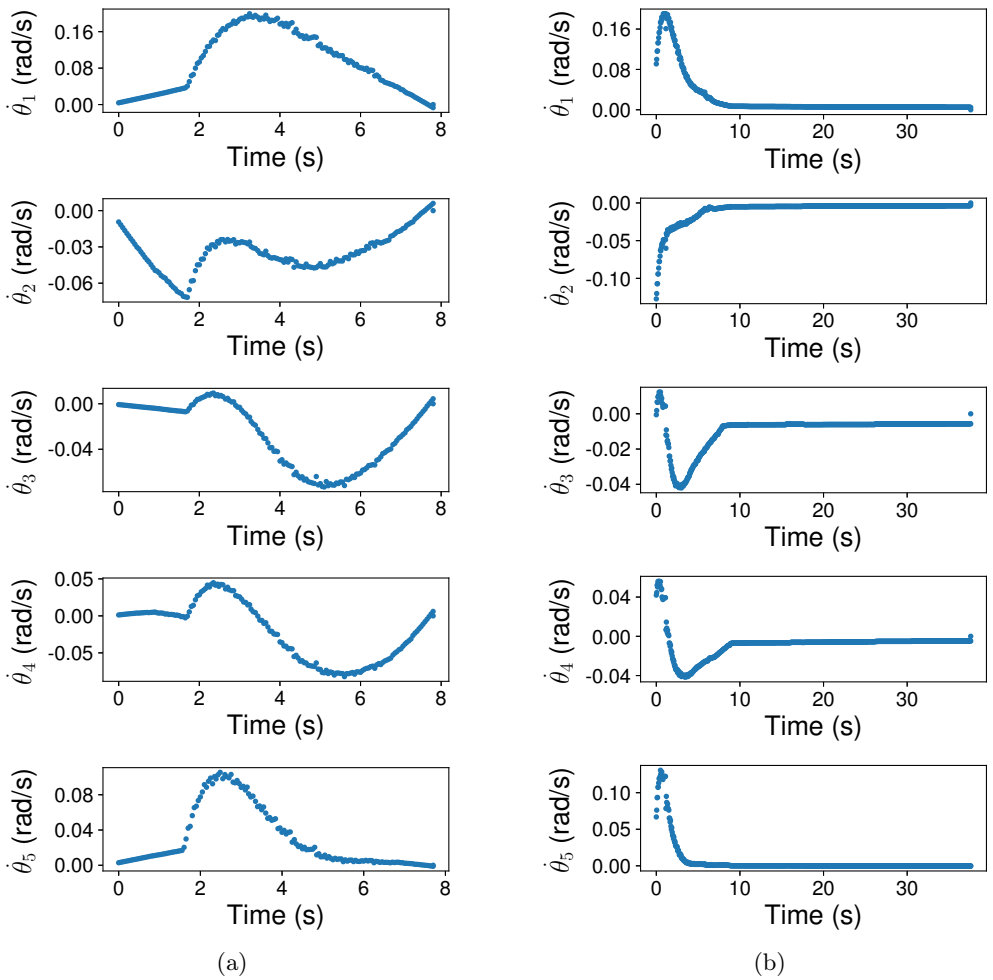
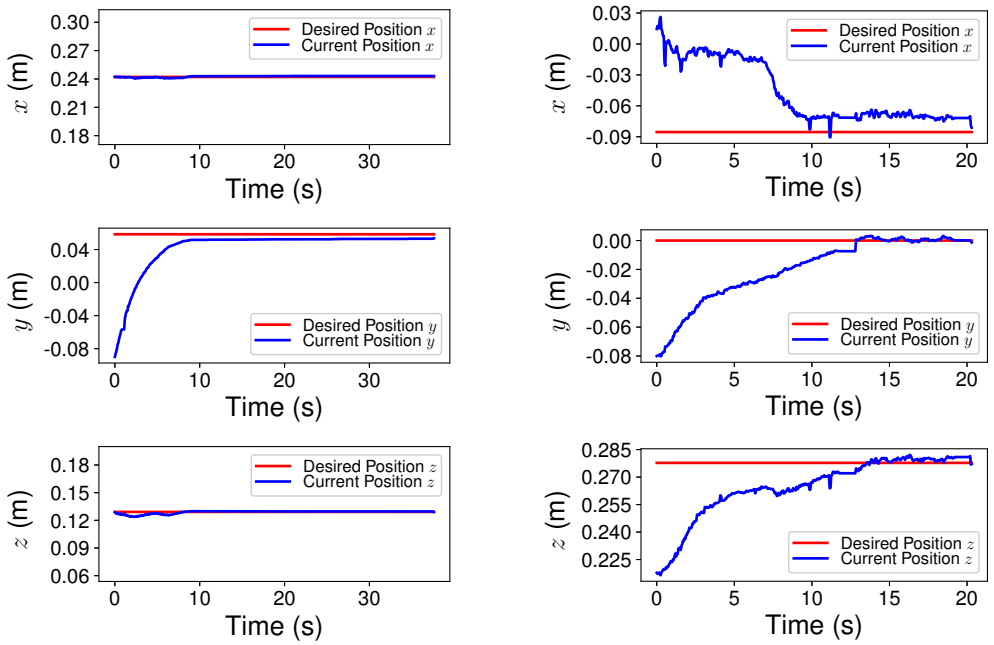


Figure 9.13: The control input $\dot{\Theta}$ for the five-module chain experiment: (a) the trajectory following task; (b) the destination navigation task.

SMORES-EP Chain

The experiment setup with four SMORES-EP modules is shown in Figure 9.15a. The base module $\bar{m} = m_1$ is fixed to the world frame $\mathcal{W}$ and frame $\mathcal{F}$ is attached to connector $\mathcal{T}$ of module m_4. This system has 16 DOFs and the task is to control $p_{\mathcal{F}}$ to navigate to a specified destination shown in Figure 9.15b. The control loop runs at 20 Hz and the gain $K = \text{diag}(0.5, 0.5, 0.5)$. The experiment result $p_{\mathcal{F}}(t)$ is shown in Figure 9.14b. The position sensors installed in SMORES-EP modules are PaintPots, and these low-cost sensors with a modified Kalman filter for nonlinear systems are used to provide position information of



(a) (b)

Figure 9.14: The motion of $p_{\mathcal{F}}$: (a) the CKBot five-module destination navigation task; (b) the SMORES-EP four-module chain destination navigation task.

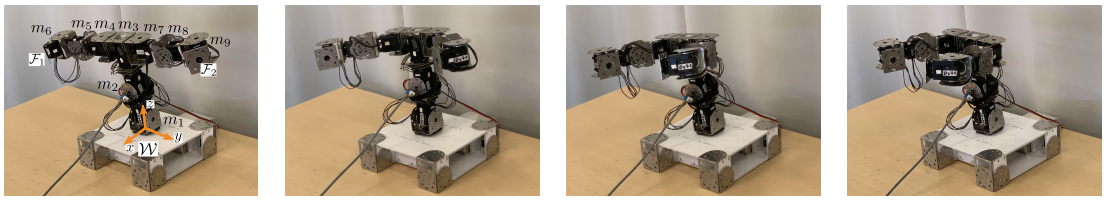
(a) (b)

Figure 9.15: Control a chain of SMORES-EP modules to navigate from its initial pose (a) to a goal pose (b). This chain is constructed by four modules with 16 DOFs.

each DOF (Chapter 5). Due to the limitations of the hardware, some noise is evident.

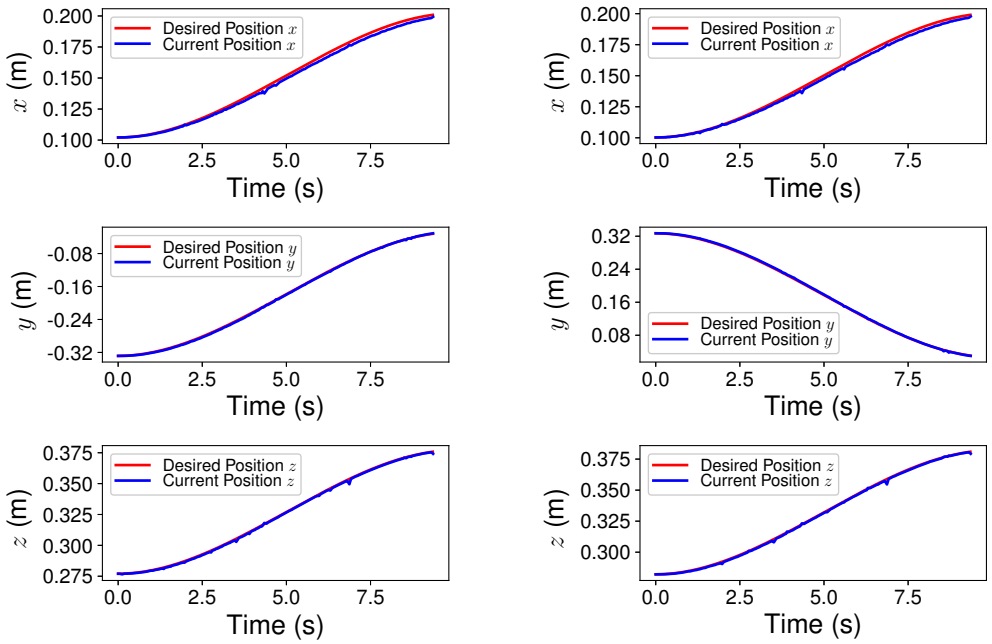
CKBot Branch

A configuration with nine CKBot UBar modules is shown in Figure 9.16a. The base module $\bar{m} = m_1$ is fixed to the world frame $\mathcal{W}$. Frame $\mathcal{F}_1$ is attached to connector $\mathcal{T}$ of module m_6 and frame $\mathcal{F}_2$ is attached to connector $\mathcal{T}$ of module m_9. Chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_1$ and



(a) (b) (c) (d)

Figure 9.16: Control $p_{\mathcal{F}_1}$ and $p_{\mathcal{F}_2}$ to follow two given trajectories respectively from their initial poses (a) to their final poses (d). Module m_1, m_2, and m_3 initially have to move backward (b) and then move forward (c) in order to control $p_{\mathcal{F}_1}$ and $p_{\mathcal{F}_2}$ to follow their trajectories.



(a) (b)

Figure 9.17: (a) The tracking result for $p_{\mathcal{F}_1}$. (b) The tracking result for $p_{\mathcal{F}_2}$.

$G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_2$ have common parts composed by module m_1, m_2, and m_3. The task is to control $p_{\mathcal{F}_1}$ and $p_{\mathcal{F}_2}$ to follow trajectories respectively to the pose shown in Figure 9.16d. The control loop runs at 20 Hz and the gain is $\text{diag}(0.1, 0.1, 0.1)$ for both motion goals. The tracking performance is shown in Figure 9.17a and Figure 9.17b.

9.5.2 Whole-Body Manipulation

A configuration with fourteen CKBot UBar modules is constructed in a simulation environment shown in Figure 9.18a. The base module $\bar{m} = m_1$ is fixed to the world frame $\mathcal{W}$. There are two obstacles in the workspace which are close to the robot. The sphere-tree construction outputs 126 obstacle spheres in total to approximate these two obstacles. Frame $\mathcal{F}_1$ and $\mathcal{F}_2$ are attached to connector $\mathcal{T}$ of module m_9 and module m_{14} respectively. Similarly, Chain $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_1$ and $G_K : \mathcal{W} \rightsquigarrow \mathcal{F}_2$ share four modules. The task is to control $p_{\mathcal{F}_1}$ and $p_{\mathcal{F}_2}$ to new locations between these two obstacles (Figure 9.18f). The control loop runs at 20 Hz and the gain is $\text{diag}(0.1, 0.1, 0.1)$ for both motion goals. In this complex scenario, the quadratic program can be constructed and solved by Gurobi [39] in 6.3 ms in average with standard deviation of 2.3 ms and a maximum time of 15.5 ms on a laptop computer (Intel Core i7-8750H CPU, 16GB RAM).

Initially the motions of $p_{\mathcal{F}_1}$ and $p_{\mathcal{F}_2}$ are symmetric because modules are all not very

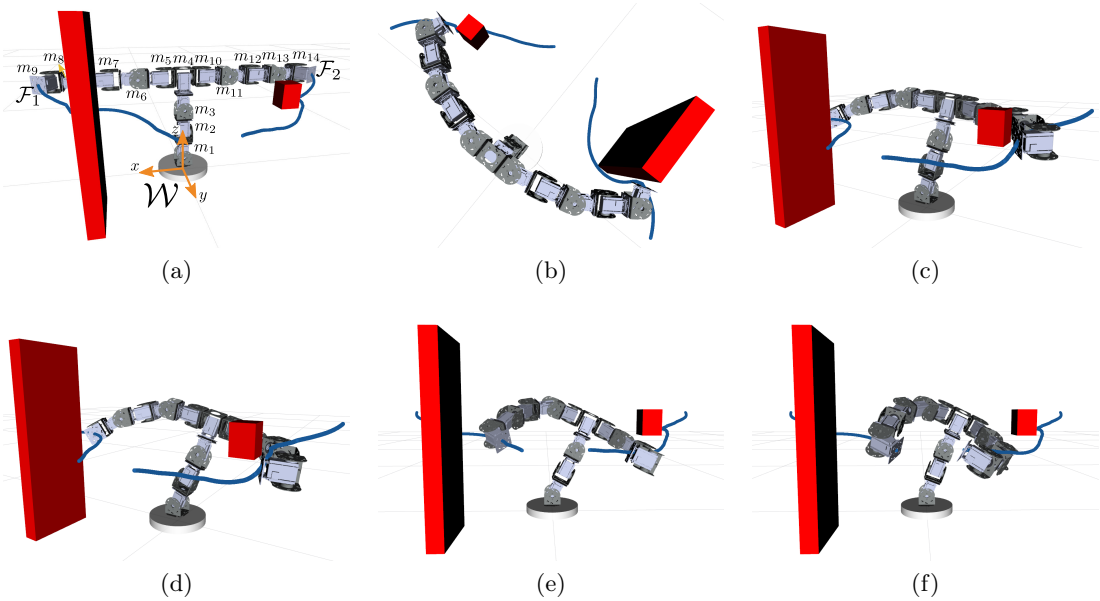
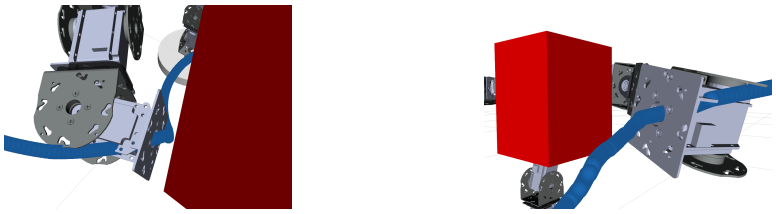
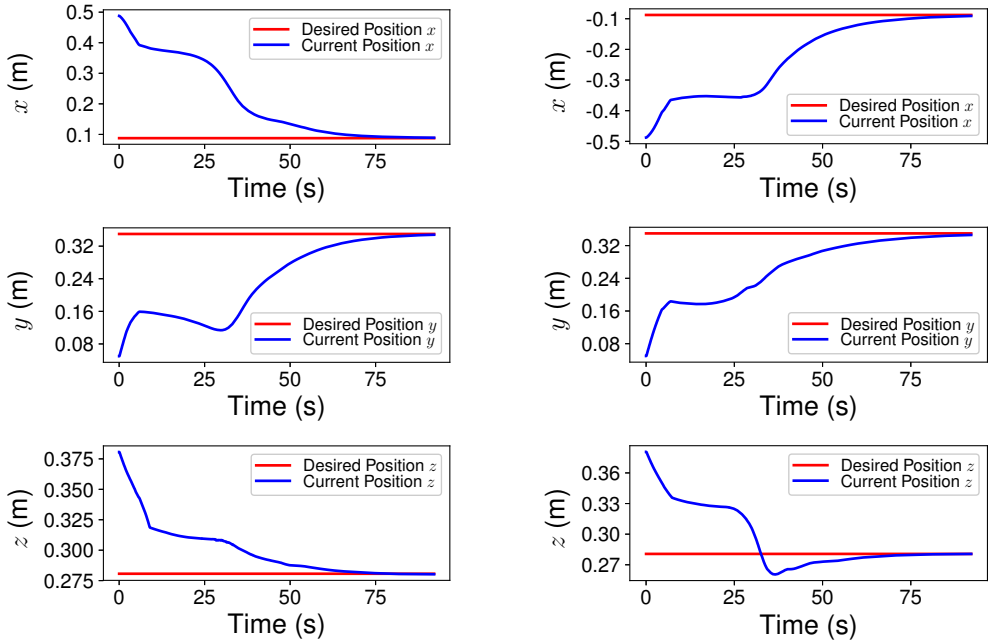


Figure 9.18: Control $p_{\mathcal{F}_1}$ and $p_{\mathcal{F}_2}$ from the initial poses (a) to new locations between the obstacles (f). The body composed by module m_1, m_2, and m_3 first moves backward a little bit (b) and then moves to one side in order to help $\mathcal{F}_1$ and $\mathcal{F}_2$ to go around obstacles (c) — (e). After going around obstacles, both frames can navigate quickly to their destinations. The planned trajectories are shown as blue lines.



(a) (b)

Figure 9.19: (a) Module m_9 approaches an obstacle. (b) Module m_{14} approaches an obstacle.



(a) (b)

Figure 9.20: (a) The motion of $p_{\mathcal{F}_1}$. (b) The motion of $p_{\mathcal{F}_2}$.

close to obstacles. At this stage, the main body composed by module m_1, m_2, and m_3 moves backward slightly. Frame $\mathcal{F}_1$ approaches one of the obstacle first (Figure 9.18b and Figure 9.19a), and the objective function is updated to penalize the motion of m_9 which has to move along the obstacle. Then module m_{14} approaches the other obstacle (Figure 9.18c and Figure 9.19b), and a penalty term for this module is also added to the objective function. Both frames move slowly during this phase (Figure 9.18c and Figure 9.18d). Figure 9.20 shows that $p_{\mathcal{F}_1}$ and $p_{\mathcal{F}_2}$ change slowly. Repulsive motion constraints are added to the optimization function when some module nearly contact obstacles. The main body leans to

one side to help both frames to go around obstacles. After moving around obstacles slowly, both frames can quickly navigate to their destinations (Figure 9.18e and Figure 9.18f). The final planned trajectory takes 92.15 s.

9.6 Conclusion

This chapter presents a new approach to online manipulation motion planning well suited for reconfigurable modular robot systems. This approach formulates the motion planning problem as a sequential quadratic program. A novel way is proposed to approximate obstacles in the environment considering both accuracy and simplicity so that the obstacle-avoidance requirement can be modeled as a small number of linear constraints. The objective function and constraints are updated according to the current scenario in order to handle a larger range of tasks. All motion constraints are linear that allows the approach to be applied to real-time control. Multiple strongly coupled motion tasks can be handled easily which is particularly useful for modular robots.

Part IV

Variable Topology Truss

Introduction

This part presents the research work for VTT, a self-reconfigurable modular robot in truss structure. A VTT is constructed by edge modules with ends being reconfigurable nodes. The linear actuators used in a VTT has a high extension ratio which allows dramatic change of the overall shape. In addition, the connections among edge modules can be changed by splitting or merging nodes which can change the topology of a VTT. These features enable promising applications of the system but also complicates the design, control, and planning.

Chapter 10 introduces some basic concepts for VTTs, the general modeling approach of an arbitrary VTT, as well as the control strategy. The state of every node and the connections among edge modules can be embedded in a graph. A VTT is in essence a parallel robot, and a general kinematics model and a controller are derived in this chapter.

Chapter 11 explores the advantage of topology reconfiguration. This special capability that can rearrange the connections among edge modules is able to significantly increase motion dexterity. A new representation of an arbitrary VTT in a space defined according to the overall shape of the truss is presented which leads to an efficient search for motion planning.

Chapter 12 delves into the configuration space for VTT nodes. As a parallel robot, it is easier to plan the motions of nodes rather than do planning directly in the joint space. The configuration space of a node in a VTT is complex mainly due to the unstructured obstacles which are the members of the truss. A fast algorithm to compute this configuration space is provided so that motion planning for a single node can be solved easily.

Chapter 13 presents the reconfiguration planning framework for the VTT platform.

Given a motion task that is to change the shape of a VTT, the planner can efficiently find the paths for all involved nodes, considering all hardware constraints. The main challenges are that motions of multiple nodes are strongly coupled and the number of topology reconfiguration actions can be large. The proposed planner is shown to be effective to address these challenges.

Chapter 14 deals with the locomotion tasks for VTT robots. Due to the complexity of the mechanical design, it is preferred to have a non-impact locomotion planner to drive the robot. A novel non-impact rolling locomotion planner is derived in this chapter which can efficiently solve locomotion tasks while avoiding receiving impacts from the environment.

Chapter 10

Configuration, Kinematics, and Control

This chapter presents the modeling approach and the control strategy for an arbitrary VTT. Some contents are excerpted from [83].

A VTT is a modular truss robot. The spatial locations of all these edge modules and the connections among them determine the structure or the configuration of a VTT. All the information can be encoded in a graph representation easily. As a parallel robot, when changing the configuration of a VTT, it is more straightforward to control the motions of nodes rather than control the member lengths directly. This chapter derives the general kinematics model and builds the controller to guarantee the motions of these moving nodes. This kinematics model and the control technique can be applied to parallel robots with similar structures.

10.1 Introduction

Modular truss robots are different from lattice and chain type systems in that the systems are made up of beams that typically form parallel structures. In addition to the capability to control the shape or geometry of a truss, a VTT can also self-reconfigure the connections between truss members. A significant advantage for self-reconfigurable modular robots over

other robots with fixed morphologies is their versatility, namely they are able to adapt themselves into different morphologies with respect to different requirements. For example, a VTT in which the members form a broad supported structure is well suited for shoring buildings or structures after disasters, while another truss with some members protruding to form an arm that has a large reachable workspace is good at manipulation tasks.

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

Thus, there are two types of reconfiguration motions: *geometry reconfiguration* and *topology reconfiguration*. Geometry reconfiguration involves changing the lengths of edge modules in a VTT leading to the motion of the corresponding nodes. In order to guarantee the controllability of every node on a VTT, it is necessary that every node should have at least three members attached (each node is of degree three or higher). Topology reconfiguration involves changing the connectivity among members. This reconfiguration behavior happens at nodes. A single node controlled by six members or more can be split into two separate nodes, and two separate nodes can dock to form a single node (Figure 10.1). It is shown

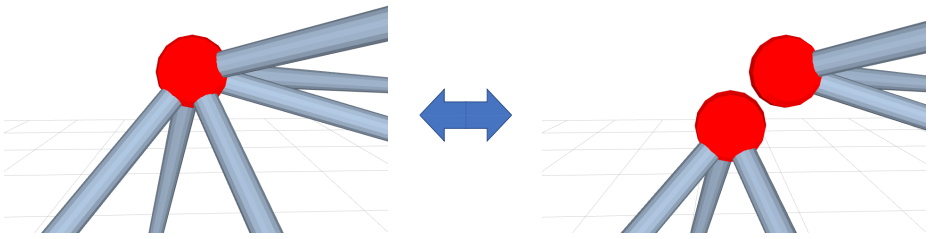


Figure 10.1: A single node with six members can be split into a pair of nodes and two separate nodes can also merge into a single node.

that reconfigurable VTT systems have at least eighteen members [132].

As VTTs are inherently parallel robots, it is much easier to solve the inverse kinematics than the forward kinematics. So, for the geometry reconfiguration, rather than controlling the active DOFs that are the member lengths, do the control over the motion of the *nodes* and then do the inverse kinematics to easily determine member lengths.

10.2 Configuration

A VTT is a special type of parallel robot — constructed by linear actuators connected at special node joints. Hence, the structure of a VTT can be modeled as an undirected graph $G = (V, E)$ where V is the set of vertices of G and E is the set of edges of G: each member can be regarded as an undirected labeled edge $e \in E$ of the graph and every intersection among members can be treated as a vertex $v \in V$ of the graph. The position of every node $v \in V$ is encoded as its property `Pos` such that $v\,[\texttt{Pos}] = [v_x, v_y, v_z]^\intercal \in \mathbb{R}^3$. Let $q^v = v\,[\texttt{Pos}]$ and the *configuration space* of node v denoted as $\mathcal{C}^v$ is simply $\mathbb{R}^3$. In this way, the state of an edge module $e = (v_1, v_2) \in E$ is fully defined by `Pos` properties of its two vertices v_1 and v_2. The position of a given node $v \in V$ is controlled by changing the lengths of all attached members denoted as $E^v \subseteq E$. E^v can be regarded as a parallel robot with all edge modules being joint actuators.

In general, given a VTT $G = (V, E)$, motions of nodes are controlled by all attached actuated members and the system is usually an overconstrained parallel robot. For a motion task, the set of all nodes V is separated into two groups: V_F and V_C where V_F contains all the fixed or stationary nodes and V_C contains all the controlled nodes. For example, given the VTT shown in Figure 10.2, when controlling the motion of node v_1 and node v_4, $V_C = \{v_1, v_4\}$ and $V_F = \{v_0, v_2, v_3, v_5\}$, and this is a 6-DOF system since there are two controlled nodes. In addition, this system is overconstrained and the motions of the two nodes are controlled by seven members. Note that V_F and V_C are not constant and they can be changed during the motion of a VTT (reconfiguration and locomotion).

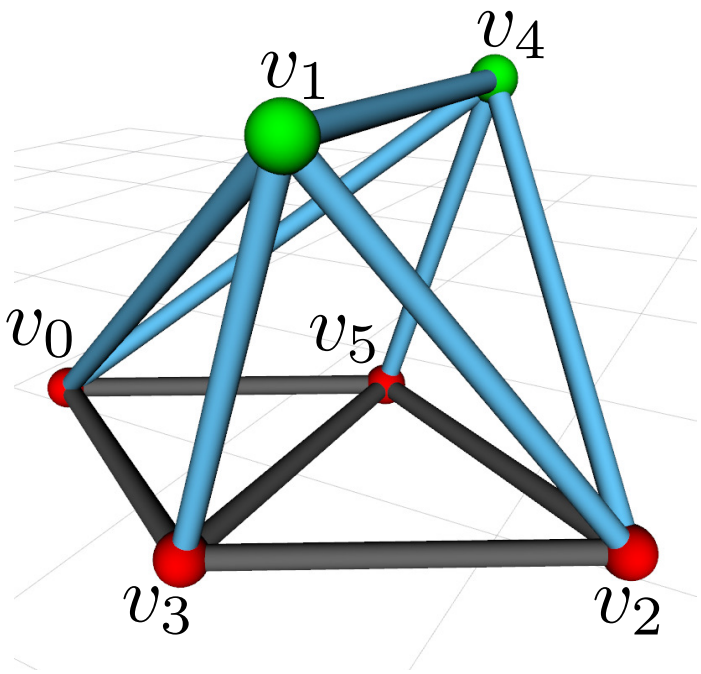


Figure 10.2: A VTT is composed of twelve members. Currently, the motion of node v_1 and node v_4 are under control by seven blue members.

10.3 Kinematics

Given V_C with Λ nodes under control, the position vector of this system is simply the stack of q^v where $v \in V_C$, and this system is controlled by all members that are attached with these nodes, namely $\bigcup_{v \in V_C} E^v$. Let l_{ij} be the *link vector* from a controlled node v_i pointing to any node v_j. There are two types of link vectors: 1. if $v_j \in V_F$, then l_{ij} is an *attachment link vector*; 2. if $v_j \in V_C$, then l_{ij} is a *connection link vector*. The link vector satisfies the following equation:

$$l_{ij} = q^{v_j} - q^{v_i} \tag{10.1}$$

in which $v_i \in V_C$. Taking time derivative of Eq. (10.1) for an attachment link vector and a connection link vector yields:

$$l_{ij}^{\intercal} \dot{l}_{ij} = (q^{v_i} - q^{v_j})^{\intercal} \dot{q}^{v_i} \quad \forall v_j \in V_F \tag{10.2a}$$

$$\dot{l}_{ij} = \dot{q}^{v_j} - \dot{q}^{v_i} \quad \forall v_j \in V_C \tag{10.2b}$$

Assume $V_C = \{\bar{v}_\alpha | \alpha = 1, 2, \cdots, \Lambda\}$, and for a controlled node $\bar{v}_\alpha$, all the fixed nodes in its neighborhood $\mathcal{N}_G(\bar{v}_\alpha)$ are denoted as $\hat{v}_1^\alpha, \hat{v}_2^\alpha, \cdots, \hat{v}_N^\alpha$, and the corresponding attachment link vectors are denoted as ${}^\alpha\hat{l}_1, {}^\alpha\hat{l}_2, \cdots, {}^\alpha\hat{l}_N$. Eq. (10.2a) is true for any ${}^\alpha\hat{l}_t$ where $t \in \{1, 2, \cdots, N\}$, so it can be rewritten into the following form for a controlled vertex v_i:

$$B_\alpha \dot{L}_\alpha = A_\alpha \dot{q}^{\bar{v}_\alpha} \tag{10.3}$$

in which

$$\dot{L}_\alpha = \begin{bmatrix} {}^\alpha\dot{\hat{l}}_1^\intercal & {}^\alpha\dot{\hat{l}}_2^\intercal & \cdots & {}^\alpha\dot{\hat{l}}_N^\intercal \end{bmatrix}_{3N\times 1}^\intercal$$

$$B_\alpha = \begin{bmatrix} {}^\alpha\hat{l}_1^\intercal & 0 & \cdots & 0 \\ 0 & {}^\alpha\hat{l}_2^\intercal & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & {}^\alpha\hat{l}_N^\intercal \end{bmatrix}_{N\times 3N}$$

$$A_\alpha = \begin{bmatrix} q^{\bar{v}_\alpha} - q^{\hat{v}_1^\alpha} & q^{\bar{v}_\alpha} - q^{\hat{v}_2^\alpha} & \cdots & q^{\bar{v}_\alpha} - q^{\hat{v}_N^\alpha} \end{bmatrix}_{N\times 3}^\intercal$$

If another controlled node $\bar{v}_\beta \in \mathcal{N}_G(\bar{v}_\alpha)$, namely $\bar{v}_\beta$ is adjacent to $\bar{v}_\alpha$, then

$$\dot{\bar{l}}_{\alpha\beta} = \dot{q}^{\bar{v}_\beta} - \dot{q}^{\bar{v}_\alpha} \tag{10.4}$$

Combining Eq. (10.3) and Eq. (10.4), the following equation of motion can be derived:

$$\mathcal{B}\dot{\mathcal{L}} = \mathcal{A}\dot{p} \tag{10.5}$$

in which

$$p = \begin{bmatrix} (q^{\bar{v}_1})^\intercal & (q^{\bar{v}_2})^\intercal & \cdots & (q^{\bar{v}_\Lambda})^\intercal \end{bmatrix}^\intercal$$

$$\dot{\mathcal{L}} = \begin{bmatrix} \dot{L}_1^\intercal & \dot{L}_2^\intercal & \cdots & \dot{L}_\Lambda^\intercal & \cdots & \dot{\bar{l}}_{\alpha\beta} & \cdots \end{bmatrix}^\intercal$$

$$\mathcal{B} = \mathrm{diag}(B_1, B_2, \cdots, B_\Lambda, \cdots, I, \cdots)$$

$$\mathcal{A} = [\mathcal{A}_1, \mathcal{A}_2]^\intercal$$

$$\mathcal{A}_1 = \mathrm{diag}(A_1, A_2, \cdots, A_\alpha, \cdots, A_\beta, \cdots, A_\Lambda)$$

$$\mathcal{A}_2 = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{3\times 3\alpha} & I_{3\times 3} & 0_{3\times(\beta-\alpha-1)} & -I_{3\times 3} & 0_{3\times 3(\Lambda-\beta)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

The size of $\mathcal{A}_2$ is determined by the number of the controlled vertices and the connection link vectors (if there are ω connection link vectors, $\mathcal{A}_2$ is a $3\omega \times 3\Lambda$ matrix). For the system shown in Figure 10.2, $\mathcal{A}_2 = [I, -I]_{3\times 6}$ since there are only two controlled nodes and one connection link vector. It can be shown that $\mathcal{B}$ has full rank as long as there are no zero-length members whereas $\mathcal{A}$ may not have full rank. This does make sense because there must exist a set of link velocities given the velocities of all controlled nodes, but some link velocities may result in invalid motions of nodes since the system is overconstrained.

Eq. (10.5) can be rearranged in two ways:

$$\dot{\mathcal{L}} = \mathcal{B}^+ \mathcal{A} \dot{p} = J_{\mathrm{BA}} \dot{p} \tag{10.6a}$$

$$\dot{p} = \mathcal{A}^+ \mathcal{B} \dot{\mathcal{L}} = J_{\mathrm{AB}} \dot{\mathcal{L}} \tag{10.6b}$$

where $\mathcal{B}^+$ and $\mathcal{A}^+$ are the pseudo-inverse of $\mathcal{B}$ and $\mathcal{A}$ respectively, and both J_{BA} and J_{AB} matrices are the *Jacobian*. These two equations are used to describe the relationship between the link velocities and the controlled node velocities. J_{BA} is always defined (as long as there is no zero-length member), but J_{AB} may not be defined. Given $\dot{p}$ is known, Eq. (10.6a) gives the minimum norm solution to Eq. (10.5), namely minimizing $\|\dot{\mathcal{L}}\|$. On the other hand, Eq. (10.6b) results in the unique least square solution to Eq. (10.5) if $\dot{\mathcal{L}}$ is known, namely minimizing $\|\mathcal{B}\dot{\mathcal{L}} - \mathcal{A}\dot{p}\|$.

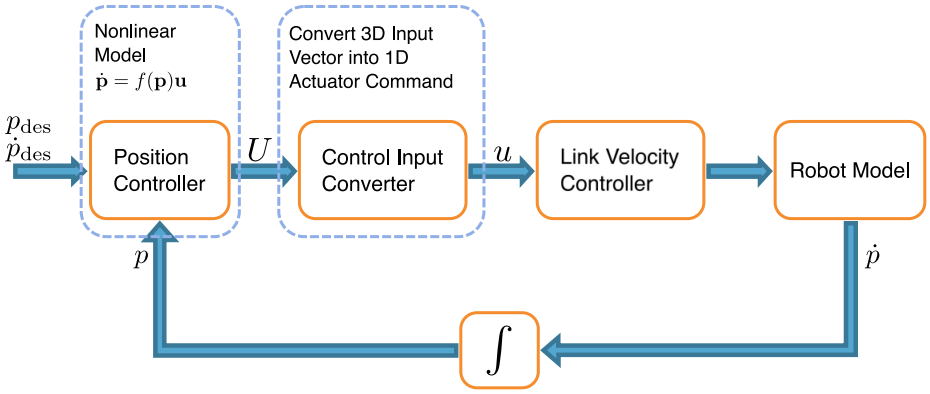


Figure 10.3: The control loop for position control.

10.4 Control

A control framework is proposed in this section to control a VTT. Given a VTT $G = (V, E)$ and the current V_C and V_F, robot position state p is defined, and the framework to control p is shown in Figure 10.3.

The error on the state and its derivative are defined as

$$e = p_{\text{des}} - p \tag{10.7a}$$

$$\dot{e} = \dot{p}_{\text{des}} - \dot{p} \tag{10.7b}$$

Let the controller input $U = \dot{l}$, a continuous time system can be defined from Eq. (10.6b) as

$$\dot{p} = J_{\text{AB}} U \tag{10.8}$$

The error is expected to converge exponentially to zero, then the following equation is derived:

$$\dot{p}_{\text{des}} - J_{\text{AB}} U + K_{\text{p}}(p_{\text{des}} - p) = 0 \tag{10.9}$$

resulting in

$$J_{\text{AB}} U = \dot{p}_{\text{des}} + K_{\text{p}}(p_{\text{des}} - p) \tag{10.10}$$

in which K_{p} is a positive definite gain matrix. Since this arm is an overconstrained system, the solution of U is not unique. This difficulty is resolved by solving the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}U^{\intercal}U \\ \text{subject to} \quad & J_{\mathrm{AB}}U = \dot{p}_{\mathrm{des}} + K_{\mathrm{p}}(p_{\mathrm{des}} - p) \end{aligned} \tag{10.11}$$

and the solution is given by

$$U = J_{\mathrm{AB}}^{+}(\dot{p}_{\mathrm{des}} + K_{\mathrm{p}}(p_{\mathrm{des}} - p)) \tag{10.12}$$

where J_{AB}^{+} is the pseudo inverse of J_{AB} and the solution is the minimum joint speed solution. Then given the desired state p_{des} and desired velocity $v_{\mathrm{des}} = \dot{p}_{\mathrm{des}}$, the controller input U that guarantees that the state approaches p_{des} in exponential time can be computed. This control input U is the stack of all control input U_{ij} which is the desired derivative of l_{ij}.

Given this continuous time system defined in Eq. (10.8) and the control input U, the following can be derived:

$$p(t + dt) \approx p(t) + \dot{p}(t)dt = p(t) + J_{\mathrm{AB}}(t)Udt \tag{10.13}$$

In addition, for each link vector l_{ij} satisfying Eq. (10.1), the following can be derived:

$$d\|l_{ij}\| \approx \|p_j(t + dt) - p_i(t + dt)\| - \|p_j(t) - p_i(t)\| \tag{10.14}$$

If l_{ij} is an attachment link vector, namely $v_j \in V_F$, then $p_j(t + dt) = p_j(t)$.

For each link vector l_{ij} and the corresponding control input U_{ij} which is a 3×1 vector in U, actuator command u_{ij} can be computed by

$$u_{ij} = \begin{cases} \|U_{ij}\| & \text{if } d\|l_{ij}\| \geq 0 \\ -\|U_{ij}\| & \text{if } d\|l_{ij}\| < 0 \end{cases} \tag{10.15}$$

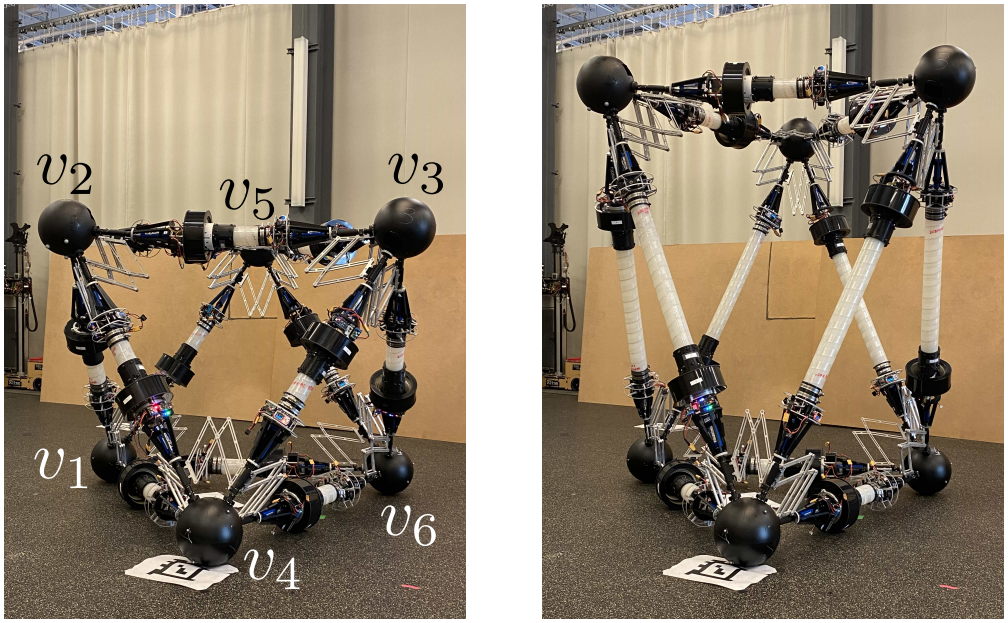
which contains the information of the speed and the moving direction for the corresponding edge module. A PID controller for the edge module can be built to control its velocity with the feedback of the length measurement.

10.5 Experiments

A VTT $G = (V, E)$ built by twelve edge modules is shown in Figure 10.4a, in which $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $E = \{(v_3, v_5), (v_2, v_3), (v_3, v_4), (v_3, v_6), (v_2, v_5), (v_1, v_6), (v_1, v_4), (v_2, v_4), (v_1, v_2), (v_1, v_5), (v_5, v_6), (v_4, v_6)\}$. The initial location of each node is shown in the following:

$$v_1\,[\texttt{Pos}] = [-0.193, -5.348, 0.094]^\intercal \quad v_2\,[\texttt{Pos}] = [0.384, -4.968, 1.049]^\intercal$$
$$v_3\,[\texttt{Pos}] = [-0.243, -3.971, 1.066]^\intercal \quad v_4\,[\texttt{Pos}] = [0.359, -4.286, 0.093]^\intercal$$
$$v_5\,[\texttt{Pos}] = [-0.787, -5.020, 1.051]^\intercal \quad v_6\,[\texttt{Pos}] = [-0.811, -4.337, 0.084]^\intercal$$

The control task is to increase the height of v_2, v_3, and v_5 by 50 cm shown in Fig-



(a) (b)

Figure 10.4: (a) A VTT is constructed by twelve edge modules. (b) v_2, v_3, and v_5 are moved to higher locations.

ure 10.4b. In this task, $V_F = \{v_1, v_4, v_6\}$ and $V_C = \{v_2, v_3, v_5\}$. In the kinematics model for this control task, there are six attachment link vectors — l_{12}, l_{15}, l_{24}, l_{34}, l_{56}, l_{36}, and three connection link vectors — l_{23}, l_{25}, l_{35}. These three moving nodes are controlled to move along straight-line trajectories which are cubic polynomials, and the controller tracking performance is shown in Figure 10.5. The control loop runs at 20 Hz and the gain is diag(1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5). The member velocity controller is running at

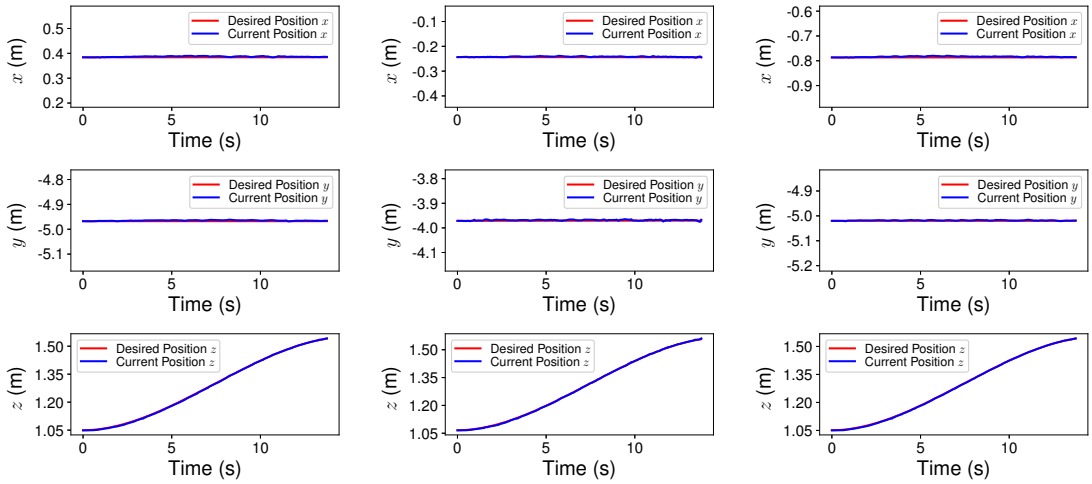


Figure 10.5: Tracking performance for q^{v_2} (a), q^{v_3} (b), and q^{v_5} (c).

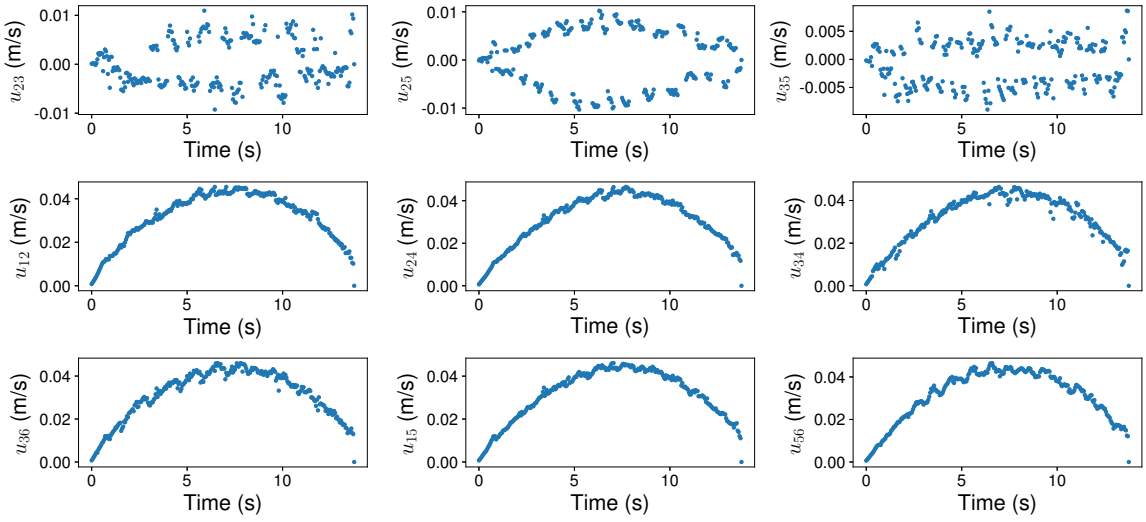


Figure 10.6: Control input of every link vector.

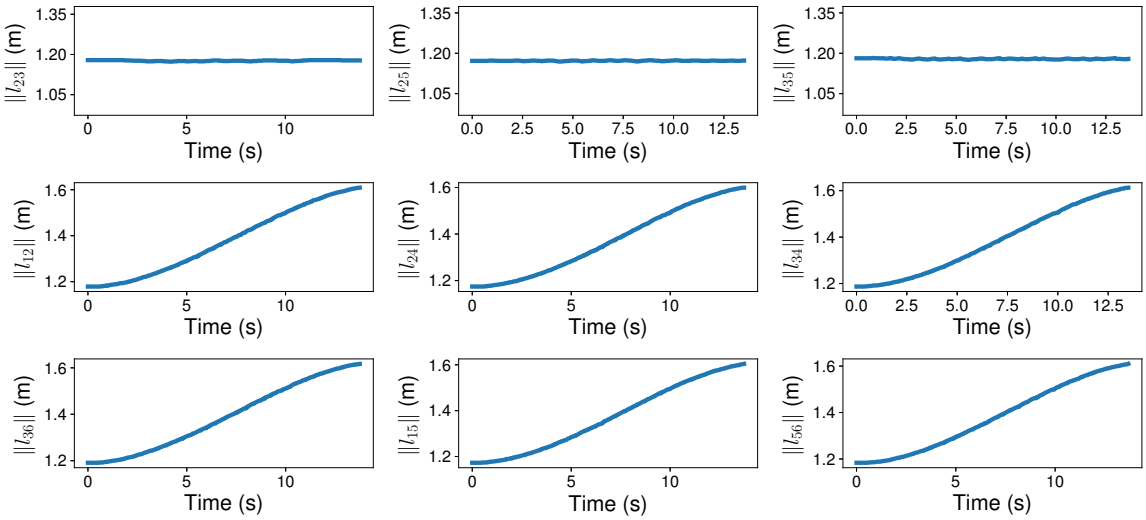


Figure 10.7: Length of every moving edge module.

100 Hz. During the task, the x position and the y position of every controlled node can be maintained. The computed control input for the task is shown in Figure 10.6 and the length of every member is shown in Figure 10.7. The commanded velocities for edge module (v_2, v_3), (v_2, v_5), and (v_3, v_5) are oscillating around zero slightly and their lengths are almost fixed since the desired relative positions among v_2, v_3, and v_5 are not changing. The lengths of all other involved edge modules are increasing as demand, and their velocities are increasing initially and then decreasing to zeros. When all these three nodes are close enough to their destinations respectively, the controller stops all actuators.

10.6 Conclusion

In this chapter, the configuration of a VTT is defined and the graph model is proposed to describe an arbitrary VTT. This chapter also derives a general kinematics model for an arbitrary VTT motion task and provides a control framework to execute motions.

Chapter 11

Topology Reconfiguration Advantage

A VTT is capable of altering its topology by merging or splitting nodes that can significantly enhance its flexibility. Some motion that is blocked by members can be achieved by changing the topology properly. This chapter explores the advantage of topology reconfiguration and presents a planning framework to make use of this capability inspired by the DNA replication process. The contents in this chapter excerpt heavily from [75].

The variable topology truss (VTT) is an extension of an existing class of robots, the variable geometry truss (VGT). A VTT has the additional capability to change its topology through self-reconfiguration. A single node with enough edge modules can be split into a pair of nodes and two separate nodes can be merged to become an individual one. This self-reconfiguration ability enables more versatile motions and results in more potential applications for this type of robot in unstructured environments, such as space. This chapter explores dexterous motions inspired by the DNA replication process — the topology of DNA can be changed by cutting and resealing strands as tanglements form. These motions can be planned efficiently based on a novel way to model the robot in a nonuniform grid space and a simple local planner for collision avoidance. This approach significantly simplifies the problem and some experiment results show that the complicated problem can be solved in a reasonable time.

11.1 Introduction

The variable geometry truss (VGT) [91] is a class of modular robotic systems in a truss structure and the truss members have variable lengths. A variable *topology* truss (VTT) is similar to a VGT with the additional capability to self-reconfigure the attachment of members at the nodes, changing its topology by merging or splitting nodes (Figure 10.1). Two separate nodes in the truss can dock to form one node which connects all of the involved members. Similarly, a single node with a sufficient amount of members can undock into a pair of nodes.

Since topological reconfigurations also happen at nodes, all actions are considered to occur at nodes. For example, if a set of edge modules E_1 intersect at node v_1, and another set of edge modules E_2 intersect at node v_2, then the states of these edge modules can be changed by manipulating node v_1 and v_2, including changing the locations of them, merging v_1 and v_2, or splitting if v_1 and v_2 are already merged. For some simple cases, translating the nodes small distances in the space can happen without collision. However, in many instances translating nodes will often lead to the truss members colliding with other truss members. In many of those cases, there may not be a path for that node to reach a goal location without collision given the topology of the VTT configuration. Traditional motion planning methods would require analysis of the motion, geometric reasoning of the topological spaces in which collision can occur, and reconfiguration planning to change the topology if required.

This chapter presents a novel method of motion planning combined with limited reconfiguration to enable motions of nodes without regard to internal collision via reconfiguration. The advantage is from topology reconfiguration capability. This behavior is similar to DNA replication in which topoisomerase can change the topology of DNA by cutting and resealing strands as tanglements form [18]. The algorithm to solve this fundamental reconfiguration problem is also inspired by this DNA motion. A new method to discretize the space nonuniformly is proposed so that discrete actions can be applied for a motion task and the space can be explored efficiently. This method is the combination of the exact cell decomposition method and the approximation method. The cells are determined based on truss geometry

and then are subdivided into octants. Different from common octree decomposition [44], cells are not necessarily to be cubic. Collision-free actions can be computed efficiently and a sequence of optimal reconfiguration actions are generated by a simple graph search algorithm. This algorithm is demonstrated on some motion tasks and the necessary analysis is provided.

As mentioned in Chapter 10, the structure of a VTT can be modeled as an undirected graph $G = (V, E)$ where V is the set of vertices of G and E is the set of edges of G. Each graph edge is correspondent to a truss member and each graph vertex is correspondent to a truss node. Because of the topology reconfiguration, the number of nodes can change. However, the number of members in a system are physical elements that cannot merge or disappear so that the number must remain constant.

In this chapter, the fundamental reconfiguration problem can be stated. For a variable topology truss $G = (V, E)$, the goal is to change the state of edge module $e \in \widehat{E}$ from its initial state to its goal state in which $\widehat{E} \subset E$. However, as the discussion above, edge modules have to be controlled in form of a group because their states can only be modified by manipulating the involved nodes. For any $e = (v_1, v_2) \in E$ in a VTT configuration $G = (V, E)$, let $e\,[v_1] = v_1\,[\texttt{Pos}]$ and $e\,[v_2] = v_2\,[\texttt{Pos}]$, then the current state of this edge module e is fully defined by $e\,[v_1]$ and $e\,[v_2]$. Assume $\forall e \in \widehat{E}$ intersect at node $\hat{v}$, then the motion task is, $\forall e \in \widehat{E}$, changing $e\,[\hat{v}]$ from $[x_i, y_i, z_i]^\intercal$ to $[x_g, y_g, z_g]^\intercal$.

11.2 Motion Planning Algorithm

11.2.1 Grid Space Model

Planning in discrete space makes it possible to efficiently plan a sequence of discrete actions for a complicated motion task. However, when discretizing space, discretization resolution is an important issue. There is a trade-off for different discretization resolutions: too fine resolution results in a large search space — too coarse resolution may bring about no solution for a motion task even if there exists a valid path. This problem is especially significant for modular robots, such as a VTT, which may have high dimensional spaces. Here a novel

way is presented to represent the system in a discretized workspace for a given VTT which is called the *grid space*.

The automatically generated grid space has a stepsize that adapts depending on the density of the workspace. A small exploration stepsize is used for high occupancy subspace and a large exploration stepsize for low occupancy subspace. The density is a function of the node positions of both the initial VTT configuration and a goal VTT configuration. Let V be the set containing all nodes with unique locations and its size is N. Firstly all of these node locations are extracted in x-axis, y-axis, and z-axis respectively:

$$X = \{v_x | v \in V\}$$
$$Y = \{v_y | v \in V\}$$
$$Z = \{v_z | v \in V\}$$

Sort X, Y, and Z in nondecreasing order to generate three sequences $\widehat{X}$, $\widehat{Y}$, and $\widehat{Z}$ respectively. For each sequence, if two adjacent elements are the same or very close, keep only one element. Use the parameter δ to set the threshold for closeness. Note that δ will often be some fraction of the size of the truss. Then, for every two adjacent elements (e.g. x_i and x_j), insert a midpoint between them, e.g. $(x_i + x_j)/2$, into the list as an intermediate position. This will subdivide each cell in space into octants like octree decomposition. With the final three sequences generated as follows:

$$\widehat{X} = \{x_0, x_1, \cdots, x_{N_x}\}$$
$$\widehat{Y} = \{y_0, y_1, \cdots, y_{N_y}\}$$
$$\widehat{Z} = \{z_0, z_1, \cdots, z_{N_z}\}$$

the following conversions from this grid space to Cartesian space in x-axis, y-axis, and z-axis can be obtained respectively:

$$f_x(i_x) = x_{i_x} \quad i_x = 0, 1, \cdots, N_x \tag{11.1a}$$

$$f_y(i_y) = y_{i_y} \quad i_y = 0, 1, \cdots, N_y \tag{11.1b}$$

$$f_z(i_z) = z_{i_z} \quad i_z = 0, 1, \cdots, N_z \tag{11.1c}$$

the inverse conversions (from Cartesian space to the grid space) are

$$f_x^{-1}(x) = \underset{i_x \in \{0,1,\cdots,N_x\}}{\arg\min} |x - x_{i_x}| \tag{11.2a}$$

$$f_y^{-1}(y) = \underset{i_y \in \{0,1,\cdots,N_y\}}{\arg\min} |y - y_{i_y}| \tag{11.2b}$$

$$f_z^{-1}(z) = \underset{i_z \in \{0,1,\cdots,N_z\}}{\arg\min} |z - z_{i_z}| \tag{11.2c}$$

Then the conversion from the grid space to Cartesian space can be defined as

$$f([i_x, i_y, i_z]^\intercal) = \left[x_{i_x}, y_{i_y}, z_{i_z}\right]^\intercal \tag{11.3}$$

and the conversion from Cartesian space to the grid space can be defined as

$$f^{-1}([x, y, z]^\intercal) = \left[f_x^{-1}(x), f_y^{-1}(y), f_z^{-1}(z)\right]^\intercal \tag{11.4}$$

Note that this is a nonuniform grid space. Given a VTT configuration $G = (V, E)$ in Cartesian space, there is a corresponding VTT configuration $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ in the grid space. A two-dimensional example is shown in Figure 11.1. Both Cartesian space and the grid space coordinates are shown. It is apparent that, given a conversion between Cartesian space and a grid space, different VTT configurations in Cartesian space may results in the same VTT configuration in the grid space.

A truss example is shown in Figure 11.2a and the equivalent truss in a grid space is shown in Figure 11.2b. In Cartesian space, the truss is a slightly deformed cube. The mapping in the grid space is a regular cube with eight cells generated inside the cube.

Note that the non-uniform resolution is different from the octree decomposition where a subdivision results in cells that are equally divided into cubes. Here the x-axis, y-axis, and

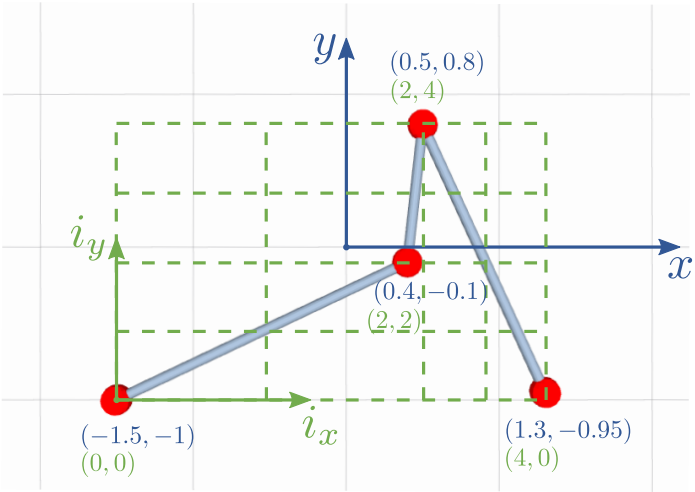


Figure 11.1: A two dimensional example with four nodes and three members when $\delta = 0.1$ is shown. The generated grids are shown with "- -". The coordinates in Cartesian space are in dark blue color and the coordinates in the grid space are in light green color.

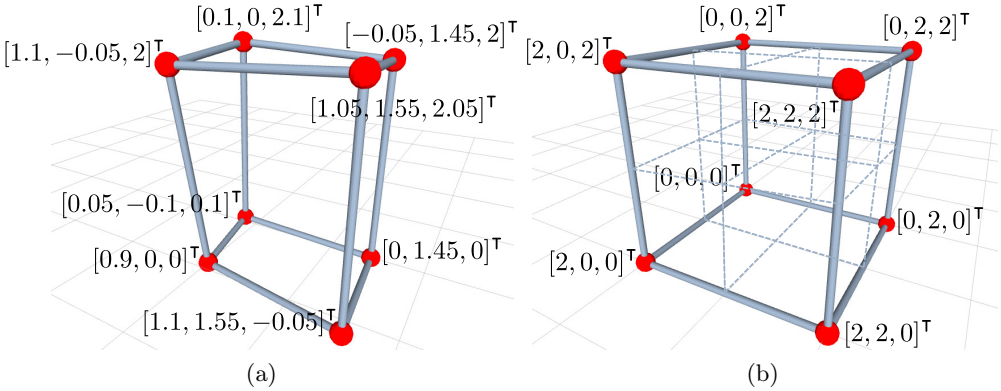


Figure 11.2: (a) A truss in Cartesian space. (b) The equivalent cubic truss in the grid space with $\delta = 0.2$.

z-axis divisions may not be uniform. In addition, since the divisions are derived directly on the location of nodes, there are fewer subdivisions than those that may occur in octree where the number of subdivisions can become large if the nodes are slightly off from an even division.

Once the conversion between Cartesian space and a grid space is computed, a VTT configuration can be simplified — if two nodes are very close and are third-degree neighbors or higher, these nodes can be merged into one. This is determined by the parameter δ. One constraint for δ is that its value cannot be too large since the edge modules have a non-zero

minimum length.

11.2.2 Node Motion Model and Reconfiguration Actions

After the nonuniform grid space is computed, the motion of a node can be modeled as discrete actions. The discrete action can be a geometric reconfiguration action or a topology reconfiguration action. In the grid space, twenty-seven different possible discrete geometric motion actions can be defined for a free node shown in Figure 11.3. The cube centered on the free node can be defined by its two corners: $[0, 0, 0]^\intercal$ and $[2, 2, 2]^\intercal$. Note that the cube in this grid space is not necessarily a cubic shape in Cartesian space.

For a node in a VTT, there are some constraints on the motion that the node can execute. Every node has to be attached to at least three members in order to maintain controllability. If a node is controlled by six or more members, this node can be split into a pair of controllable nodes with multiple ways to separate the involved members. For example, if a node is controlled by six members, there are ten different ways to split six members into two groups of three. However, if a node is controlled by five members, then there is no way to split this single node as both nodes need to have at least three members.

The reconfigurability of nodes can be exploit to ease the collision-free motion planning

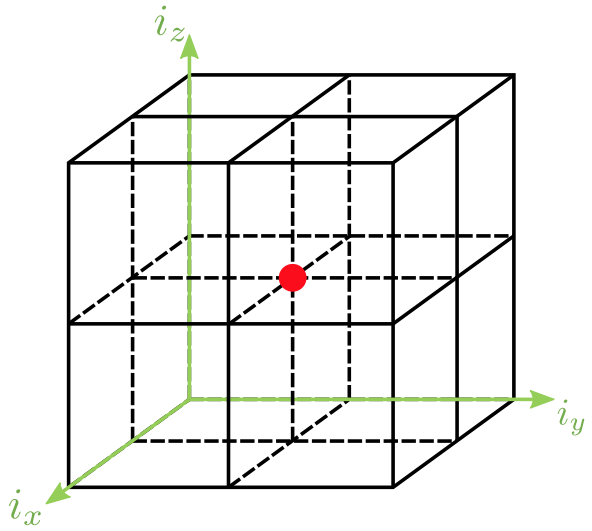


Figure 11.3: In the grid space, a node (•) can move to 27 different locations with one discrete action, defined to be the 27 points of intersection between lines in the figure including the center of the cube that is the no motion action.

problem. If a node moves to a location in the grid space that is already occupied by a node, then the corresponding moving action will end up with a merge action. In this way, there is no need to worry about collisions of nodes. For the VTT system, there are three possible atomic actions for a specific node: `Move`, `Merge` and `Split`. Since only one node may exist in a discrete location in the grid space, there are four different combinations of atomic motion for a node:

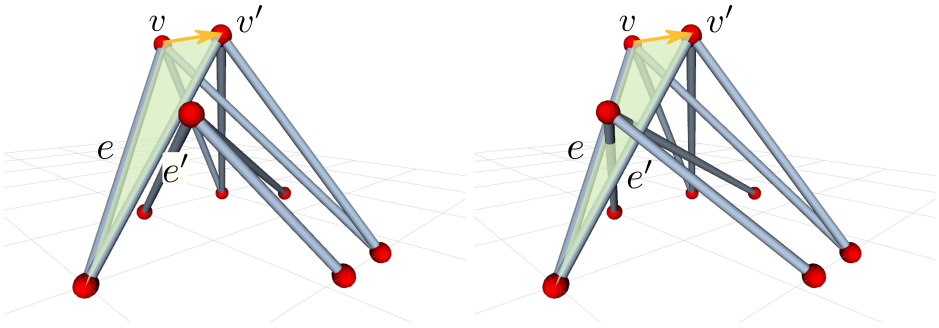
$$a = \begin{cases} \texttt{Move} \\ \texttt{Split} + \texttt{Move} \\ \texttt{Move} + \texttt{Merge} \\ \texttt{Split} + \texttt{Move} + \texttt{Merge} \end{cases} \tag{11.5}$$

in which a is defined to be the *discrete reconfiguration action*. Note that not all actions are executable because some actions may violate constraints resulting in a not valid motion.

As the reconfiguration process proceeds, some nodes may disappear and some nodes may appear. Let $\widehat{V} \subset V$ contain the current nodes intersected by edge modules in $\widehat{E}$. Initially, $\widehat{V} = \{\hat{v}\}$ because all edge modules in $\widehat{E}$ intersect at node $\hat{v}$. When the discrete reconfiguration action occurs, $\widehat{V}$ may change accordingly. For example, if $\hat{v}$ is split into $\hat{v}'$ and $\hat{v}''$, then $\widehat{V} = \{\hat{v}', \hat{v}''\}$, or if $\hat{v}$ is merged with another node $\bar{v}$, then $\widehat{V} = \{\bar{v}\}$. For each node in $\widehat{V}$, the same reconfigurability analysis can be applied to generate all possible actions and the states of the involved edge modules in set $\widehat{E}$ will be changed accordingly. To minimize the search space for all possible actions, in this work, only one node is active in each step, which means if there are multiple nodes in $\widehat{V}$, apply action a for only one node.

11.2.3 Collision

When controlling a node to execute a motion action a, a parallel robot is actually under control to move in a complex environment occupied by many other edge modules. It is not straightforward to compute the collision-free space for this parallel robot directly but it is possible to exhaustively check collision for every pair of members during the motion. For example, for a VTT configuration $G = (V, E)$, node $v \in V$ is going to execute an action a, and the involved member set is E^v. It is required to check whether any $e \in E^v$ will



(a) (b)

Figure 11.4: Light green triangle (△) is sweeped by member e when moving node v to new location v' along the yellow trajectory (→) and the new state of member e is e'. e doesn't collide with any other members in (a) but does collide with two members in (b) during the motion.

collide with any other edge module in set $E \backslash E^v$ during the motion action process. Every edge module can be modeled as a line segment in space, thus, for every $e \in E^v$, naïvely the task is just to check the intersection between a moving line segment and a still line segment. This is actually not easy. With the nonuniform grid space and the node motion model, the node is actually moving along a line segment. Hence, an easier and more efficient method is presented to do collision check.

For a VTT configuration $G = (V, E)$ and a node $v \in V$, once a reconfiguration action a is executed, every moving member in E^v sweeps a triangle area (Figure 11.4a), and if this member $e \in E^v$ collides with another member $\bar{e} \in E \backslash E^v$, then $\bar{e}$ must intersect with the triangle $\triangle$ generated by e (Figure 11.4b). There are two cases: $\bar{e}$ is not parallel to $\triangle$ and $\bar{e}$ is parallel to $\triangle$. For the first case, there are already many efficient algorithms to test the intersection between a line segment and a triangle in 3D space, such as Möller-Trumbore ray-triangle intersection algorithm [92]. For the second case, it is just a simple 2D geometry math problem.

11.2.4 Transition Model

A transition model is required to describe the effect of a reconfiguration action on a VTT configuration. For a VTT configuration $G = (V, E)$, given a discrete reconfiguration action a, a new VTT configuration $G' = (V', E')$ is able to be computed by this transition model

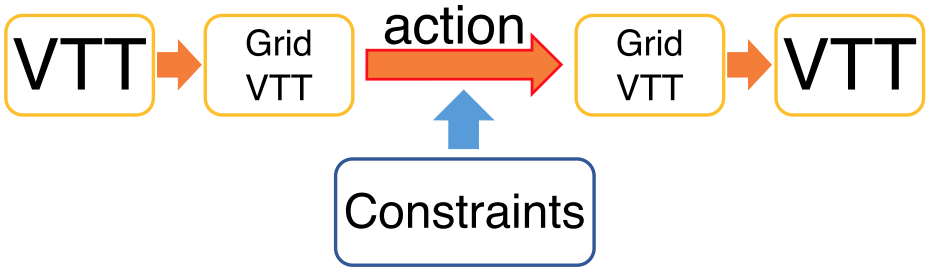


Figure 11.5: Transition model diagram.

$\mathcal{F}(G, a)$ if action a is executable.

The transition model is shown in Figure 11.5. Given a VTT configuration $G = (V, E)$ and a reconfiguration task, the grid space can be computed as described in Section 11.2.1. A corresponding VTT configuration $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ in this grid space can be computed, where for each $v \in V$, its corresponding position in the grid space $\mathbb{v}\,[\texttt{Pos}]$ can be computed by Eq. (11.4). Then the discrete action a can be applied on the corresponding node of the VTT configuration $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ in the grid space to obtain the new states of all involved edge modules. With Eq. (11.3), the new position of the corresponding node (the intersection of these involved edge modules) can be calculated. Constraints need to be satisfied in Cartesian space. Only collision-free criteria are considered here but it is straightforward to add more constraints. An action satisfying all constraints is an *executable reconfiguration action*. There are four different types of reconfiguration actions shown in Eq. (11.5). For geometry reconfiguration actions, just apply the model in Section 11.2.3. For actions containing topology reconfiguration actions, apply collision check for only `Move` action because `Split` and `Merge` never cause any collision. Once the action is verified to be executable, the new VTT configuration in Cartesian space can be obtained.

11.2.5 Graph Search Algorithm

With the transition model, it is straightforward to do motion planning by a graph search algorithm. Two VTT configurations can be connected by an edge if and only if there is an executable reconfiguration action taking the first VTT configuration to the second VTT configuration. The search starts from the initial VTT configuration and stops until the goal

configuration is visited. A graph search algorithm designed based on A* framework is shown in Algorithm 6.

Line 1 — 5: Compute the nonuniform grid space and the equivalent initial VTT configuration and the equivalent goal VTT configuration in this grid space. Also, make two sets $\mathcal{Q}$ and $\overline{\mathcal{Q}}$ where $\mathcal{Q}$ contains all newly computed VTT configurations or non-visited VTT configurations in the grid space and $\overline{\mathcal{Q}}$ contains all visited VTT configurations in the grid

Algorithm 6: Graph Search Algorithm

Input: Initial VTT configuration $G_i = (V_i, E_i)$, edge set $\widehat{E}$, the intersection node $\hat{v}$, and the goal state of $e\,[\hat{v}]\ \ \forall e \in \widehat{E}$

Output: Tree of VTT configuration graphs p

```
1  Compute the nonuniform grid space;
2  Compute 𝔾_i = (𝕍_i, 𝔼_i) and 𝔾_g = (𝕍_g, 𝔼_g) in the grid space and the corresponding
     edge set 𝔼̂ with ∀e ∈ 𝔼̂ intersecting at node v̂;
3  Q ← {𝔾_i, 𝔾_g};
4  Q̄ ← ∅;
5  g(𝔾_i) ← 0, g(𝔾_g) ← ∞;
6  while 𝔾_g = (𝕍_g, 𝔼_g) ∈ Q do
7      Ḡ ← arg min_{𝔾∈Q} r(𝔾) = arg min_{𝔾∈Q} (g(𝔾) + h(𝔾));
8      Q ← Q \ {Ḡ};
9      Q̄ ← Q̄ ∪ {Ḡ};
10     A = ComputeActions(𝔼̂);
11     foreach a ∈ A do
12         𝔾 ← F(Ḡ, a);
13         if 𝔾 ∉ Q̄ then
14             if 𝔾 ∈ Q then
15                 if g(Ḡ) + c(a) < g(𝔾) then
16                     g(𝔾) ← g(Ḡ) + c(a);
17                     p(𝔾) ← Ḡ
18                 end
19             else
20                 Q ← Q ∪ {𝔾};
21                 g(𝔾) ← g(Ḡ) + c(a);
22                 p(𝔾) ← Ḡ
23             end
24         end
25     end
26 end
```

space. The size of these two sets will change as the algorithm explores the configuration space. It is not feasible to compute all possible VTT configurations as the number grows exponentially in the number of members. At the beginning, only the initial VTT configuration and the goal VTT configuration are in set $\mathcal{Q}$ and the algorithm starts with the initial configuration. The value $g(\mathbb{G})$ is the cost of the path from $\mathbb{G}_i$ to a VTT configuration $\mathbb{G}$, so $g(\mathbb{G}_i) = 0$ and $g(\mathbb{G}_g) = \infty$ at the beginning.

Line 7 — 10: Every iteration always starts with the VTT configuration with the smallest cost $r(\mathbb{G})$ in set $\mathcal{Q}$ and $r(\mathbb{G}) = g(\mathbb{G})+h(\mathbb{G})$ where $h(\mathbb{G})$ is a heuristic function of configuration $\mathbb{G}$ that estimates the cost of the cheapest path from $\mathbb{G}$ to $\mathbb{G}_g$. The heuristic value can be related to the distance from the current location to the goal location of every moving node. In the beginning, the initial VTT configuration has the smallest cost. Then update set $\mathcal{Q}$ and $\overline{\mathcal{Q}}$. The connection information among VTT configurations are not fully known, thus it is necessary to try all possible actions for these moving members to create connections. Recall that only one node is active for each reconfiguration step, so the total number of all executable reconfiguration actions is not large and the worst case is $C^3_{|\widehat{\mathbb{E}}|}$ (the total number of all combinations of all elements in $\widehat{\mathbb{E}}$ taken three at a time) for edge set $\widehat{\mathbb{E}}$.

Line 11 — 25: For every executable action $a \in A$, a new connection with another VTT configuration can be obtained by the transition model. If this VTT configuration has been visited, then this connection is not a new connection. If not, then there are two cases: this configuration is not a newly found configuration which means there is already a connection between this configuration and another VTT configuration, or this configuration is a new one that has no connection before. For the first case, check whether its value needs to be updated. $c(a)$ is the cost of the current action a. It is reasonable to set the topology reconfiguration action cost a relatively higher value because, for modular robots, docking and undocking are usually difficult. If its value is updated, then its parent $p(\mathbb{G})$ is also updated accordingly. For the second case, initialize the value and the parent of this new VTT configuration, and update set $\mathcal{Q}$.

Once $\mathbb{G}_g = (\mathbb{V}_g, \mathbb{E}_g)$ is visited, then the algorithm ends. With p, a tree with visited VTT

configurations as vertices, it is straightforward to find the path connecting the initial VTT configuration and the goal VTT configuration as well as the optimal reconfiguration action sequence. With this algorithm, there is no need to compute all possible VTT configurations, which is usually very time-consuming, and the tree is built as exploring VTT configurations.

11.3 Test Scenarios

The algorithm is implemented and tested with some tasks. Two reconfiguration examples are presented. The initial VTT configuration $G = (V, E)$ is shown in Figure 11.6. The location of each node is in the following:

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

In order to explore and search the graph efficiently, a suitable model of action cost and a good heuristic function are necessary. There are three different atomic actions: `Move`, `Split`, and `Merge`. The action cost model in the experiment is that the cost of `Move` action is the

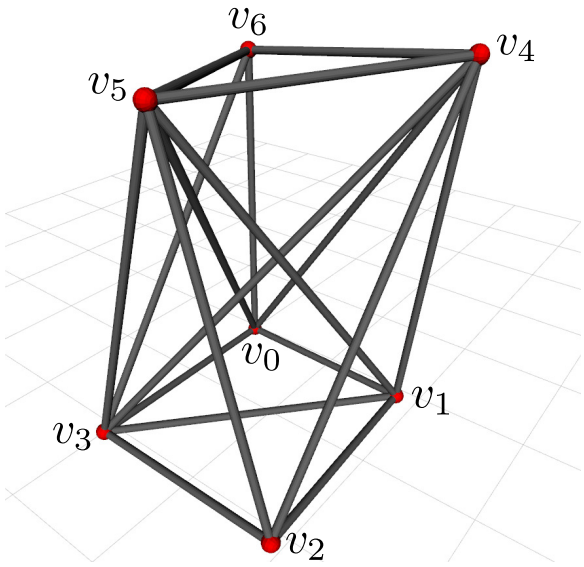


Figure 11.6: The initial VTT.

moving distance and the cost of `Split` or `Merge` is 1. The heuristic function is

$$h(\mathbb{G}) = \begin{cases} \sum_{v \in \widehat{V}} \| v^g\,[\texttt{Pos}] - v\,[\texttt{Pos}]\,\| + 1, & |\widehat{V}| > 1 \\ \| v^g\,[\texttt{Pos}] - v\,[\texttt{Pos}]\,\|, & |\widehat{V}| = 1 \end{cases}$$

in which $v^g\,[\texttt{Pos}]$ and $v\,[\texttt{Pos}]$ are the goal location and the current location of node v respectively, and $|\widehat{V}|$ is the size of set $\widehat{V}$. Recall that $\widehat{V} \subset V$ contains all the current nodes intersected by edge modules in $\widehat{E}$ and if there are more than one node in set $\widehat{V}$, then there must be at least one `Merge` action afterward, so the heuristic value is the sum of Euclidean distance for every node from its current location to its goal location plus one. Due to the collision avoidance constraints, this heuristic function $h(\mathbb{G})$ must be less than or equal to the cost of moving from $\mathbb{G}$ to $\mathbb{G}_g$, so the algorithm is guaranteed to find the optimal action sequence (or shortest path).

11.3.1 Scenario 1

The first reconfiguration task is shown in Figure 11.7. The task is to move edge module set $\widehat{E} = \{(v_0, v_5), (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}$ and they all intersect at node v_5. $\forall e \in \widehat{E}$, move $e\,[v_5]$ from its current position $[1.95, 0.9, 3]^\intercal$ to $[1, 0.9, 3]^\intercal$ which is very close to its original position. However, this motion cannot be executed by moving node v_5 to the

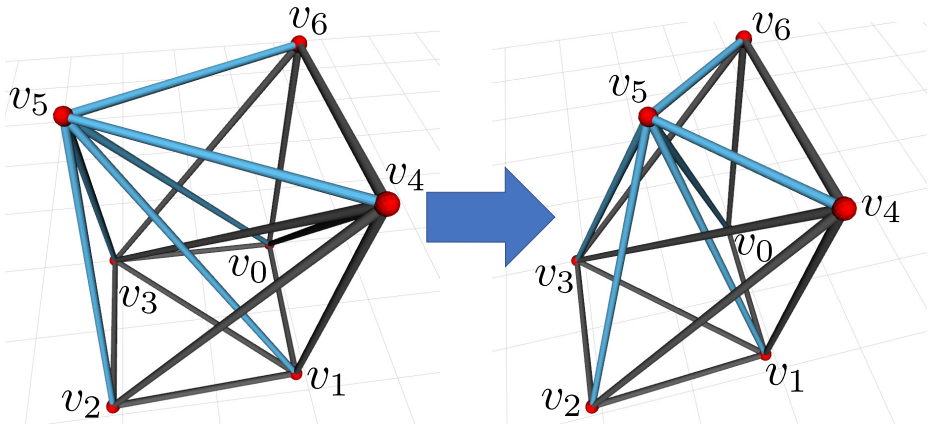


Figure 11.7: $\forall e \in \{(v_0, v_5), (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}$, move $e\,[v_5]$ from the initial location to the goal location. With only geometry reconfiguration actions, edge module (v_1, v_5) will collide with edge module (v_3, v_4).

goal position directly because edge module (v_1, v_5) will collide with edge module (v_3, v_4). Topology reconfiguration actions are needed for this task.

The conversion between the grid space and Cartesian space is computed first with $\delta = 0.2$ and the corresponding node locations in the grid space are

$$\begin{aligned} &\mathbb{v}_0\,[\texttt{Pos}] = [0,0,0]^\intercal \quad \mathbb{v}_1\,[\texttt{Pos}] = [0,4,0]^\intercal \quad \mathbb{v}_2\,[\texttt{Pos}] = [4,4,0]^\intercal \\ &\mathbb{v}_3\,[\texttt{Pos}] = [4,0,0]^\intercal \quad \mathbb{v}_4\,[\texttt{Pos}] = [0,4,2]^\intercal \quad \mathbb{v}_5\,[\texttt{Pos}] = [4,2,2]^\intercal \\ &\mathbb{v}_6\,[\texttt{Pos}] = [0,0,2]^\intercal \end{aligned}$$

and $\forall e \in E$, the goal location of $e\,[v_5]$ in the grid space is $[2,2,2]^\intercal$. The lower bound of the grid space is $[0,0,0]^\intercal$ and the upper bound of the grid space is $[4,4,2]^\intercal$. In total, there are 75 grid locations. The size of $\widehat{E}$ is six and the number of all possible arrangements for them in the grid space is $75 \times 75 \times (C_6^3 C_3^3/2) = 56250$ and the size of the action space is $56250 \times 52 = 2925000$. However, the algorithm only explores 3771 VTT configurations and the sequence of optimal actions can be found efficiently.

The solution by the motion planner is illustrated in Figure 11.8. First move node v_5

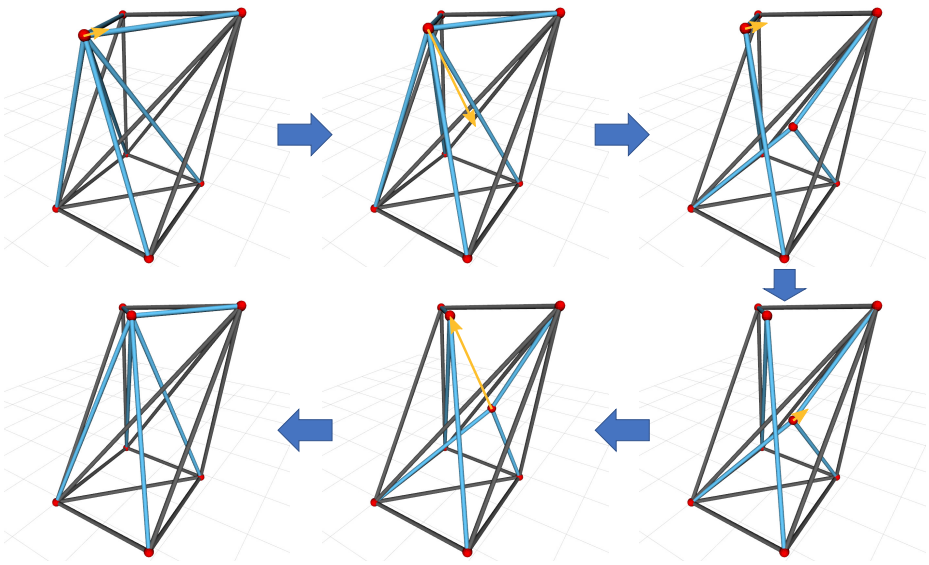


Figure 11.8: The sequence to change the states of all involved edge modules is shown and the motion directions are denoted as →. First move the intersection node in a small step, and then split it into two separate nodes and move one of the node downward in a large step. Move both nodes closer and finally merge them in the goal location.

along $-x$-axis in the grid space to a closer location ($[1.475, 0.9, 3.0]^\intercal$) in Cartesian space, then split the node into a pair of nodes so that two groups of edge modules can be controlled separately and move one of them to the location $[1.0, 1.35, 1.5]^\intercal$ which is below the obstacle edge module. This action moves the node with a longer distance than the previous one because the space occupancy is sparser along z-axis than that along x-axis. Next move two newly generated nodes along $-x$-axis to $[1.0, 0.9, 3.0]^\intercal$ and $[0.5, 1.35, 1.5]^\intercal$ respectively so that they can be merged in the goal location in the last step.

11.3.2 Scenario 2

The second reconfiguration task is shown in Figure 11.9. The task is also to move edge module set $\widehat{E} = \{(v_0, v_5), (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}$ intersecting at node v_5, and $\forall e \in \widehat{E}$, change $e\,[v_5]$ from its current position $[1.95, 0.9, 3]^\intercal$ to $[1, 1.2, 0.9]^\intercal$. The initial position is almost on the boundary of the truss but the goal position is almost in the center of the structure. Still with only geometry reconfiguration actions, it is impossible to finish this motion task because edge module (v_3, v_4) is between two edge modules (v_0, v_5) and (v_1, v_5).

Set $\delta = 0.2$ to compute the conversion between the grid space and Cartesian space and

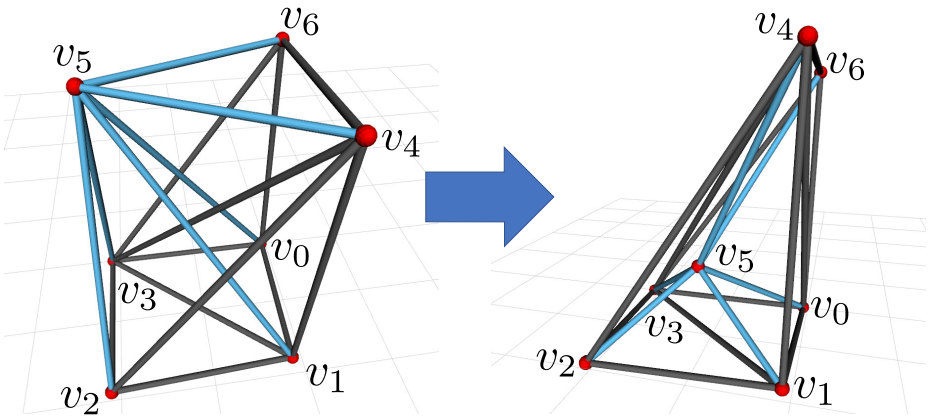


Figure 11.9: $\forall e \in \{(v_0, v_5), (v_1, v_5), (v_2, v_5), (v_3, v_5), (v_4, v_5), (v_6, v_5)\}$, move $e\,[v_5]$ from the initial location to the goal location. With only geometry reconfiguration action, there is no way for edge module (v_0, v_5) and (v_1, v_5) to traverse edge module (v_3, v_4).

the corresponding node locations in the grid space are

$$\begin{aligned}
&\mathbb{v}_0\,[\texttt{Pos}] = [0,0,0]^\intercal \quad \mathbb{v}_1\,[\texttt{Pos}] = [0,6,0]^\intercal \quad \mathbb{v}_2\,[\texttt{Pos}] = [4,6,0]^\intercal \\
&\mathbb{v}_3\,[\texttt{Pos}] = [4,0,0]^\intercal \quad \mathbb{v}_4\,[\texttt{Pos}] = [0,6,4]^\intercal \quad \mathbb{v}_5\,[\texttt{Pos}] = [4,2,4]^\intercal \\
&\mathbb{v}_6\,[\texttt{Pos}] = [0,0,4]^\intercal
\end{aligned}$$

and $\forall e \in E$, the goal location of $e\,[v_5]$ in the grid space is $[2,4,2]^\intercal$. The lower bound of the grid space is still $[0,0,0]^\intercal$ but the upper bound of the grid space is $[4,6,4]^\intercal$. This shows that the way to decompose the space depends on the motion task. For this motion task, finer cells along y-axis and z-axis are generated with the same parameter δ. With higher resolution, there are more grid locations, in this example 175. The number of all possible arrangements for this set of edge modules is $175 \times 175 \times (C_6^3 C_3^3/2) = 306250$ and the size of action space is $306250 \times 52 = 15925000$ which is much larger. However, the algorithm only needs to explore 8146 VTT configurations.

The solution is illustrated in Figure 11.10. Similarly, the conversion between the

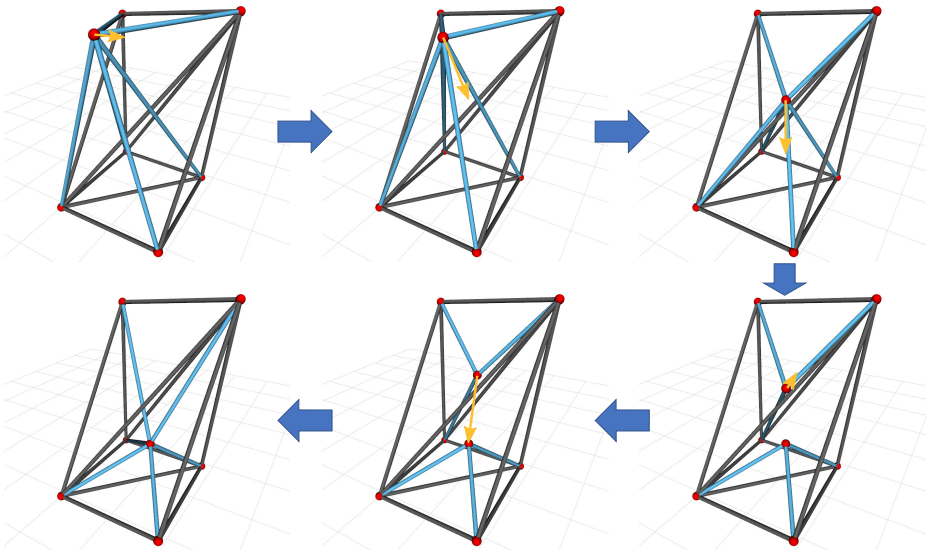


Figure 11.10: The sequence to change the states of all involved edge modules is shown and the motion directions are denoted as $\rightarrow$. First move the intersection node to the center by two `Move` actions, and then split them into two separate nodes and move them in different directions to go around the obstacle member. In the end, merge these two nodes into a single one.

grid space and Cartesian space is computed accordingly. Firstly, move node v_5 to $[1.525, 1.05, 2.9]^\intercal$ and then to $[1.1, 1.2, 1.9]^\intercal$ with a larger step because this part of space is very sparse, and then split this node into a pair of nodes so that six edge modules are separated into two groups to go across the obstacle. Move one of them to the goal location and move the other one to $[0.55, 1.05, 1.9]^\intercal$ to bypass the obstacle, and then merge them at the goal location.

From these two examples, it is observed that there is no uniform motion stepsize because the space occupancy is different. This makes the reconfiguration planning algorithm efficient to explore VTT configurations using the proposed graph search algorithm.

11.4 Conclusions

This chapter shows the advantage of topology reconfiguration and presents a new reconfiguration planning algorithm to perform versatile motions inspired by the DNA replication process. The reconfiguration planning problem is challenging due to its large topology configuration space. Being a truss presents a variety of constraints to the inherent parallel robot planning problems. A new method is presented to discretize the space called a grid space, considering the overall shape of the robot and the motion task which results in a more efficient decomposition of the workspace. A robot is then modeled in both Cartesian space and this non-uniform grid space. Based on this new model, discrete reconfiguration actions can be applied, an efficient collision checking method is developed, and then a transition model is derived. The VTT configuration space can be explored efficiently using a graph search algorithm with a heuristic function to do motion planning combined with topology reconfiguration.

Chapter 12

Node Configuration Space

This chapter presents the fundamental analysis of the configuration space for VTT nodes. This space is complicated due to the complexity of the truss structure. A fast algorithm to compute this space is introduced which can be used for motion planning. This chapter excerpts heavily from [76]. Credit is due to Sencheng Yu who contributed significantly to this work.

A VTT is usually composed of many members spanning the workspace in a very flexible manner. Motion planning and avoiding self-collision are difficult as these systems usually have dozens of DOFs with complex intersecting parallel actuation. There are two different types of shape-changing actions for a VTT: geometry reconfiguration and topology reconfiguration. This chapter focuses on geometry reconfiguration actions. A fast and complete method to compute the configuration space of a node in a VTT is presented. Based on this, a cell decomposition approach can be applied to achieve efficient motion planning for a single node, and a shape-morphing task can also be solved quickly by moving one node at a time.

12.1 Introduction

A VTT is inherently a parallel robot. Different from open chain kinematic structures, it is much easier to solve inverse kinematics problems than to solve forward kinematics for parallel robots. Thus, rather than planning the motion of a VTT from the member lengths,

the motion planning problem is solved by planning node motions and the required member lengths are determined afterward. Geometry reconfiguration is to change the lengths of members that can be achieved by moving the corresponding nodes, and topology reconfiguration is to change to connectivity among members that happens at nodes. Hence, planning the motion of nodes is the fundamental to reconfigure a VTT.

For a VTT, the node assignment defines its topology and the lengths of all members define its shape. In order to be a reconfigurable VTT, there are some constraints on the arrangement of members, including that a VTT has to be a rigid structure to maintain its shape and be statically determinant. In general, a node in a VTT needs at least three members attached to ensure its controllability. In addition, topology-reconfigurable VTT systems require at least eighteen members to reconfigure which means they have at least eighteen actuated DOFs. Thus, motion planning for these systems involves at least eighteen and typically more than twenty-one dimensions. In addition, members forming VTT structures typically span the workspace in a very non-uniform manner as the truss configuration creates a complicated configuration space. These facts and constraints make the analysis and planning for VTTs difficult.

This work focuses on the node configuration space in an arbitrary VTT. A node has three DOFs but its obstacle region and free space are complicated. This chapter presents a fast and complete approach to compute the obstacle region and the free space of a node, and shows that multiple nodes in a VTT are usually strongly coupled in the space, namely moving one node can significantly affect the configuration space of other nodes. It is also shown that the free space of a node is usually not convex and can be further decomposed into multiple convex polyhedrons in which the corresponding node can move freely without considering collision. Then a simple graph search algorithm can be used to plan a path for this node efficiently. This is beneficial to changing the overall shape of a VTT. The constraint that motions of multiple nodes are strongly coupled can be relaxed by moving one node at a time, which may lead to longer execution time compared to having all DOFs move at the same time. But the advantage is that this very high-dimensional planning

problem is converted into multiple 3D problems which are feasible to be solved by graph search algorithm rather than using high DOF search algorithms such as Rapidly-exploring Random Trees (RRT). In this way, the multi-node planning problem can be significantly simplified. This strategy is demonstrated effective by several tasks.

12.2 Single Node Configuration Space

As introduced in Section 10.2, the configuration of a VTT is modeled as a graph $G = (V, E)$. The `Pos` property of a vertex $v \in V$ defines the Cartesian coordinates of the corresponding node denoted as $q^v = [v_x, v_y, v_z]$ which is the configuration of this node, and the *configuration space* of node v denoted as $\mathcal{C}^v$ is $\mathbb{R}^3$. Node v is fully controlled by all attached edge modules $E^v \subseteq E$, and E^v is a parallel robot actuated by edge modules. When moving node v, collision could happen between some pair of members in E^v or between $e \in E^v$ and $\bar{e} \in E \setminus E^v$.

12.2.1 Obstacle Region and Free Space

When moving a single node $v \in V$ in $G = (V, E)$, the state of every $e \in E^v$, denoted as $\mathcal{A}^v(q^v)$, can only be changed by altering q^v. For this node v, the *obstacle region* $\mathcal{C}^v_{\text{obs}} \subseteq \mathcal{C}^v = \mathbb{R}^3$ is defined as

$$\mathcal{C}^v_{\text{obs}} = \{q^v \in \mathbb{R}^3 | \mathcal{A}^v(q^v) \cap \mathcal{O}^v \neq \emptyset\} \tag{12.1}$$

in which $\mathcal{O}^v$ is the obstacle for E^v. Then the *free space* of node v is just the leftover configurations denoted as

$$\mathcal{C}^v_{\text{free}} = \mathbb{R}^3 \setminus \mathcal{C}^v_{\text{obs}} \tag{12.2}$$

Theorem 2. *For a given node v in $G = (V, E)$, $\mathcal{C}^v_{\text{obs}}$ can be fully defined by the states of $\forall e \in E \setminus E^v$.*

Proof. Suppose the node to move is $v_i \in V$ then there are three cases where a moving member $e = (v_i, v_j) \in E^{v_i}$ could collide with other members:

1. Member $(v_i, v_j) \in E^{v_i}$ may collide with $(v_m, v_n) \in E \setminus E^{v_i}$ as shown in Figure 12.1a, and the obstacle region generated by (v_m, v_n) can be defined by an unbounded polygon formed by member (v_m, v_n) and node v_j (the blue polygon in Figure 12.1a). Since one

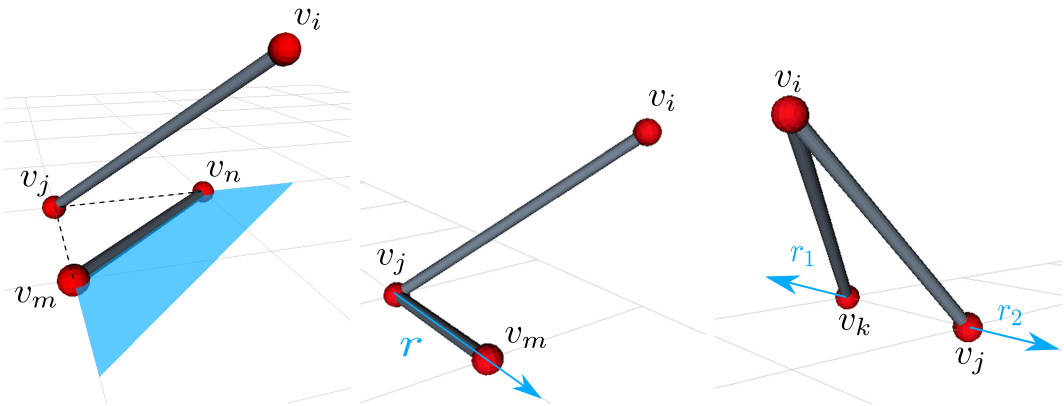


Figure 12.1: (a) (v_i, v_j) collides with (v_m, v_n) when v_i is on the blue polygon. (b) (v_i, v_j) collides with (v_m, v_j) when v_i is on the blue ray r. (c) (v_i, v_j) collides with (v_i, v_k) when v_i is on the blue ray r_1 or r_2.

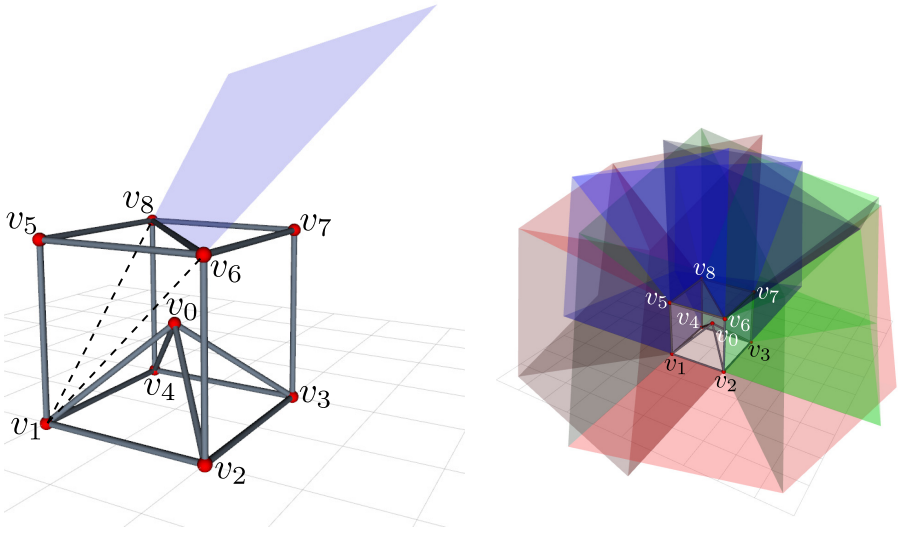
node is connected with at least three nodes in a VTT, it can be concluded that $\exists v_l \in V$ such that $(v_j, v_l) \in E \setminus E^v$. Therefore, the obstacle region in this case is defined by the states of members in $E \setminus E^{v_i}$;

2. Member $(v_i, v_j) \in E^{v_i}$ collides with member $(v_j, v_m) \in E \setminus E^{v_i}$ if and only if the trajectory of v_i intersects with r, a ray starting at q^{v_j} and pointing in the direction of $q^{v_m} - q^{v_j}$ as shown in Figure 12.1b. Then this ray-shaped obstacle region is defined by the state of the member $(v_j, v_m) \in E \setminus E^{v_i}$;

3. Member $(v_i, v_j) \in E^{v_i}$ collides with $(v_i, v_k) \in E^{v_i}$ if and only if the trajectory of v_i intersects with ray r_1 or r_2 where r_1 starts from q^{v_k} and points in the direction of $q^{v_k} - q^{v_j}$ and r_2 starts from q^{v_j} and extends in the opposite direction, as shown in Figure 12.1c. Since $\exists v_m \in V$ such that $(v_m, v_k) \in E \setminus E^{v_i}$, r_1 is contained in the obstacle region caused by (v_m, v_k) and (v_i, v_j). Hence, r_1 is fully defined by the states of members in $E \setminus E^{v_i}$. And similar approach can be applied to r_2. Therefore, these obstacle regions are fully defined by the states of members in $E \setminus E^{v_i}$.

□

With Theorem 2, $\mathcal{C}^v_{\text{obs}}$ is constructed by all polygons generated from $E \setminus E^v$. For a simple



(a) (b)

Figure 12.2: (a) Given node v_0, one of its neighbors v_1 and a member (v_6, v_8) can define the blue polygon. This polygon is part of $\mathcal{C}_{\text{obs}}^{v_0}$. (b) The obstacle region $\mathcal{C}_{\text{obs}}^{v_0}$ for node v_0.

VTT shown in Figure 12.2a, the obstacle region $\mathcal{C}_{\text{obs}}^{v_0}$ for node v_0 is shown in Figure 12.2b. The number of these polygons is of $\mathcal{O}(|E|^2)$. From Theorem 2, it can be concluded that the motions of multiple nodes are strongly coupled. In a VTT $G = (V, E)$, for any pair of nodes v_i and v_j, $\mathcal{C}_{\text{obs}}^{v_i}$ and $\mathcal{C}_{\text{obs}}^{v_j}$ are fully defined by the edge module set $E \backslash E^{v_i}$ and $E \backslash E^{v_j}$ respectively. If v_i and v_j are not adjacent, then $E^{v_i} \subseteq E \setminus E^{v_j}$ and $E^{v_j} \subseteq E \setminus E^{v_i}$. Otherwise, v_i and v_j share one edge module (v_i, v_j), and $E^{v_i} \setminus \{(v_i, v_j)\} \subseteq E \setminus E^{v_j}$, $E^{v_j} \setminus \{(v_i, v_j)\} \subseteq E \setminus E^{v_i}$. In both cases, $\mathcal{C}_{\text{obs}}^{v_i}$ is partially determined by E^{v_j} and $\mathcal{C}_{\text{obs}}^{v_j}$ is partially determined by E^{v_i}. Hence, the motions of v_i and v_j are strongly coupled.

12.2.2 Free Space Boundary

All the polygons defined by $E \setminus E^v$ form the obstacle region $\mathcal{C}_{\text{obs}}^v$ for node v. If v is on any polygon, collision must happen among members. However, node v may not be able to move to any location inside $\mathcal{C}_{\text{free}}^v$ from its initial configuration q_i^v because $\mathcal{C}_{\text{free}}^v$ may be not fully connected and can be separated by $\mathcal{C}_{\text{obs}}^v$. For example, $\mathcal{C}_{\text{obs}}^v$ is composed of six polygons shown in Figure 12.3, but node v is only able to move freely inside the space enclosed by polygon P_1, P_2, P_3, P_4, and P_5 starting from its current location. P_6 and parts of polygons

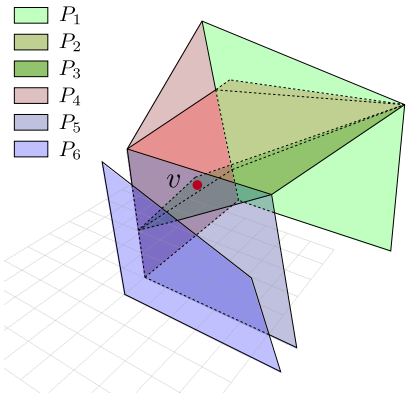


Figure 12.3: v is enclosed by polygon P_1, P_2, P_3, P_4, and P_5, and polygon P_6 is outside the enclosure that has nothing to do with $\mathcal{C}^v_{\text{free}}(q^v)$.

P_1 — P_5 are outside the enclosed space. This enclosed space is the subset of $\mathcal{C}^v_{\text{free}}$ denoted as $\mathcal{C}^v_{\text{free}}(q^v)$ where q^v is the initial location of node v. It is necessary to find the boundary of $\mathcal{C}^v_{\text{free}}(q^v)$ in order to do motion planning for v.

Polygon Intersection

Given a polygon P_i with α sides denoted as s_{ij} where $j \in [1, \alpha]$, if P_i is connected with a set of polygons denoted as $\mathcal{N}_{ij}$ through s_{ij}, then $\mathcal{N}_{ij}$ is called the neighbor set of P_i at side s_{ij}. When checking intersections, there are four possible cases in terms of the intersection set of two polygons. For the cases that the intersection set is empty and the intersection set only contains a single point, no further computation is needed.

Another case is that the intersection of two polygons is a line segment or a ray. In this case, both polygons (Figure 12.4c) would be cut into two pieces by the intersection line, and there are two special situations that zero (Figure 12.4a) or only one (Figure 12.4b) polygon is cut. Let the two intersected polygons be P_1 and P_2, and the resulted separated polygons are $\{P'_1, P''_1\}$ and $\{P'_2, P''_2\}$, respectively. Assume that P_1 has α_1 sides denoted as $S_1 = \{s_{1j} \mid j = 1, 2, \cdots, \alpha_1\}$. Similarly, S'_1 for P'_1 is also defined. Then, for any side $s'_{1j} \in S'_1$ such that $s'_{1j} = s_{1m} \in S_1$, namely this edge is inherited from P_1, the neighbor set $\mathcal{N}'_{1j}$ of P'_1 is equal to the neighbor set $\mathcal{N}_{1m}$ of P_1. This is called the *inheritance process*. And for any side $s'_{1j} \in S'_1$ such that $\exists s_{1m} \in S_1$, s'_{1j} is a part of s_{1m}, namely s_{1m} in P_1 is cut and P'_1 only

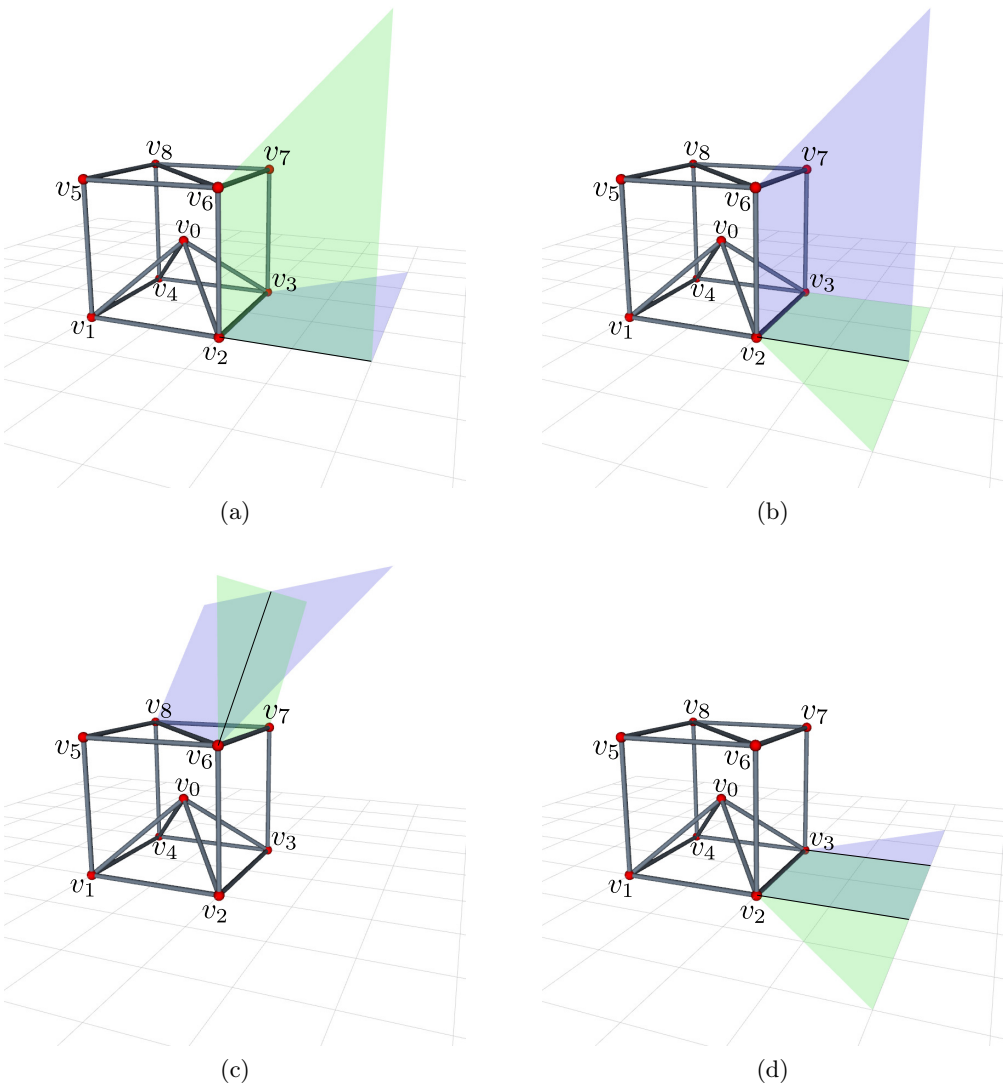


Figure 12.4: (a) The intersection between the polygon generated by node v_1 with member (v_2, v_3) and the polygon by node v_1 with member (v_2, v_6) is a ray and no polygon is cut. (b) The intersection between the polygon generated by node v_1 with member (v_2, v_6) and the polygon by node v_4 with member (v_2, v_3) is a ray and only one polygon is cut into two pieces. (c) The intersection between the polygon generated by node v_1 with member (v_6, v_8) and the polygon by node v_2 with member (v_6, v_7) is a ray and both polygons are cut into two pieces. (b) The intersection between the polygon generated by node v_1 with member (v_2, v_3) and the polygon by node v_4 with member (v_2, v_3) is also a polygon which is the region between the two parallel black lines.

inherits part of s_{1m}, check every polygon in $\mathcal{N}_{1m}$ and all those polygons that are connected with P_1' compose $\mathcal{N}_{1j}'$. This is called the *recheck process*. Finally, for the side $s_{1j}' \in S_1'$ which coincides with the cutting line, let $\mathcal{N}_{1j}' = \{P_1'', P_2', P_2''\}$. This is called the *adding process*. The same approaches will also be applied to P_1'', P_2', and P_2'' to compute their neighbor sets. In addition, there are special situations for the third case. If the intersection lies on a side of one polygon, e.g. P_1, and inside another, e.g. P_2, then only P_2 is cut and P_1 only requires the adding process. If the intersection lies on a side of both polygons, only the adding process is applied to both.

The last case occurs when two polygons lie on the same plane and their intersection is also a polygon (e.g. Figure 12.4d). For this case, one of the polygons, e.g. P_1, remains unchanged and the other one, e.g. P_2, can be divided into a set of convex polygons, $\mathcal{R}_2$, so that no polygon in $\mathcal{R}_2$ overlaps P_1. To do this, starting from any $s_{1m} \in S_1$, cut P_2 into P_2' and P_2'' using the line on which s_{1m} lies and apply the above three processes to compute the neighbors of P_2' and P_2''. Only one of the resulted two polygons overlaps P_1, e.g. P_2''. P_2' is put into $\mathcal{R}_2$ and, if it is a neighbor of P_1, add it to the neighbor sets of P_1 and add P_1 to the neighbor set of P_2'. Then continue to apply this operation to cut P_2'' using another side $s_{1n} \in S_1 \setminus \{s_{1m}\}$. Repeat this process until there is a newly generated polygon that is fully contained in P_1 and remove it from the neighbor sets of polygons in $\mathcal{R}_2$.

The motion of a VTT node is kept inside the boundary of the workspace. The above operations are also applied to the boundary of the workspace. And after this process, a set of polygons $\mathcal{P}_{\text{obs}}$ forming the whole $\mathcal{C}_{\text{obs}}^v$ and the boundary of the workspace is obtained. Since each pair of polygons is checked, this process has quadratic time complexity in the number of polygons and therefore $\mathcal{O}(|E|^4)$.

Boundary Search

For a polygon P_i with α_i sides and all of its neighbor sets $\{\mathcal{N}_{ij} \mid j = 1, 2, \cdots, \alpha_i\}$, it is fully connected if and only if $\forall j \in [1, \alpha_i]$ such that $\mathcal{N}_{ij} \neq \emptyset$. The resulting polygons after *Polygon Intersection* are separated into two sets: the set of all fully connected polygons $\mathcal{F}$ and the set of all not fully connected polygons $\mathcal{U}$. Suppose the workspace is closed, it can

be seen that the boundary of $\mathcal{C}^v_{\text{free}}(q^v)$, a polyhedron, can only be constructed from fully connected polygons. For any polygon P_i, there are two normal vectors in different directions perpendicular to the plane on which it lies. The one pointing inside $\mathcal{C}^v_{\text{free}}(q^v)$ is called the inner direction vector denoted as n_i. The distance between node v and a polygon is the minimum distance between v and any points on this polygon. The polygon P_{s} with the shortest distance to v (if multiple, choose the one with the maximum distance between v and the plane on which the polygon lies) must be a boundary polygon and n_{s} can be found regardless of whether $\mathcal{C}^v_{\text{free}}(q^v)$ is convex or not. For node v_0 in Figure 12.2, P_{s} is shown in Figure 12.5. One set of its neighbor polygons contains two obstacle polygons sharing an edge with P_{s} in which the green polygon is the innermost one along vector n_{s}.

Then, Algorithm 7 is used to search for a set of boundary polygons $\mathcal{P}_b$ of $\mathcal{C}^v_{\text{free}}(q^v)$, and the result of $\mathcal{C}^{v_0}_{\text{free}}(q^{v_0})$ for node v_0 in Figure 12.2 is shown in Figure 12.6. Since every polygon is checked at most once, this process takes $\mathcal{O}(|E|^2)$ time. After this process, the set $\mathcal{U}$ has to be refined by removing polygons that are outside the obtained boundary.

Algorithm 7: Boundary Search Algorithm

Input: The current node position q^v, the set of obstacle polygons $\mathcal{P}_{\text{obs}}$
Output: The set of boundary polygons $\mathcal{P}_b$ of $\mathcal{C}^v_{\text{free}}(q^v)$

```
1  P_s ← polygon closest to node v;
2  P_b ← ∅;
3  Q_P ← ∅;
4  Q_P.enqueue(P_s);
5  while Q_P ≠ ∅ do
6  |   P_i ← Q_P.dequeue();
7  |   P_b ← P_b ∪ {P_i};
8  |   foreach s_ij ∈ S_i do
9  |   |   P̄_ij ← the innermost polygon in N_ij along n_i;
10 |   |   Compute inner direction vector n̄_ij of P̄_ij;
11 |   |   if P̄_ij ∉ P_b ∧ P̄_ij ∉ Q_P then
12 |   |   |   Q_P.enqueue(P̄_ij);
13 |   |   end
14 |   end
15 end
```

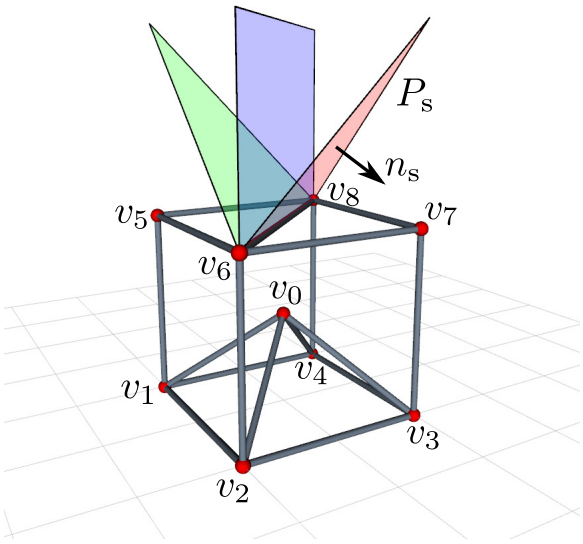


Figure 12.5: For node v_0, P_s is the red polygon. One set of its neighbor polygons contains two obstacle polygons. The green polygon is the innermost one along vector n_s that is added to the boundary of $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$.

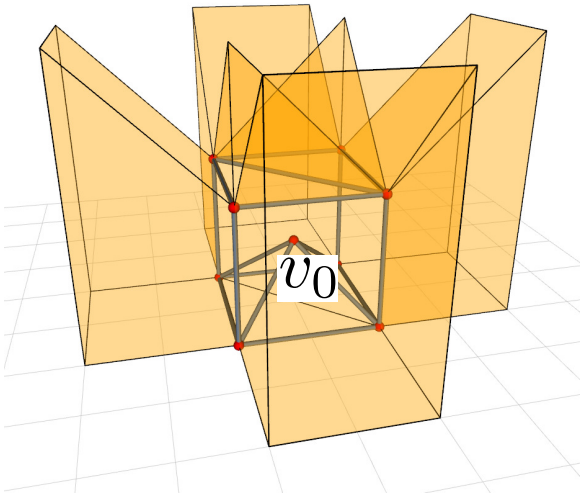
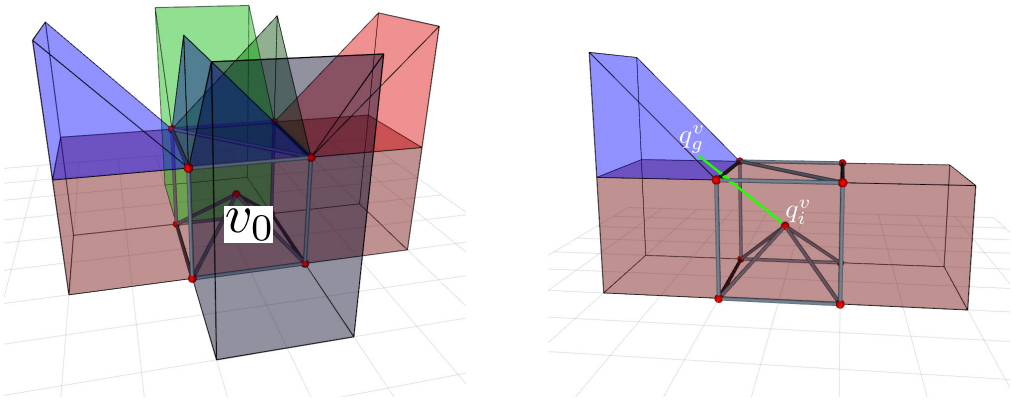


Figure 12.6: $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$ is bounded by polygons.

12.3 Single Node Path Planning

12.3.1 Cell Decomposition

The enclosed free space $\mathcal{C}^v_{\text{free}}(q^v)$ is not necessarily convex which can be further decomposed into several convex polyhedrons. The problem to decompose a non-convex polyhedron into a minimum number of convex pieces is known to be NP-hard [19]. All polygons in $\mathcal{P}_b$ are passed into a function of the Computational Geometry Algorithms Library (CGAL) to decompose the space into $\mathcal{O}(r^2)$ where r is the number of edges that have two adjacent facets that span an angle of more than 180° with respect to the interior of the polyhedron [140]. The cell decomposition result of the space in Figure 12.6 is shown in Figure 12.7a. Each convex polyhedron denoted as c is a cell in which the node can move freely without considering collision. However, some cells may intersect with some polygon P in $\mathcal{U}$ (the set of not fully connected polygons) and these cells need to be further separated into two cells by the plane on which P lies (this is because CGAL ignores this case when doing convex decomposition). This checking process takes also $\mathcal{O}(|E|^4)$ time, so the total time complexity of the boundary construction for $\mathcal{C}^v_{\text{free}}(q^v)$ is $\mathcal{O}(|E|^4)$.



(a) (b)

Figure 12.7: (a) $\mathcal{C}^{v_0}_{\text{free}}(q^{v_0})$ is decomposed into multiple convex polyhedrons. (b) The path planned for v_0 to move from its initial location q_i^v to the goal location q_g^v is shown as the green path, and v_0 only needs to traverse two convex polyhedrons.

12.3.2 Path Planning

With all decomposed cells, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be constructed where $\mathcal{V}$ is the set of all decomposed cells and $\mathcal{E}$ is the set of edges with each edge representing the connection of two adjacent cells. The cost of an edge is the distance of the path the node has to traverse from one cell to another. Two cells are adjacent if and only if they are not separated by polygons in $\mathcal{U}$. When traversing from one cell c' to its adjacent cell c'' in $\mathcal{G}$, the path is a straight line connecting the centers of two adjacent cells if it is inside $\mathcal{C}^v_{\text{free}}(q^v)$. Otherwise, the path is from the center of cell c' to the center of the intersection polygon between c' and c'', then to the center of c''. For node v, given its initial location q^v_i and a goal location q^v_g, find the cell c_i and c_g containing q^v_i and q^v_g respectively. Then a path from c_i to c_g in $\mathcal{G}$ can be found using Dijkstra Algorithm. For example, the planned path for v_0 in a simple VTT is shown in Figure 12.7b.

12.3.3 Completeness for Single Node Planning

It is claimed that this single-node planning algorithm is complete, i.e. it must compute a (continuous) *path*, $\tau : [0, 1] \to \mathcal{C}^v_{\text{free}}$, such that $\tau(0) = q^v_i$ and $\tau(1) = q^v_g$, or correctly report that such a path does not exist [66].

According to Theorem 2, the obstacle region $\mathcal{C}^v_{\text{obs}}$ of a node v is fully defined, as well as the free space $\mathcal{C}^v_{\text{free}} = \mathbb{R}^3 \setminus \mathcal{C}^v_{\text{obs}}$. Recall that for a configuration of a node q^v, $\mathcal{C}^v_{\text{free}}(q^v)$ is the enclosed space containing q^v and its boundary is formed by either $\mathcal{C}^v_{\text{obs}}$ or the workspace boundary, thus v cannot go outside $\mathcal{C}^v_{\text{free}}(q^v)$ from q^v, or it will traverse the obstacle region or the workspace boundary. Hence, if the goal location $q^v_g \notin \mathcal{C}^v_{\text{free}}(q^v)$, there is no feasible path. Otherwise, there must be a feasible path τ connecting q^v_g with q^v_i.

$\mathcal{C}^v_{\text{free}}(q^v)$ is a polyhedron that can be decomposed into T convex polyhedrons denoted as $\{c_t | t = 1, 2, \cdots, T\}$ by convex decomposition operation, and q^v_i and q^v_g must be in some cells. Let $q^v_i \in c_i$ and $q^v_g \in c_g$. Node v can move freely from q^v_1 to q^v_2 as long as $q^v_1 \in c_t$ and $q^v_2 \in c_t$ where $t = 1, 2, \cdots, T$. Then, for adjacent cells c_m and c_n, there must exist a feasible path $\tau_{mn} : [0, 1] \to \mathcal{C}^v_{\text{free}}(q^v) \subseteq \mathcal{C}^v_{\text{free}}$ such that $\tau_{mn}(0) = q^v_m$ and $\tau_{mn}(1) = q^v_n$ where $q^v_m \in c_m$

and $q_n^v \in c_n$ since node v can always move from q_m^v to the common boundary of c_m and c_n freely and then move to q_n^v freely. Recall that all of the convex polyhedron cells construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and Dijkstra Algorithm is complete to find the shortest path, then it is guaranteed to find a sequence of convex cells from c_i to c_g. Hence, the path $\tau : [0, 1] \rightarrow \mathcal{C}_{\text{free}}^v$ such that $\tau(0) = q_i^v$ and $\tau(1) = q_g^v$ is derived.

12.4 Shape Morphing Approach

With the ability to find a path of a single node in VTT, shape morphing can be achieved by a sequence of single-node motion. Assume there are n nodes v_1, v_2, $\cdots$, v_n that should be moved from $q_i^{v_1}$, $q_i^{v_2}$, $\cdots$, $q_i^{v_n}$ to $q_g^{v_1}$, $q_g^{v_2}$, $\cdots$, $q_g^{v_n}$ respectively to achieve a shape morphing task. First compute $\mathcal{C}_{\text{free}}^{v_1}(q_i^{v_1})$ and move v_1 to $q_g^{v_1}$, then compute $\mathcal{C}_{\text{free}}^{v_2}(q_i^{v_2})$. If $q_g^{v_2} \in \mathcal{C}_{\text{free}}^{v_2}(q_i^{v_2})$, move v_2 to $q_g^{v_2}$. Otherwise, try to move the next candidate. The process is repeated until all nodes are moved to their destinations or returns failure. Thus, the problem reduces to be finding a sequence of node motions. In the worst case, the approach has to try $n!$ times which is the permutation of the number of moving nodes. The efficiency of the single-node planning makes this approach applicable to the multi-node planning problem. The benefit of doing this is that it is not necessary to do the free space computation and cell decomposition frequently. However, the drawback is that it may not be able to find the solution for some extreme cases even there exists a feasible path.

12.5 Experiments

The algorithm is implemented in C++ with the Boost Graph Library (BGL) [139] and the Computational Geometry Algorithms Library (CGAL) [140]. CGAL is used to do convex decomposition of a non-convex polyhedron and BGL is used to do graph search. To demonstrate the algorithm, two experiments were conducted, both on a laptop computer (Intel Core i7-8750H CPU, 16GB RAM). The first experiment is about single-node planning and the second one is a multi-node planning test.

Table 12.1: Comparison between the proposed method and the RRT approach from [52].

Test Node	Proposed Method / Cell Decomposition (s)	RRT (s)
v_0	0.3031 / 0.2301	0.3050
v_1	0.3581 / 0.2740	1.0245
v_2	0.5145 / 0.4383	0.9092
v_3	0.1292 / 0.0552	2.2419
v_4	0.1013 / 0.0294	0.7646
v_5	0.0578 / 0.0327	0.8052
v_6	0.2796 / 0.2022	0.6283

12.5.1 Single-Node Experiment

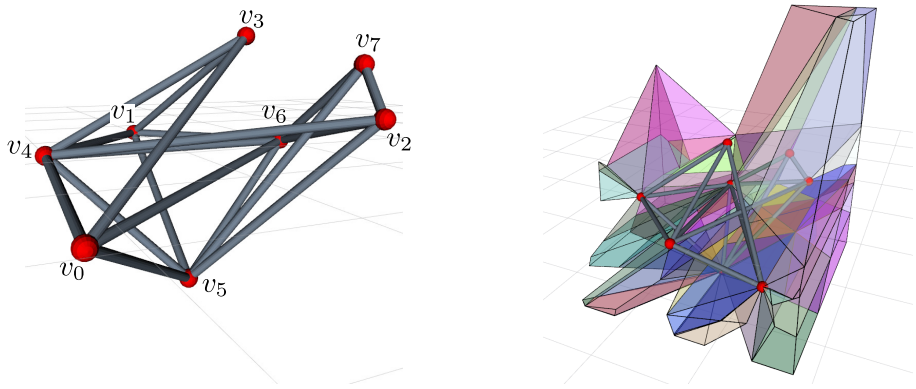
Several tests were used to evaluate the presented approach in this Chapter and do comparison with the retraction-based RRT from [52] which is previously used for VTT geometry motion planning. The VTT used in this test is shown in Figure 11.6. For each node v, a goal location $q_g^v \in \mathcal{C}_{\text{free}}^v(q_i^v)$ satisfying $\|q_g^v - q_i^v\| < 1$ is selected randomly. The results are shown in Table 12.1.

From this test result, the proposed approach is faster than the retraction-based RRT. In addition, one significant feature of VTTs is that they are able to achieve a large workspace. But RRT can take more than 40 s to search a path with four times larger workspace, while the proposed method is not affected by the workspace size. Also, most time in this approach is taken by CGAL for cell decomposition which can be improved by using a faster cell decomposition method but may end up with more convex cells generated.

A VTT shown in Fig 12.8a, constructed from 18 members, is used for a difficult task test. The initial positions of all nodes are listed in the following:

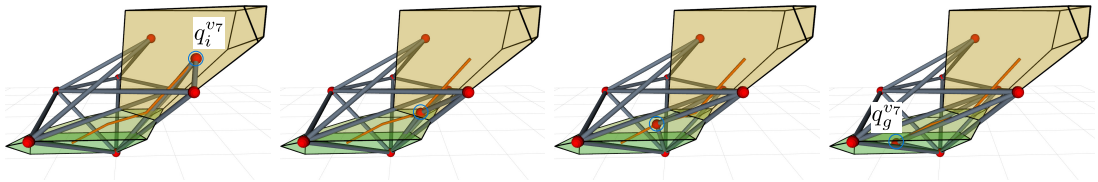
$$
\begin{aligned}
q^{v_0} &= [1.0, 1.0, -0.3]^\intercal \quad & q^{v_1} &= [0, -1.4, 0]^\intercal \\
q^{v_2} &= [-1.0, 1.0, 0.3]^\intercal \quad & q^{v_3} &= [-0.5851, -0.1749, 0.8851]^\intercal \\
q^{v_4} &= [1.0, -0.3, 0]^\intercal \quad & q^{v_5} &= [0, 0, -1.0]^\intercal \\
q^{v_6} &= [-1.0, -0.3, 0]^\intercal \quad & q^{v_7} &= [-1.107, 0.6449, 0.6597]^\intercal
\end{aligned}
$$

The task is to move node v_7 from its initial configuration $q_i^{v_7}$ to a goal configuration



(a) (b)

Figure 12.8: (a) A VTT is constructed from 18 members with 8 nodes. (b) $\mathcal{C}_{\text{free}}^{v_7}(q_i^{v_7})$ is decomposed into 56 cells in total.



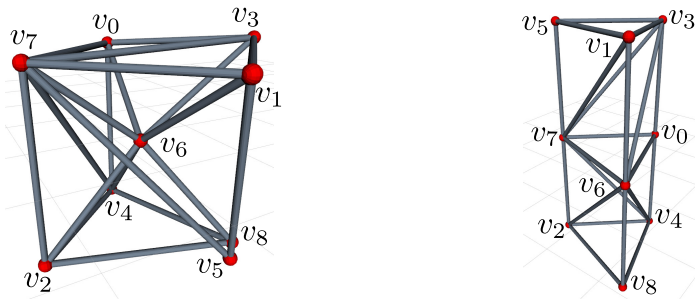
(a) (b) (c) (d)

Figure 12.9: The task is to move node v_7 from its initial configuration $q_i^{v_7}$ shown in (a) to a goal configuration $q_g^{v_7}$ shown in (d). The three cells and the complete path node v_7 has to traverse are shown. v_7 is moved to the intersection between the first cell and the second cell shown in (b), then to the center of the second cell shown in (c), and finally to the goal location inside the third cell.

$q_g^{v_7} = [0.6666, 0.23333, 0.375]^\intercal$ shown in Figure 12.9d. This is extremely hard for the retraction-based RRT because the space is narrow. The cell decomposition result is shown in Figure 12.8b with 56 cells in total. For this task, node v_7 needs to traverse three cells as shown in Figure 12.9. The planning process takes 1.054 s, among which CGAL takes 0.959 s.

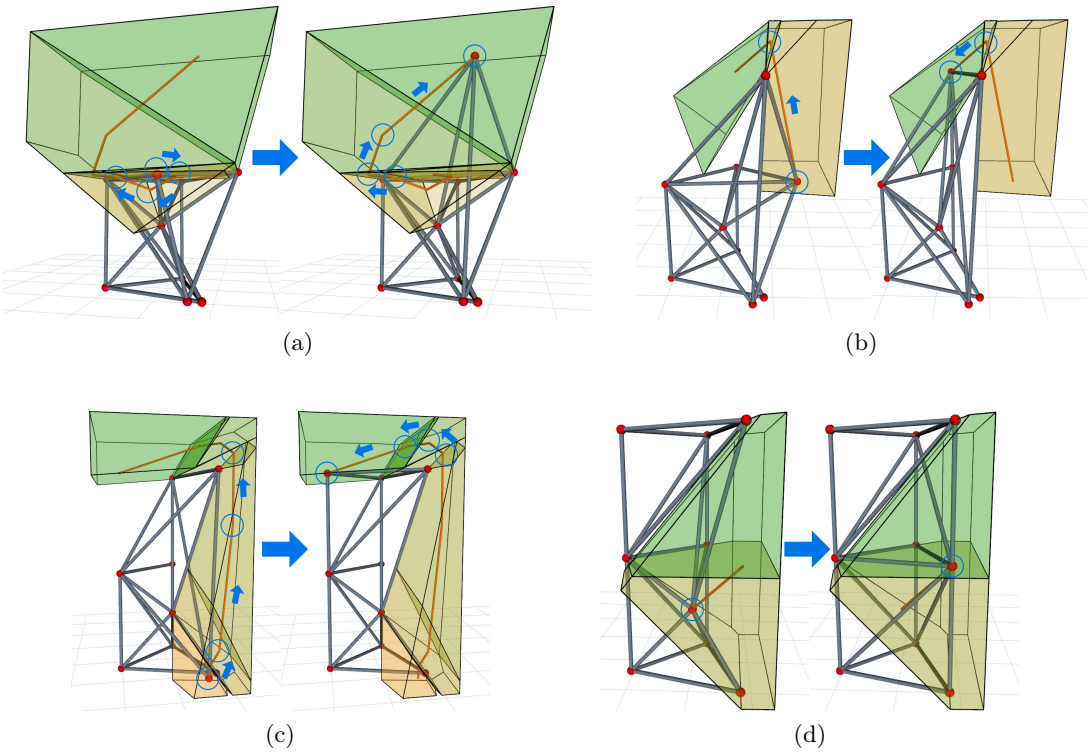
12.5.2 Multi-Node Experiment

The experiment for multi-node planning is to change the shape of a cubic VTT shown in Fig 12.10a to a tower VTT shown in Figure 12.10b. The VTT is composed of 21 members



(a) (b)

Figure 12.10: The motion task is to change the shape of a VTT from a cubic truss for rolling locomotion (a) to a tower truss for shoring (b).



(a) (b)

(c) (d)

Figure 12.11: The nodes are all encircled by "○" and their complete paths are shown as "—". (a) For node v_1, there are 61 cells generated after the cell decomposition process and it has to traverse five cells to go to the destination. (b) For node v_3, there are 87 cells generated after the cell decomposition process and it has to traverse three cells to go to the destination. (c) For node v_5, there are 40 cells generated after the cell decomposition process and it has to traverse seven cells to go to the destination. (d) For node v_6, there are 34 cells generated after the cell decomposition process and it has to traverse two cells to go to the destination.

and 9 nodes. The initial positions of all nodes are listed in the following:

$$
\begin{aligned}
q^{v_0} &= [-1.605, -0.771, 2.075]^\intercal & q^{v_1} &= [0.7779, -0.7642, 2.075]^\intercal \\
q^{v_2} &= [-0.4756, -2.022, 0.075]^\intercal & q^{v_3} &= [-0.4142, 0.4228, 2.175]^\intercal \\
q^{v_4} &= [-1.605, -0.771, 0.075]^\intercal & q^{v_5} &= [0.3819, -0.3707, 0.125]^\intercal \\
q^{v_6} &= [-0.4314, -0.9559, 1.2321]^\intercal & q^{v_7} &= [-0.4756, -2.022, 2.075]^\intercal \\
q^{v_8} &= [0.1819, -0.1707, 0.075]^\intercal
\end{aligned}
$$

There are four nodes v_1, v_3, v_5, and v_6 involved in this shape morphing process, and their goal positions are $q_g^{v_1} = [0.1819, -0.1707, 4.125]^\intercal$, $q_g^{v_3} = [-1.605, -0.771, 4.075]^\intercal$, $q_g^{v_5} = [-0.4756, -2.022, 4.075]^\intercal$, and $q_g^{v_6} = [0.1819, -0.1707, 2.125]^\intercal$. The approach is able to find a valid sequence of moving each node and the result is to move v_1, v_3, v_5, and v_6 in order. The motions for all involved nodes are shown in Figure 12.11a, Figure 12.11b, Figure 12.11c, and Figure 12.11d respectively. It takes 4.004 s to find the path to do shape morphing and 3.509 s is consumed by CGAL. This experiment is also tested in [52] and, with the retraction-based RRT algorithm, a waypoint that node v_5 has to be moved higher than node v_1 needs to be added manually to mitigate the narrow passage problem.

12.6 Conclusion

A fast and complete approach is presented to compute the configuration space of a given node in a VTT which makes it possible to do cell decomposition more efficiently and accurately. Then a simple graph-based search algorithm can be used to generate an optimal path for single-node motion tasks. Based on this approach, a shape morphing method for VTT systems is introduced by changing the locations of multiple nodes. Some useful applications are used to demonstrate the effectiveness of the approach.

Chapter 13

Reconfiguration

This chapter presents the reconfiguration planning framework for VTTs. The goal is to allow a VTT to perform dexterous reconfiguration activities in order to handle a variety of tasks. This chapter excerpts heavily from [82] and [83]. Credit is due to Sencheng Yu who contributed significantly to this work.

VTT robots are able to change their configurations by not only controlling joint positions (edge module lengths) but also reconfiguring the connections among edge modules. Motion planning for VTT robots is difficult due to their non-fixed morphologies, high dimensionality, the potential for self-collision, and complex motion constraints. In this chapter, a new reconfiguration planning algorithm to dramatically alter the structure of a VTT is presented, as well as some comparative tests to show its effectiveness.

13.1 Introduction

A significant advantage for self-reconfigurable modular robots over other robots with fixed morphologies is their versatility, namely they are able to adapt themselves into different morphologies with regard to different requirements. As mentioned, a VTT in which the members form a broad supported structure is well-suited for shoring damaged buildings to enable rescue operations. Another VTT with some members protruding to form an arm may have a large reachable workspace that is good at manipulation tasks. However, a fundamental complication with VTT systems comes from the motion of the complex parallel

structures that may result in self-collision. Developing self-collision-free motion plans is difficult due to the large number of DOFs leading to large search space.

Geometry reconfiguration involves the motions of multiple nodes. However, the arrangement of edge modules in a VTT can result in a complicated configuration space even for a single node while keeping the others rigid, and motions of multiple nodes are strongly coupled, namely moving one node can significantly affect the configuration space of other nodes. In a topology reconfiguration process, a single node controlled by enough edge modules can be split into a pair of nodes so that they can go around some internal blocking members and then merge back to an individual node. This process also requires motion planning for two nodes at the same time. In addition, a number of physical constraints need to be considered.

This chapter presents a new framework for the reconfiguration planning of an arbitrary VTT. This method dramatically reduces the search space when planning for multiple node motions greatly improving efficiency. Furthermore, a fast method to compute the configuration space which is often not fully connected can explicitly answer whether a topology reconfiguration action is required for a motion goal. An algorithm to compute a sequence of topology reconfiguration actions is then introduced, if required, for a motion task.

13.2 Related Work

Some approaches have been developed for VGT systems that are similar to VTT systems but without topology reconfiguration capability. Hamlin and Sanderson [41] presented a kinematic control but is limited to tetrahedrons or octahedrons. Usevitch *et al.* [148] introduced linear actuator robots (LARs) as well as a shape morphing algorithm. These systems are in mesh graph topology constructed by multiple convex hulls, and therefore self-collision can be avoided easily. However, this does not apply to VTT systems because edge modules span the workspace in a very non-uniform manner. There has been some work on VTT motion planning. The retraction-based RRT algorithm was developed in [52] in order to handle this high dimensional problem with narrow passage difficulty that is a well-known issue in sampling-based planning approaches; nevertheless, this approach is not efficient because it samples the whole workspace for every node and collision checking needs to be done for

every pair of members. Also, sometimes waypoints have to be assigned manually.

Chapter 11 discusses the advantage of topology reconfiguration and presents a reconfiguration planning framework inspired by the DNA replication process — the topology of DNA can be changed by cutting and resealing strands as tanglements form. This work is based on a new method to discretize the workspace depending on the space density and an efficient way to check self-collision. Both topology reconfiguration actions and geometry reconfiguration actions are involved if needed. However, only a single node is involved in each step and the transition model is more complicated, which makes the algorithm limited in efficiency. Chapter 12 develops an algorithm to compute the configuration space of a given node which can be used to plan a path for this node efficiently. However, multiple nodes are usually involved in shape morphing. In this chapter, the approach is extended to compute the obstacles for multiple nodes so that the search space can be decreased significantly. In addition, only the collision among a small number of edge modules needs to be considered when moving multiple nodes at the same time. Hence, sampling-based planners can be applied efficiently. This idea has been discussed briefly in [77]. For some motion tasks, topology reconfiguration is required. An updated algorithm is developed to compute the whole not fully connected free space and required topology reconfiguration actions can then be generated using a hybrid planning framework (sampling-based and search-based) which can achieve behaviors that are similar to the DNA replication process.

13.3 Problem Statement

Given a node $v \in V$ in $G = (V, E)$, its *obstacle region* is defined in Eq. (12.1) and its *free space* is defined in Eq. (12.2). However, $\mathcal{C}^v_{\text{free}}$ may not be fully connected and may be partitioned by $\mathcal{C}^v_{\text{obs}}$. Only the enclosed subspace containing q^v which is denoted as $\mathcal{C}^v_{\text{free}}(q^v)$ is free for node v to move. The algorithm to compute the boundary of this subspace is shown in Chapter 12. For example, given the VTT in Figure 12.2a, $\mathcal{C}^{v_0}_{\text{obs}}$ — the obstacle region of v_0 — is shown in Figure 12.2b, $\mathcal{C}^{v_0}_{\text{free}}$ — the free space of v_0 — is just the leftover configurations, and the enclosed subspace $\mathcal{C}^{v_0}_{\text{free}}(q^{v_0})$ is shown in Figure 12.6.

$\mathcal{C}^v_{\text{free}}$ is usually partitioned by $\mathcal{C}^v_{\text{obs}}$ into multiple enclosed subspaces, and it is impossible

to move v from one enclosed subspace to another one without topology reconfiguration. The physical system constraints from [132] allow two atomic actions on nodes that enable topology reconfiguration: `Split` and `Merge`. Since the physical system must be statically determinate with all nodes of degree three, a node v must be composed of six or more edge modules to undock and split into two new nodes v' and v'', and both nodes should still have three or more members. This process is called `Split`. Two separate nodes are able to merge into an individual one in a `Merge` action. The simulation of these two actions is shown in Figure 10.1. Hence, in a topology reconfiguration process, the number of nodes can change, but the number of members that are physical elements remains constant.

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

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

- **Topology Reconfiguration**. Compute the topology reconfiguration actions, `Merge` and `Split`, and find collision-free path(s) to move a node v from its initial position q_i^v to its goal position q_g^v, while satisfying all constraints.

13.4 Hardware and Environmental Constraints

A VTT has to maintain some hardware constraints during its motion, and this includes the limitations from the hardware, and the stability and controllability of the system.

13.4.1 Length Constraints

In a VTT, each edge module has an active prismatic joint called Spiral Zipper [24] and this joint is able to achieve a high extension ratio as well as form a high strength-to-weight-ratio column. The mechanical components determine the minimum member length and the total material determines the maximum member length. Hence, in a VTT $G = (V, E)$,

$\forall e = (v_i, v_j) \in E$, the following constraint can be obtained:

$$\overline{L}_{\min}^2 \leq \|q^{v_i} - q^{v_j}\| \leq \overline{L}_{\max}^2 \tag{13.1}$$

13.4.2 Collision Avoidance

During the motion of a VTT, the collision between members have to be avoided. The distance between every two members have to be greater than $\bar{d}_{\min}$, the diameter of an edge module which is a cylinder (or the sum of the radius of a node and the radius of an edge module). The minimum distance between member (v_i, v_j) and (v_m, v_n) can be expressed as

$$\min \|(q^{v_i} + \alpha(q^{v_j} - q^{v_i})) - (q^{v_m} + \gamma(q^{v_n} - q^{v_m}))\| \tag{13.2}$$

in which $\alpha, \gamma \in (0, 1)$. This is not easy to compute and can be more complicated when both members are actuating. In Section 13.5, the presented approach that doesn't need to solve this problem is shown. Also the angle between connected members has to remain above a certain value due to the mechanical design of the node. The angle constraint between member (v_i, v_j) and (v_i, v_k) can be expressed as

$$\arccos\left(\frac{(q^{v_j} - q^{v_i}) \bullet (q^{v_k} - q^{v_i})}{\|q^{v_j} - q^{v_i}\|\|q^{v_k} - q^{v_i}\|}\right) \geq \bar{\theta}_{\min} \tag{13.3}$$

13.4.3 Stability

The truss structure of a VTT is meant to be statically stable under gravity when interacting with the environment and sufficient constraints between the robot and the environment must be satisfied to ensure the location of the structure is fully defined. At least three still nodes should be on the ground in order to form a valid support polygon. For the VTT shown in Figure 10.2, v_0, v_2, v_3, and v_5 are stationary on the ground to form the current support polygon. In addition, the center of mass of a VTT as represented on the ground has to be inside this support polygon. Furthermore, no collision is allowed between a VTT and the environment, and the simplest condition is that all nodes have to be above the ground.

13.4.4 Manipulability

In order to control the motions of nodes, a VTT has to maintain an amount of manipulability for these moving nodes. Given a VTT $G = (V, E)$ and the current controlled node set V_C, the Jacobian J_{AB} can be derived as shown in Eq. (10.6b). By applying singular value decomposition on J_{AB}, its maximum singular value $\sigma_{\max}(J_{\text{AB}})$ and minimum singular value $\sigma_{\min}(J_{\text{AB}})$ can be derived, and the manipulability of the current moving nodes can be constrained as

$$\mu = \frac{\sigma_{\min}(J_{\text{AB}})}{\sigma_{\max}(J_{\text{AB}})} \geq \bar{\mu}_{\min} \tag{13.4}$$

13.5 Geometry Reconfiguration

The overall shape of a VTT is altered by moving nodes around in the workspace. For an individual node, its configuration space is $\mathbb{R}^3$. Apparently, the configuration space for n nodes is $\mathbb{R}^{3n}$. When multiple nodes are involved, the motion planning problem will be in high-dimensional space. The strategy to avoid this high dimensionality is to divide the moving nodes into multiple groups and each group contains one or a pair of nodes. The motion planning space for each group is either in $\mathbb{R}^3$ or $\mathbb{R}^6$. Even with lower-dimensional space, it is still a challenge to search for a valid solution. First, it is not efficient to search the whole $\mathbb{R}^3$ or $\mathbb{R}^6$ and narrow passage is a significant problem when applying sampling-based algorithms, e.g. RRT. In addition, it is difficult to guarantee the collision avoidance for a given motion by simply discretizing the motion to some resolution and checking states at that resolution because of the case that a member goes across another member may not be detected, or very fine resolution has to be used which will slow down the planner significantly. These issues are overcome by computing the free space of the group in advance so that the sampling space is decreased significantly.

13.5.1 Obstacle Region and Free Space

A group can contain either one node or a pair of nodes. Given a VTT $G = (V, E)$, for an individual node $v \in V$, an efficient algorithm to compute $\mathcal{C}^v_{\text{obs}}$ and $\mathcal{C}^v_{\text{free}}(q^v)$ with its boundary

is introduced in Chapter 12. If there are two nodes $v_i \in V$ and $v_j \in V$ in a group, then any collision among members in E^{v_i} and E^{v_j} is treated as self-collision inside the group, and all the members in $E \setminus (E^{v_i} \cup E^{v_j})$ define the obstacle region of this group denoted as $\widehat{\mathcal{C}}^{v_i}_{\text{obs}}$ (the obstacle region of v_i in the group) and $\widehat{\mathcal{C}}^{v_j}_{\text{obs}}$ (the obstacle region of v_j in the group) respectively, namely

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

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

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

$$\widehat{\mathcal{C}}^{v_i}_{\text{free}} = \mathbb{R}^3 \setminus \widehat{\mathcal{C}}^{v_i}_{\text{obs}} \tag{13.5}$$
$$\widehat{\mathcal{C}}^{v_j}_{\text{free}} = \mathbb{R}^3 \setminus \widehat{\mathcal{C}}^{v_j}_{\text{obs}} \tag{13.6}$$

Using the same boundary search approach in Section 12.2.2, the boundary of $\widehat{\mathcal{C}}^{v_i}_{\text{free}}(q^{v_i})$ — the enclosed subspace containing the current position of node v_i — can be obtained efficiently. Similarly, the boundary of $\widehat{\mathcal{C}}^{v_j}_{\text{free}}(q^{v_j})$ can be obtained. For example, given the VTT shown in Figure 13.1, if node v_0 and v_1 form a group, then $\widehat{\mathcal{C}}^{v_0}_{\text{free}}(q^{v_0})$ and $\widehat{\mathcal{C}}^{v_1}_{\text{free}}(q^{v_1})$ can be computed and shown in Figure 13.2. $\mathcal{C}^{v_0}_{\text{free}}(q^{v_0})$ contains some space on the right side of v_1 because E^{v_1} is ignored. This space informs that it is possible to move v_0 to some locations which are currently blocked by v_1 since v_1 can be moved away. It is guaranteed that as long as v_0 is moving inside $\widehat{\mathcal{C}}^{v_0}_{\text{free}}(q^{v_0})$ (the space shown in Figure 13.2a), there must be no collision between any member in E^{v_0} and any member in $E \setminus (E^{v_0} \cup E^{v_1})$. Similarly, no collision between any member in E^{v_1} and any member in $E \setminus (E^{v_0} \cup E^{v_1})$ can happen if v_1 is moving inside $\widehat{\mathcal{C}}^{v_1}_{\text{free}}(q^{v_1})$ (the space shown in Figure 13.2b). In this way, when planning the motion of node v_0 and v_1 using RRT, the sample will only be generated inside $\widehat{\mathcal{C}}^{v_0}_{\text{free}}(q^{v_0})$ and $\widehat{\mathcal{C}}^{v_1}_{\text{free}}(q^{v_1})$, and only self-collision in the group needs to be considered, namely the collision can only happen among members in $E^{v_0} \cup E^{v_1}$. There is a special case when these two nodes in

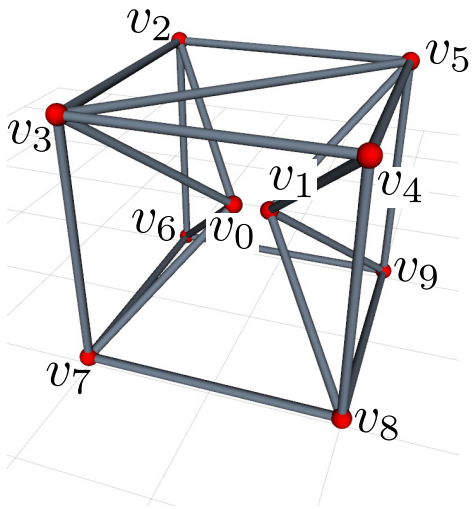


Figure 13.1: A VTT is composed of 21 edge modules with 10 nodes among which v_0 and v_1 form a group.

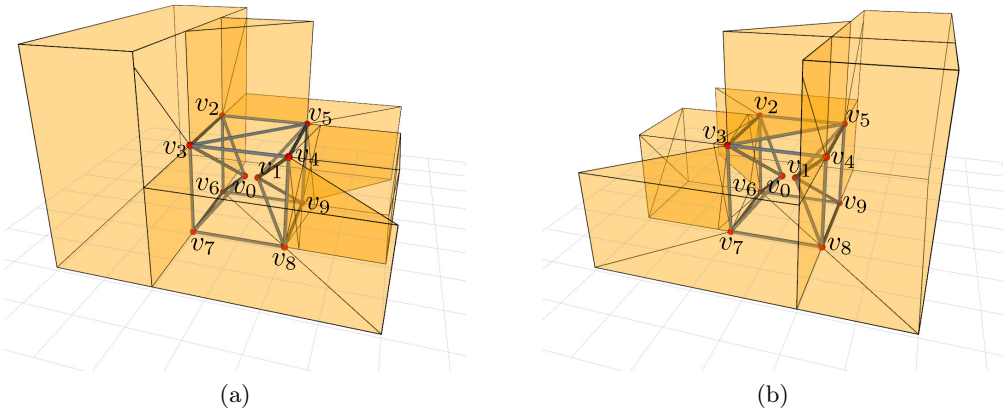


Figure 13.2: (a) $\widehat{\mathcal{C}}_{\text{free}}^{v_0}(q^{v_0})$ is computed with all members controlling v_1 ignored. (b) $\widehat{\mathcal{C}}_{\text{free}}^{v_1}(q^{v_1})$ is computed with all members controlling v_0 ignored.

the group are connected by a member. Both ends of the member are moving which is not considered by the obstacle model. This is an extra case when doing collision check.

The previous free space and the obstacle region are derived by ignoring the physical size of members and nodes, such as the obstacle polygon shown in Figure 12.2a that is derived by regarding node v_1 as a point and member (v_6, v_8) as a line segment without thickness. In practice, a node is a sphere and a member is a cylinder, so it is not guaranteed that the

minimum distance among members (Eq. 13.2) is greater than the diameter of an edge module (or the sum of the radius of a node and the radius of an edge module) even the involved node is not inside its obstacle region. This difficulty is overcome through increasing the derived obstacle region by considering the physical size of VTT components. In the obstacle region, every polygon is converted into a polyhedron. For example, for the VTT shown in Figure 12.2a, the purple polygon defined by node v_1 and edge module (v_6, v_8) is one obstacle polygon for node v_0, and this polygon is converted into a polyhedron shown in Figure 13.3. The boundary of this obstacle polyhedron is composed of five polygons that can be obtained as follows:

The current locations of nodes v_1, v_6, and v_8 are q^{v_1}, q^{v_6}, and q^{v_8} respectively, and the unit vector normal to the plane formed by these three nodes is $\hat{n}_p$ ($-\hat{n}_p$ is also a normal unit vector but in opposite direction). As shown in Figure 13.4b, first adjust q^{v_6} and q^{v_8} by moving them along $\hat{n}_p$ and $-\hat{n}_p$ respectively:

$$\hat{q}_1^{v_6} = q^{v_6} + \lambda \hat{n}_p$$
$$\hat{q}_2^{v_6} = q^{v_6} - \lambda \hat{n}_p$$
$$\hat{q}_1^{v_8} = q^{v_8} + \lambda \hat{n}_p$$
$$\hat{q}_2^{v_8} = q^{v_8} - \lambda \hat{n}_p$$

where λ is the growing size that is taken to be the sum of the radius of the node and the radius of the edge. Then, the four rays shown in Figure 13.4a can be easily derived as

$$\hat{r}_1^{16} = \frac{\hat{q}_1^{v_6} - q^{v_1}}{\|\hat{q}_1^{v_6} - q^{v_1}\|}$$
$$\hat{r}_2^{16} = \frac{\hat{q}_2^{v_6} - q^{v_1}}{\|\hat{q}_2^{v_6} - q^{v_1}\|}$$
$$\hat{r}_1^{18} = \frac{\hat{q}_1^{v_8} - q^{v_1}}{\|\hat{q}_1^{v_8} - q^{v_1}\|}$$
$$\hat{r}_2^{18} = \frac{\hat{q}_2^{v_8} - q^{v_1}}{\|\hat{q}_2^{v_8} - q^{v_1}\|}$$

$\hat{n}_m$ is the unit vector perpendicular to the edge $(v_6,\ v_8)$, pointing to v_1 and lying on the

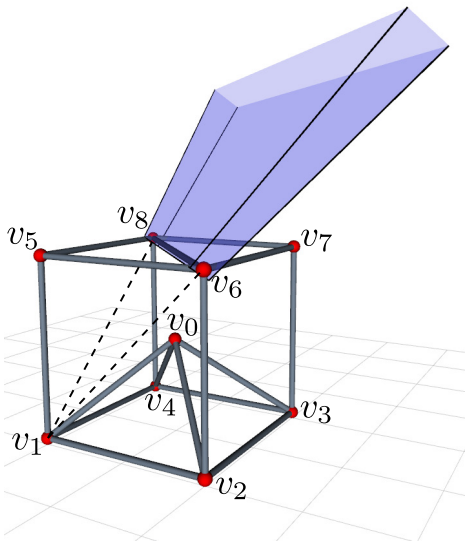


Figure 13.3: One obstacle polygon of $\mathcal{C}_{\text{obs}}^{v_0}$ shown in Figure 12.2a becomes a polyhedron bounded by five polygons if the size of VTT components is considered.

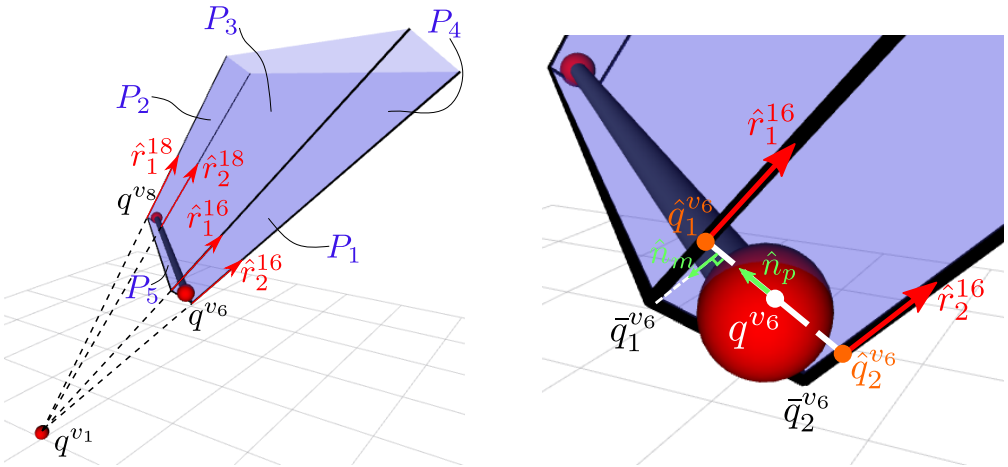


Figure 13.4: (a) Detailed illustration of the formation of the obstacle polyhedron. (b) Close view of the obstacle polyhedron.

plane formed by v_1, v_6, and v_8. Then $\hat{q}_1^{v_6}$, $\hat{q}_2^{v_6}$, $\hat{q}_1^{v_8}$, and $\hat{q}_2^{v_8}$ are moved by distance $\lambda/2$ projected on $\hat{n}_m$ along $-\hat{r}_1^{16}$, $-\hat{r}_2^{16}$, $-\hat{r}_1^{18}$, and $-\hat{r}_2^{18}$, respectively:

$$\bar{q}_1^{v_6} = \hat{q}_1^{v_6} - \frac{\lambda}{2\left|\hat{n}_m \cdot \hat{r}_1^{16}\right|}\hat{r}_1^{16}$$
$$\bar{q}_2^{v_6} = \hat{q}_2^{v_6} - \frac{\lambda}{2\left|\hat{n}_m \cdot \hat{r}_2^{16}\right|}\hat{r}_2^{16}$$
$$\bar{q}_1^{v_8} = \hat{q}_1^{v_8} - \frac{\lambda}{2\left|\hat{n}_m \cdot \hat{r}_1^{18}\right|}\hat{r}_1^{18}$$
$$\bar{q}_2^{v_8} = \hat{q}_2^{v_8} - \frac{\lambda}{2\left|\hat{n}_m \cdot \hat{r}_2^{18}\right|}\hat{r}_2^{18}$$

With all the information, the boundary of the polyhedron that is composed of five polygons shown in Figure 13.4a can be encoded in the following:

$$P_1 = \left\{V = (\bar{q}_1^{v_6},\ \bar{q}_2^{v_6}),\ R = \left(\hat{r}_1^{16},\ \hat{r}_2^{16}\right)\right\}$$
$$P_2 = \left\{V = (\bar{q}_1^{v_8},\ \bar{q}_2^{v_8}),\ R = \left(\hat{r}_1^{18},\ \hat{r}_2^{18}\right)\right\}$$
$$P_3 = \left\{V = (\bar{q}_1^{v_6},\ \bar{q}_1^{v_8}),\ R = \left(\hat{r}_1^{16},\ \hat{r}_1^{18}\right)\right\}$$
$$P_4 = \left\{V = (\bar{q}_2^{v_6},\ \bar{q}_2^{v_8}),\ R = \left(\hat{r}_2^{16},\ \hat{r}_2^{18}\right)\right\}$$
$$P_5 = \{V = (\bar{q}_1^{v_6},\ \bar{q}_2^{v_6},\ \bar{q}_1^{v_8},\ \bar{q}_2^{v_8})\}$$

Note that P_5 consists of no rays.

Every obstacle polygon in Figure 12.2b can be converted into an obstacle polyhedron (bounded by five polygons) in this way, and then process these polygons by *Polygon Intersection* and *Boundary Search* from Section 12.2.2 to derive $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$ shown in Figure 13.5. The *Boundary Search* step can be simplified and the modified boundary search algorithm is shown in Algorithm 8. Recall that the free space of a node is usually partitioned by its obstacle region into multiple enclosed subspaces, and given an obstacle polygon P_{s} and the set of all obstacle polygons $\mathcal{P}_{\text{obs}}$ generated by *Polygon Intersection* step, this algorithm can find the enclosed subspace with P_{s} being part of the boundary. Note that after *Polygon Intersection*, each polygon in $\mathcal{P}_{\text{obs}}$ can only bound one enclosed subspace because every ob-

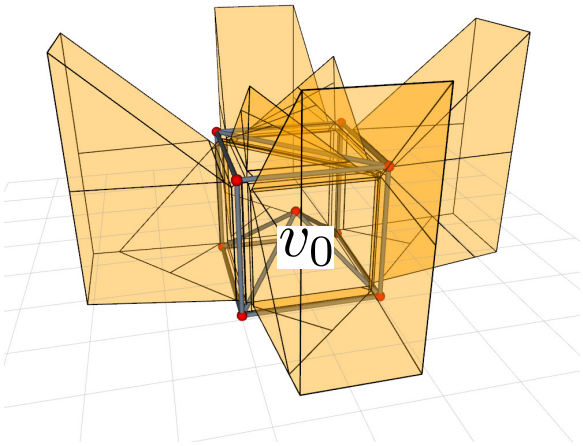


Figure 13.5: $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$ with physical size of the robot components being considered.

Algorithm 8: Boundary Search Algorithm

Input: One obstacle polygon P_{s}, the set of obstacle polygons $\mathcal{P}_{\text{obs}}$
Output: The set of boundary polygons $\mathcal{P}_b$

```
1  P_b ← ∅;
2  Q_P ← ∅;
3  Q_P.enqueue(P_s);
4  while Q_P ≠ ∅ do
5  |   P_i ← Q_P.dequeue();
6  |   P_b ← P_b ∪ {P_i};
7  |   foreach s_ij ∈ S_i do
8  |   |   P̄_ij ← the innermost polygon in N_ij;
9  |   |   if P̄_ij ∉ P_b ∧ P̄_ij ∉ Q_P then
10 |   |   |   Q_P.enqueue(P̄_ij);
11 |   |   end
12 |   end
13 end
```

stacle polygon is the boundary between the free space and the obstacle region. In order to find the boundary of $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$, set P_{s} to be the obstacle polygon that is closest to q^{v_0}. In the algorithm, S_i is the set of all edges of polygon P_i, s_{ij} is the jth edge of P_i, and $\mathcal{N}_{ij}$ is the set of all polygons that share s_{ij} with P_i. For each obstacle polygon P_i, its normal vector pointing outward the obstacle region is computed, and $\overline{P}_{ij}$ in *Line 8* is the innermost polygon in $\mathcal{N}_{ij}$ along the normal vector of P_i.

The new $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$ is smaller and bounded by more polygons. In addition to considering

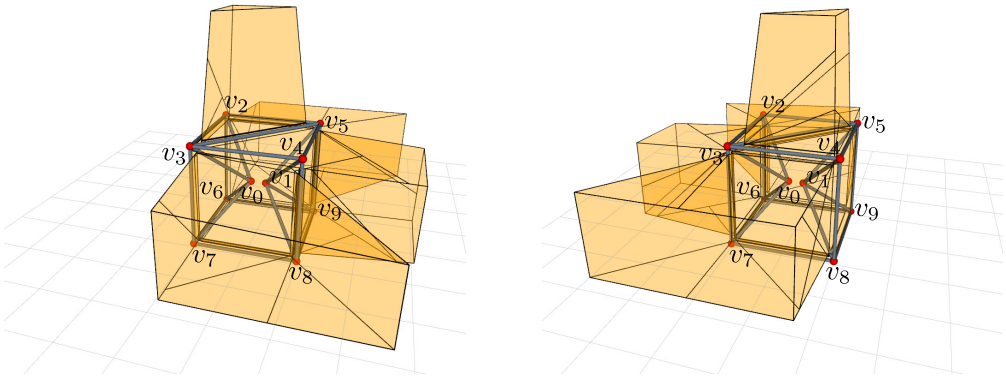


Figure 13.6: (a) $\widehat{\mathcal{C}}^{v_0}_{\text{free}}(q^{v_0})$ is computed with all members controlling v_1 ignored. (b) $\widehat{\mathcal{C}}^{v_1}_{\text{free}}(q^{v_1})$ is computed with all members controlling v_0 ignored.

the physical size of the robot components, for a single node, it is easy to find the region when the node is in a singular configuration. For a single node, if all of its neighbor nodes are on the same plane, then this node cannot traverse this plane, or it will be in a singular configuration. In such a case, add this plane to the obstacle polygon set when running the boundary search algorithm. By taking these constraints into consideration, $\widehat{\mathcal{C}}^{v_0}_{\text{free}}(q^{v_0})$ and $\widehat{\mathcal{C}}^{v_1}_{\text{free}}(q^{v_1})$ can be derived for the group containing v_0 and v_1 in the VTT shown in Figure 13.1 and the result is shown in Figure 13.6. Compared with the space shown in Figure 13.2, $\mathcal{C}^{v_0}_{\text{free}}(q^{v_0})$ becomes smaller and v_0 cannot go beyond the plane formed by v_2, v_3, v_6, and v_7 due to the singularity constraint. In the rest of this chapter, the free space ignoring the physical size of the robot components is used for illustration purposes.

13.5.2 Path Planning for a Group of Nodes

If there is only one node v in the group and the motion task is to move the node from its initial position q^v_i to its goal position q^v_g where $q^v_i \in \mathcal{C}^v_{\text{free}}(q^v_i)$ and $q^v_g \in \mathcal{C}^v_{\text{free}}(q^v_i)$ (q^v_i and q^v_g are in the same enclosed subspace), then it is straightforward to apply RRT approach in $\mathcal{C}^v_{\text{free}}(q^v_i)$ and no collision can happen as long as the motion of each step is inside $\mathcal{C}^v_{\text{free}}(q^v_i)$ since this space is usually not convex.

When moving two nodes v_i and v_j in a group, sampling will only happen inside $\widehat{\mathcal{C}}^{v_i}_{\text{free}}(q^{v_i})$

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

In summary, when planning node v_i and v_j in a VTT $G = (V, E)$, for each step, in order to avoid collision, it is required to ensure the following:

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

It is difficult to check the second and third collision case during the motion if both nodes are moving simultaneously. But since the step size for each node is limited and both of them are moving in straight lines, the planner can first check the collision during the motion of v_i while keeping v_j fixed, and then check the motion of v_j. By doing so, the collision can be checked efficiently using the approach presented in Section 11.2.3. Every edge module can be modeled as a line segment in space, thus, when moving node v, every $e \in E^v$ sweeps a triangle area, and if this member collides with another member $\bar{e} \in E$, then $\bar{e}$ must intersect with the triangle generated by e (Figure 11.4). Similar to the approach in Section 13.5.1, buffer this triangle area in order to consider the size of physical components. Another way is to compute the local obstacle region of v_i by only taking E^{v_j} into account when moving v_i, and similarly compute the local obstacle region of v_j by only taking E^{v_i} into account when moving v_j. The local obstacle region of each node can be derived easily. For v_i, given $\mathcal{N}_G(v_i)$ that is the neighbors of v_i and E^{v_j}, the local obstacle region of v_i is simply the union

of all the obstacle polyhedron defined by $\forall(v,e) \in \mathcal{N}_G(v_i) \times E^{v_j}$. If the trajectory of v_i that is a line segment intersects with its local obstacle region, then collision happens. Repeat the same procedures when moving v_j. If edge module (v_i, v_j) exists, when moving v_i, it is also necessary to consider the obstacle polyhedron defined by v_j and $e \in E \setminus (E^{v_i} \cup E^{v_j})$, and similarly consider the obstacle polyhedron defined by v_i and $e \in E \setminus (E^{v_i} \cup E^{v_j})$ when moving v_j. By this approach, there is no need to solve Eq. (13.2) for collision avoidance.

In addition to this constraint that the distance between every pair of members has to be greater than some value, a discrete motion also needs to satisfy the length constraint (Eq. 13.1), the angle constraint (Eq. 13.3), the manipulability constraint (Eq. 13.4), and the stability constraint when interacting with the environment. For these constraints, it is straightforward to discretize the motion in the current step to some resolution and check the states for validity at this resolution. When checking the stability constraint, first find all supporting nodes by checking the height of every node, and if its height is close enough to the ground, then this node is regarded as a supporting node. There have to be at least three supporting nodes all the time. Otherwise, this VTT cannot be stable. Then the center of mass of the current truss is computed and projected onto the ground. Given this projected center of mass and all the supporting nodes on the ground, the convex hull of them (the smallest convex polygon that contains all these points) can be computed efficiently using either Jarvis's march [51] or Graham's scan [37]. If the projected center of mass is not on the boundary of the convex hull, then it is inside the support polygon that is the convex hull determined only by the support nodes. Otherwise, this VTT is not stable.

With all these works, the state validity and the motion validity can be checked efficiently, and RRT-type approaches can be applied easily. Open Motion Planning Library (OMPL) [137] is used to implement RRT for this path planning problem.

13.5.3 Geometry Reconfiguration Planning

Assuming there are n nodes $\{v_t \in V | t = 1, 2 \cdots, n\}$ that should be moved from their initial positions $q_i^{v_1}, q_i^{v_2}, \cdots, q_i^{v_n}$ to their goal positions $q_g^{v_1}, q_g^{v_2}, \cdots, q_g^{v_n}$ respectively, these nodes are first divided into $\lceil n/2 \rceil$ groups. Each group contains at most two nodes. Then the

motion task is achieved by moving nodes one group by one group. Then this geometry reconfiguration problem results in finding a sequence of groups that can achieve the task. If failed, try another grouping and find the corresponding sequence. Repeat this process until the task is finished, otherwise failed. With this approach, this geometry reconfiguration planning problem can be solved much faster than the approach in Section 12.4 and the detailed test results are in Section 13.7.

13.6 Topology Reconfiguration

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

13.6.1 Enclosed Subspace in Free Space

As mentioned before, each polygon after *Polygon Intersection* in $\mathcal{C}^v_{\text{obs}}$ is bounding only one enclosed subspace. Therefore all the enclosed subspaces can be computed by repeatedly applying Algorithm 8 as shown in Algorithm 9.

The algorithm first obtains the set of all obstacle polygons $\mathcal{P}^v_{\text{obs}}$ by *Polygon Intersection*. Then, search for the enclosed subspace containing the current node configuration — $\mathcal{C}^v_{\text{free}}(q^v)$ — from a starting polygon P_{s} that is the nearest one to the node (Section 12.2.2). Afterward, all other enclosed subspaces in $\mathcal{C}^v_{\text{free}}$ are computed by searching the boundary starting from any polygon that has not been used. Figure 13.7 shows two enclosed subspaces of node v_0 in a simple cubic truss. In total, there are 33 enclosed subspaces in $\mathcal{C}^{v_0}_{\text{free}}$ above the ground.

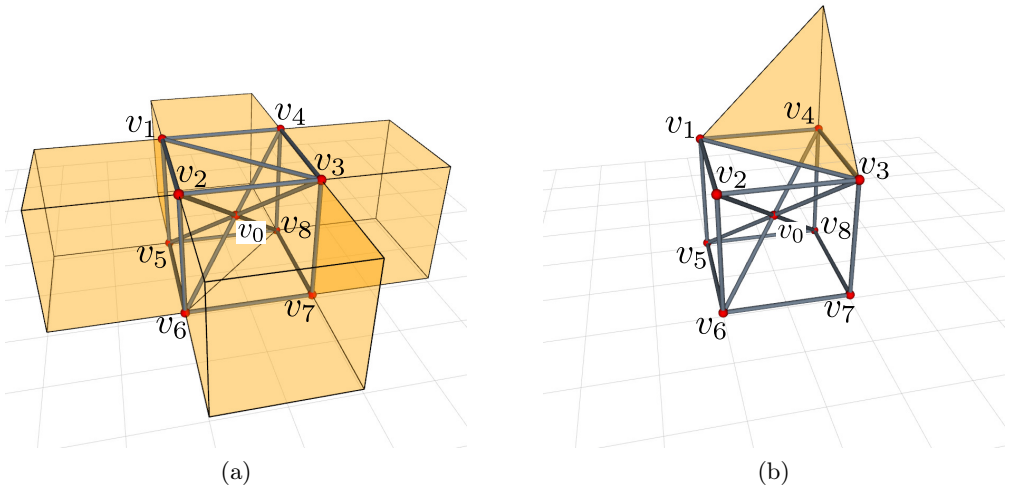
13.6.2 Topology Reconfiguration Actions

There are two topology reconfiguration actions: `Split` and `Merge`. A node constructed by six or more edge modules can be split into two separate nodes, and two separate nodes

Algorithm 9: Enclosed Subspace Search

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

Compute $\mathcal{P}_{\text{obs}}$;
$P_{\text{s}} \leftarrow$polygon closest to node v;
$\mathcal{C}^v_{\text{free}}(q^v) \leftarrow$ `BoundarySearch`$(P_{\text{s}},\ \mathcal{P}_{\text{obs}})$;
$\mathcal{C}^v_{\text{free}} \leftarrow \{\mathcal{C}^v_{\text{free}}(q^v)\}$;
foreach $P_i \in \mathcal{P}^v_{\text{obs}}$ **do**
    if $P_i \notin \mathcal{C}\ \forall \mathcal{C} \in \mathcal{C}^v_{\text{free}}$ **then**
        $\mathcal{C}_{\text{new}} \leftarrow$ `BoundarySearch`$(P_i,\ \mathcal{P}_{\text{obs}})$;
        $\mathcal{C}^v_{\text{free}} \leftarrow \mathcal{C}^v_{\text{free}} \cup \{\mathcal{C}_{\text{new}}\}$;
    end
end



(a) (b)

Figure 13.7: (a) Enclosed subspace $\mathcal{C}^{v_0}_{\text{free}}(q^{v_0})$ contains the current position of v_0. (b) Another enclosed subspace is separated from $\mathcal{C}^{v_0}_{\text{free}}(q^{v_0})$ by obstacles.

can merge into an individual node. These actions can significantly affect the motions of the involved nodes and there are several constraints when applying them. For the VTT shown in Figure 13.1, $\mathcal{C}^{v_0}_{\text{free}}(q^{v_0})$ that is the enclosed subspace containing the current location of node v_0 is shown in Figure 13.8a. This node can move to some locations outside the truss but its motion is also blocked in some directions. The members attached with node v_1 blocked the motion of node v_0 on this side, and the plane formed by node v_2, v_3, v_6, and v_7 also blocks the motion of v_0 due to: 1. collision avoidance with member (v_2, v_3), (v_2, v_6), (v_3, v_7), and

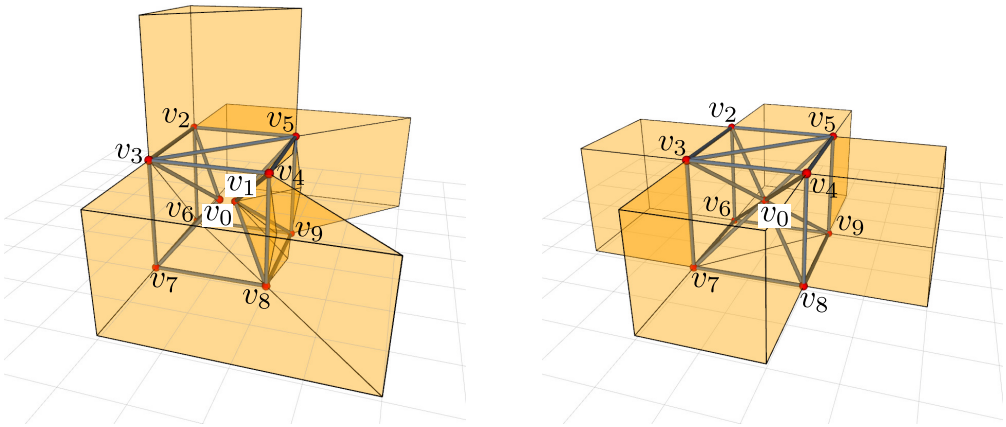
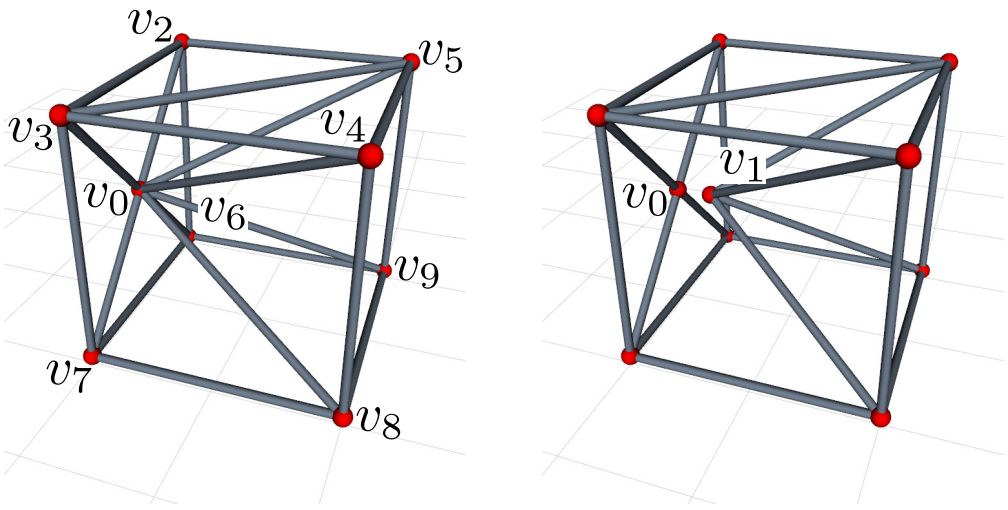


Figure 13.8: (a) Enclosed subspace $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$ when v_0 and v_1 are separated; (b) Enclosed subspace $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$ after merging v_1 with v_0.

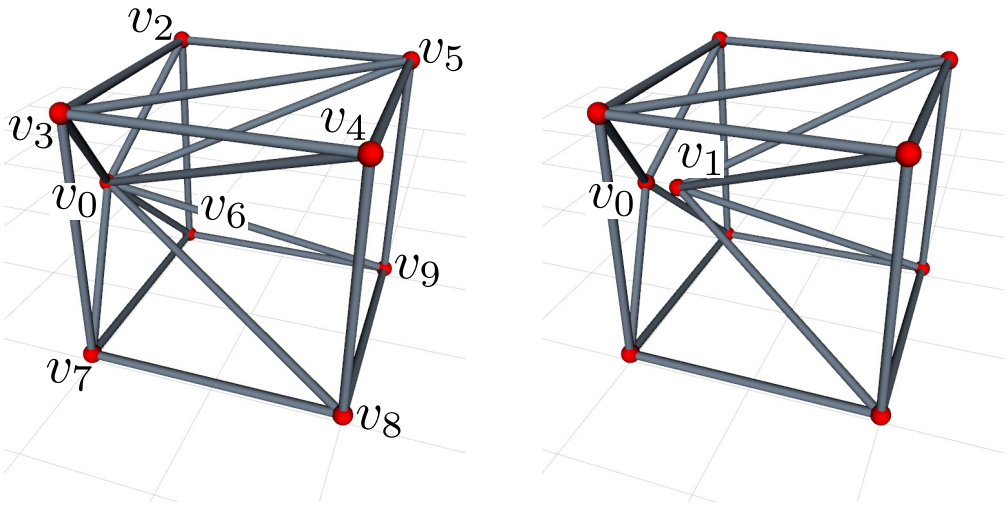
(v_6, v_7); 2. singularity avoidance that v_0 cannot move through the square formed by v_2, v_3, v_6, and v_7. If merging v_1 and v_0 as v_0 shown in Figure 13.8b, $\mathcal{C}_{\text{free}}^{v_0}(q^{v_0})$ will be changed. The boundary originally blocked by members attached with node v_1 disappears because all members controlling v_0 and v_1 are combined into a single set. In addition, this `Merge` action increases the motion manipulability of node v_0 that can move through the square formed by v_2, v_3, v_6, and v_7. However, the side effect is that some originally reachable locations become not reachable, such as the space above the truss and around the member (v_4, v_8). This is because E^{v_0} contains more members leading to the increase of the obstacle region of node v_0.

When executing a `Split` action on a node, the basic requirement is that this node has to be constructed by at least six edge modules. In the process, a node needs to be physically split into a pair which then be moved away. In order to guarantee motion controllability, singularity should be avoided during the whole process. In some situations, it is necessary to carefully consider if it is feasible to execute this action. For example, when node v_0 is on the same plane with node v_2, v_3, v_6, and v_7 shown in Figure 13.9a, it is not allowed to split node v_0 in the way shown in Figure 13.9b, because one of the newly generated nodes (v_0 in this case) will be in a singular configuration. In comparison, if node v_0 is outside the truss



(a) (b)

Figure 13.9: (a) Node v_0, v_2, v_3, v_6, and v_7 are on the same plane. (b) Splitting node v_0 in this way to separate E^{v_0} into two sets is not valid, because node v_0 will be in singular configuration. Similarly, it is not valid to merge v_1 with v_0.

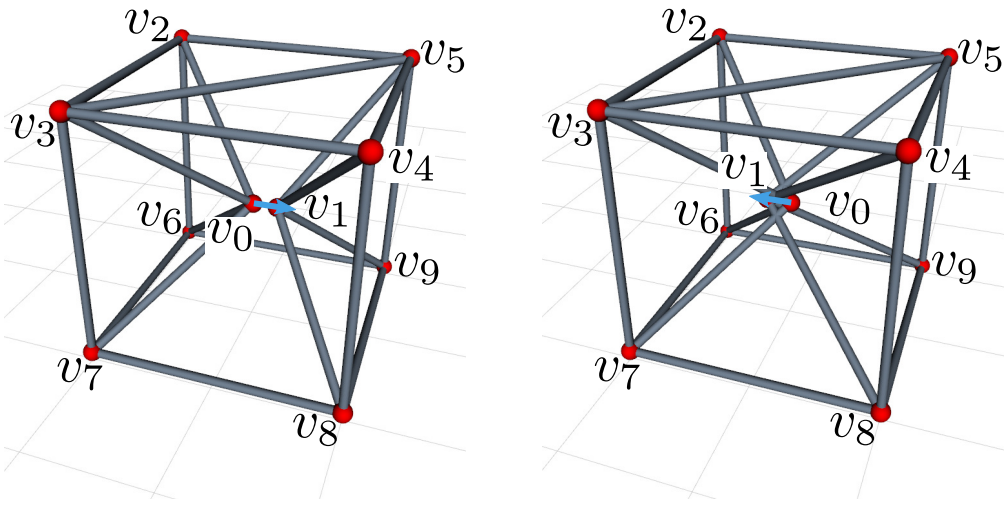


(a) (b)

Figure 13.10: (a) Node v_0 is outside the truss. (b) It is possible to split node v_0 in this way to generate v_1. Reversely, v_0 and v_1 can be merged.

shown in Figure 13.10a, then it is feasible to apply the same `Split` action to generate two new nodes shown in Figure 13.10b. This constraint also applies for `Merge`. In Figure 13.9b, since node v_0 is in singular configuration, it is not allowed to merge v_1 with v_0 to become a new truss shown in Figure 13.9a. However, it is feasible to merge v_0 and v_1 in Figure 13.10b to become v_0 in Figure 13.10a.

After splitting a node, the members attached to this node are separated into two groups that can be controlled independently. However, it is still necessary to consider how to move the two newly generate nodes away from each other. In Figure 13.11a, after splitting v_0 into v_0 and v_1, v_1 is moved to the right side of v_0 and this motion can separate these involved members without any collision. However, in Figure 13.11b, for the same `Split` action, v_1 is moved to the left side of v_0 and this motion is in fact not feasible because members will collide. This is also important for `Merge`. When merging two nodes, they first need to move to some locations that are close to each other. If the new locations are not selected correctly, it is impossible to move them there and execute `Merge` action.



(a) (b)

Figure 13.11: Split node v_0 into v_0 and v_1: (a) a right way to move node v_1 away from node v_0; (b) a wrong way to move node v_1 away from node v_0.

13.6.3 Topology Reconfiguration Planning

Given a VTT $G = (V, E)$ and the motion task that is to move node v from q_i^v to q_g^v, if q_i^v and q_g^v belong to the same enclosed subspace, then geometry reconfiguration planning is able to handle this problem by either the approach in Section 12.3 or the approach introduced in Section 13.5.2. Otherwise, topology reconfiguration is needed. The node has to execute a sequence of `Split` and `Merge` in order to avoid collision with other members.

There are multiple ways for a node v to split into nodes v' and v'' as there are multiple ways to take the members into two groups, and it is straightforward to compute all possible ways to split E^v into two groups in which both sets contain at least three edge modules. Let $\mathcal{A}$ be the set of all possible ways to separate E^v into two groups. If v is split into v' and v'', then E^v is separated into $E^{v'}$ and $E^{v''}$ accordingly. This split process is denoted as $(E^{v'}, E^{v''})$. Not all possible ways in $\mathcal{A}$ can be applied on node v. Given a valid VTT $G = (V, E)$ and a node v, a `Split` action can be encoded as a tuple $(E^{v'}, E^{v''}, q_{\mathrm{s}}^{v'}, q_{\mathrm{s}}^{v''})$ where v' and v'' are the newly generated node after splitting, and $q_{\mathrm{s}}^{v'}$ and $q_{\mathrm{s}}^{v''}$ are the locations of these two new nodes. For simplicity, $q_{\mathrm{s}}^{v'} = q^v$ and v'' is moved away from v' along a unit vector d_v so that there is no collision introduced after this process. The direction of d_v is determined by $q_{\mathrm{s}}^{v'}$ (the current location of v') and $\mathcal{N}_G(v')$ (the neighbors of v'), and can be derived by normalizing the following vector:

$$\sum_{\hat{v} \in \mathcal{N}_G(v')} \frac{q_{\mathrm{s}}^{v'} - q^{\hat{v}}}{\| q_{\mathrm{s}}^{v'} - q^{\hat{v}} \|} \tag{13.7}$$

and the distance moved along vector d_v can be tried iteratively until there is no collision in this VTT. In addition to the collision-free requirement, the resulted VTT should satisfy all other hardware constraints described in Section 13.4 after splitting, and let $V_C = \{v', v''\}$ when checking the motion manipulability by Eq. 13.4.

Given a location of node v and one way to split E^v into $E^{v'}$ and $E^{v''}$, Function Compute-SplitAction is used to check and compute this `Split` action. In this function, G is the given VTT, $a \in \mathcal{A}$ is a possible split way, and D, ΔD, and $D_{\max}$ are three parameters. Function

Function ComputeSplitAction(G, q^v, $a \in \mathcal{A}$)

Data: D, ΔD, $D_{\max}$

Move node v to q^v;
if `VTTValidation(`G`)` = FALSE **then**
  return NULL;
end
Split node v into v' and v'' according to $a \in \mathcal{A}$;
$q_{\mathrm{s}}^{v'} \leftarrow q^v$, $q_{\mathrm{s}}^{v''} \leftarrow q^v$;
Compute d_v;
repeat
  $q_{\mathrm{s}}^{v''} = q^v + Dd_v$;
  Move node v'' to $q^{v''}$;
  if `VTTCollisionCheck` *(*G*)* = TRUE **then**
    $D \leftarrow (D + \Delta D)d_v$;
  else
    if `VTTValidation(`G`)` = TRUE **then**
      return $(E^{v'}, E^{v''}, q_{\mathrm{s}}^{v'}, q_{\mathrm{s}}^{v''})$;
    else
      return NULL;
    end
  end
until $D \leq D_{\max}$;
return NULL;

`VTTValidation` returns true if a given VTT satisfies all hardware constraints, otherwise returns false. Function `VTTCollisionCheck` returns false if there is no collision among all members, otherwise returns true. The basic idea is to move v'' gradually away from v' until a valid VTT is found. The maximum distance between v' and v'' for searching is $D_{\max}$ that should be a small value. Within this small range, member collision is a more significant constraint, such as the case shown in Figure 13.11b. Hence initially only checking collision among members, and once a collision-free location for v'' is found, check other hardware constraints. Reversely, if merging v' and v'' as v at a location q^v, with this computed `Split` action $(E^{v'}, E^{v''}, q_{\mathrm{s}}^{v'}, q_{\mathrm{s}}^{v''})$, first check if $q_{\mathrm{s}}^{v'} = q^v \in \mathcal{C}_{\mathrm{free}}^{v'}(q^{v'})$ and $q_{\mathrm{s}}^{v''} \in \mathcal{C}_{\mathrm{free}}^{v''}(q^{v''})$, and if true, move v' to $q_{\mathrm{s}}^{v'}$ and move v'' to $q_{\mathrm{s}}^{v''}$, and then merge them at q^v.

Given q^v, the current position of node v, and $\mathcal{C}_{\mathrm{free}}^{v} = \{{}^t\mathcal{C}_{\mathrm{free}}^{v} | t = 1, 2, \cdots, T\}$ that contains T enclosed subspaces in the free space of node v, apply a valid `Split` action on this node to

separate E^v into $E^{v'}$ and $E^{v''}$ and generate two new nodes v' and v''. $q^{v'}$ and $q^{v''}$ are the locations of v' and v'', and $\mathcal{C}^{v'}_{\text{free}}(q^{v'})$ and $\mathcal{C}^{v''}_{\text{free}}(q^{v''})$ can be computed accordingly. Assuming there is a position $q \in {}^t\mathcal{C}^v_{\text{free}}$ where node v can also be split in the same way (E^v can be separated into $E^{v'}$ and $E^{v''}$ with `Split` action $(E^{v'}, E^{v''}, q_{\text{s}}^{v'}, q_{\text{s}}^{v''})$), if $q_{\text{s}}^{v'} \in \mathcal{C}^{v'}_{\text{free}}(q^{v'})$ and $q_{\text{s}}^{v''} \in \mathcal{C}^{v''}_{\text{free}}(q^{v''})$, namely v' and v'' can navigate to $q_{\text{s}}^{v'}$ and $q_{\text{s}}^{v''}$ respectively and merge at q, then this node v can navigate from $\mathcal{C}^v_{\text{free}}(q^v)$ to ${}^t\mathcal{C}^v_{\text{free}}$ by a pair of `Split` and `Merge` actions, and these two enclosed subspaces can be connected under this action pair. This is the transition model when applying topology reconfiguration actions. From Section 13.6.2, it is shown that the possible ways to split a node are highly dependent on the location of the node, and the resulted enclosed subspaces of the newly generated nodes can be very different when splitting the node at different locations. So multiple samples are needed for every enclosed subspace. When applying the transition model described above, it is necessary to compute $\mathcal{C}^{v'}_{\text{free}}(q^{v'})$ and $\mathcal{C}^{v''}_{\text{free}}(q^{v''})$ many times (for every sample and every valid split action) and can be time-consuming. An alternative is to regard v' and v'' as a group and compute their group free space $\widehat{\mathcal{C}}^{v'}_{\text{free}}(q^{v'})$ and $\widehat{\mathcal{C}}^{v''}_{\text{free}}(q^{v''})$ respectively for every possible way to split v in advance. If $q_{\text{s}}^{v'} \in \widehat{\mathcal{C}}^{v'}_{\text{free}}(q^{v'})$ and $q_{\text{s}}^{v''} \in \widehat{\mathcal{C}}^{v''}_{\text{free}}(q^{v''})$, then these two enclosed subspaces can be connected. This transition model based on the group free space can be much more efficient but may cause failures when applying the path planning for a group of nodes since it is less strict than the previous one.

There are two phases in the topology reconfiguration planning: *sample generation* and *graph search*. In *sample generation*, multiple samples are generated for every enclosed subspace. These samples are expected to provide valid `Split` actions as many as possible and also cover the space as much as possible. Algorithm 10 is introduced to sample a given enclosed subspace. $\mathcal{S}$ is the set containing all generated valid samples and $\mathcal{A}^{\mathcal{S}}$ stores the `Split` actions for every sample. $N_{\max}$ is the maximum number of samples for a given enclosed subspace ${}^t\mathcal{C}^v_{\text{free}}$ and it is determined by the size of the space: a larger space is expected to have more samples. In the current setup, it is easy to find the range of a given space along x-axis, y-axis, and z-axis denoted as x_{range}, y_{range}, and z_{range} respectively, then $N_{\max}$

Algorithm 10: Sample Generation

Input: ${}^{t}\mathcal{C}^{v}_{\text{free}}$, G, $\mathcal{A}$
Output: $\mathcal{S}$, $\mathcal{A}^{\mathcal{S}}$

$\mathcal{S} \leftarrow \emptyset$;
Initialize an empty map $\mathcal{A}^{\mathcal{S}}$;
Initialize $d_{\min}$;
$N_{\max} \leftarrow$ `SampleNumber`$({}^{t}\mathcal{C}^{v}_{\text{free}})$;
for $k = 1$ **to** K **do**
  $\mathcal{A}_{\text{valid}} \leftarrow \emptyset$;;
  $q_{\text{rand}} \leftarrow$ `RandomPosition`$({}^{t}\mathcal{C}^{v}_{\text{free}})$;
  foreach $a \in \mathcal{A}$ **do**
    $\hat{a} =$ `ComputeSplitAction`$(G,\ q,\ a)$;
    if $\hat{a}$ *is not* NULL **then**
      $\mathcal{A}_{\text{valid}} = \mathcal{A}_{\text{valid}} \cup \{\hat{a}\}$;
    end
  end
  $\mathcal{S}_{\text{close}} \leftarrow \{q | q \in \mathcal{S} \wedge \|q - q_{\text{rand}}\| \leq d_{\min}\}$;
  if $\mathcal{S}_{\text{close}} = \emptyset \wedge |\mathcal{S}| < N_{\max}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{q_{\text{rand}}\}$;
    $\mathcal{A}^{\mathcal{S}}[q] = \mathcal{A}_{\text{valid}}$;
  else if $\mathcal{S}_{\text{close}} \neq \emptyset \wedge |\mathcal{S}| < N_{\max}$ **then**
    if $|\mathcal{A}_{\text{valid}}| = |\mathcal{A}|$ **then**
      foreach $q \in \mathcal{S}_{\text{close}}$ **do**
        del $\mathcal{A}^{\mathcal{S}}[q]$;
      end
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \mathcal{S}_{\text{close}} + \{q_{\text{rand}}\}$;
      $\mathcal{A}^{\mathcal{S}}[q_{\text{rand}}] = \mathcal{A}_{\text{valid}}$;
    else
      UpdateFlag $\leftarrow$ FALSE;
      foreach $q \in \mathcal{S}_{\text{close}}$ **do**
        if $\mathcal{A}^{\mathcal{S}}[q] \subset \mathcal{A}_{\text{valid}}$ **then**
          del $\mathcal{A}^{\mathcal{S}}[q]$;
          $\mathcal{S} \leftarrow \mathcal{S} \setminus \{q\}$;
          $N_{\max} \leftarrow N_{\max} - 1$;
          UpdateFlag $\leftarrow$ TRUE;
        else if $\mathcal{A}_{\text{valid}} \setminus \mathcal{A}^{\mathcal{S}}[q] \neq \emptyset$ **then**
          UpdateFlag $\leftarrow$ TRUE;
        end
      end
      if UpdateFlag $=$ TRUE **then**
        $\mathcal{S} \leftarrow \mathcal{S} + \{q\}$;
        $\mathcal{A}^{\mathcal{S}}[q_{\text{rand}}] = \mathcal{A}_{\text{valid}}$;
        $N_{\max} \leftarrow N_{\max} + 1$;
      end
    end
  end
end

is simply $\left\lceil \sqrt{x_{\text{range}}^2 + y_{\text{range}}^2 + z_{\text{range}}^2} \Big/ d_{\min} \right\rceil$ in which $d_{\min}$ is the minimum distance between every pair of samples. K is the maximum number of iterations set by users and can be related to $N_{\max}$. For each iteration, first randomly generate a sample q_{rand} that is inside the given enclosed subspace, then find all valid `Split` actions stored in $\mathcal{A}_{\text{valid}}$ (*Line 6 — 13*). Then find all previous valid samples that are close to q_{rand} (*Line 14*). If q_{rand} is far from all previous valid samples and the size of $\mathcal{S}$ is less than $N_{\max}$, add this new sample and store its valid `Split` actions (*Line 15—17*). If q_{rand} is close to some previous valid samples, there are two cases for consideration. If this new sample provides all possible `Split` actions, then remove all valid nearby samples and only keep this new sample for this area since this is the best option (*Line 19—24*). Otherwise, check two conditions: 1. whether there exists any valid nearby sample that has fewer number of valid `Split` actions than q_{rand}; 2. whether the new sample q_{rand} introduces any knew `Split` actions compared with all valid nearby samples. If the first condition is true, remove the old sample and add the new sample to $\mathcal{S}$. If the second condition is true, add the new sample to $\mathcal{S}$ (*Line: 26—41*). Run Algorithm 10 for every enclosed subspace to generate enough samples, and then add q_i^v (the initial location of node v) to the sample set of $\mathcal{C}_{\text{free}}^v(q_i^v)$ and add q_g^v (the goal location of node v) to the sample set of $\mathcal{C}_{\text{free}}^v(q_g^v)$. After the *sample generation* phase, ${}^{\mathcal{C}}\mathcal{S}$ and the corresponding `Split` actions for all samples ${}^{\mathcal{C}}\mathcal{A}^{\mathcal{S}}$ are obtained for every enclosed subspace $\mathcal{C} \in \mathcal{C}_{\text{free}}^v$.

Then enter the *graph search* phase to generate a sequence of topology reconfiguration actions. With the transition model discussed before, a graph search algorithm can be applied to compute a sequence of enclosed subspaces starting from $\mathcal{C}_{\text{free}}^v(q_i^v)$ to $\mathcal{C}_{\text{free}}^v(q_g^v)$ while exploring the topology connections among these enclosed subspaces. An example is shown in Figure 13.12. Here, the graph has enclosed subspaces as vertices. An edge in this graph connecting two enclosed subspaces denotes that node v can move from a sample in one enclosed subspace to a sample in the other. The graph is built from $\mathcal{C}_{\text{free}}^v(q^v)$, grows as valid transitions among enclosed subspaces are found, and stops when the enclosed subspace containing q_g^v is visited. A graph search algorithm designed based on Dijkstra's framework is shown in Algorithm 11.

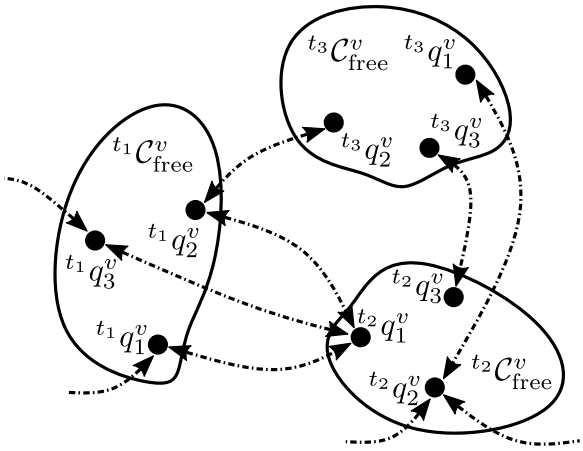


Figure 13.12: Topology connections among three enclosed subspaces of node v.

Line 1 — 5: If q_i^v and q_g^v are in the same enclosed subspace, then no topology reconfiguration is needed. Otherwise, make two sets $\mathcal{Q}$ and $\overline{\mathcal{Q}}$ where $\mathcal{Q}$ contains all newly checked or non-visited enclosed subspaces and $\overline{\mathcal{Q}}$ contains all visited enclosed subspaces. The size of these two sets will change as the algorithm explores $\mathcal{C}^v_{\text{free}}$. Initially, only the enclosed subspace containing q_i^v that is $\mathcal{C}^v_{\text{free}}(q_i^v)$ and the enclosed subspace containing q_g^v that is $\mathcal{C}^v_{\text{free}}(q_g^v)$ are in $\mathcal{Q}$, and the algorithm starts with $\mathcal{C}^v_{\text{free}}(q_i^v)$. The value $g(\mathcal{C})$ is the cost of the path from q_i^g to the enclosed subspace $\mathcal{C}$, so $g(\mathcal{C}^v_{\text{free}}(q_i^v)) = 0$ and $g(\mathcal{C}^v_{\text{free}}(q_g^v)) = \infty$ at the beginning.

Line 7 — 9: Every iteration starts with the enclosed subspace that has the lowest cost $g(\mathcal{C})$ in $\mathcal{Q}$. At the beginning, $\mathcal{C}^v_{\text{free}}(q_i^v)$ has the lowest cost. After selecting an enclosed subspace, update $\mathcal{Q}$ and $\overline{\mathcal{Q}}$.

Line 10 — 26: Iterate every enclosed subspace $\mathcal{C}$ except $\overline{\mathcal{C}}$ in $\mathcal{C}^v_{\text{free}}$ and check if it is already visited. If so, then this potential transition is not a new transition. Otherwise, check if there exists a valid motion to move the node from any sample in $\overline{\mathcal{C}}$ that is the enclosed subspace with the lowest cost to any sample in $\mathcal{C}$ handled by Function `ValidMotion`. Both transition models are implemented and compared in the test scenarios. If this is true, then there are two cases: this subspace is not checked for the first time namely that there is already a connection between this enclosed subspace and another enclosed subspace, or this subspace has never been checked which means it has no connection before. For the first case, it is necessary to further check whether its cost needs to be updated. $c(^{\overline{\mathcal{C}}}q, ^{\mathcal{C}}q)$ is the cost of the

Algorithm 11: Topology Reconfiguration Planning

Input: G, q_i^v, q_g^v, $\mathcal{C}_{\text{free}}^v$, $\{({}^{\mathcal{C}}\mathcal{S}, {}^{\mathcal{C}}\mathcal{A}^{\mathcal{S}}) | \mathcal{C} \in \mathcal{C}_{\text{free}}^v\}$

Output: Tree of enclosed subspaces p

```
if C^v_free(q^v_i) = C^v_free(q^v_g) then
    return NULL;
end
Q ← {C^v_free(q^v_i), C^v_free(q^v_g)}, Q̄ ← ∅;
g(C^v_free(q^v_i)) ← 0, g(C^v_free(q^v_g)) ← ∞;
while C^v_free(q^v_g) ∈ Q do
    C̄ ← arg min_{C∈Q} g(C);
    Q ← Q \ {C̄};
    Q̄ ← Q̄ ∪ {C̄};
    foreach C ∈ C^v_free \ {C̄} ∧ C ∉ Q̄ do
        foreach (C̄q, Cq) ∈ C̄S × CS do
            if ValidMotion(C̄q, Cq) = TRUE then
                if C ∈ Q then
                    if g(C̄) + c(C̄q, Cq) < g(C) then
                        g(C) ← g(C̄) + c(C̄q, Cq);
                        p(C) ← C̄;
                    end
                else
                    Q ← Q + {C};
                    g(C) ← g(C̄) + c(C̄q, Cq);
                    p(C) ← C̄;
                end
            end
            break;
        end
    end
end
```

motion from ${}^{\overline{\mathcal{C}}}q$ to ${}^{\mathcal{C}}q$. This cost can be related to the estimated distance or other factors. In the current setup, all valid motions from one enclosed subspace to another one have the same cost. If its cost is updated, then its parent $p(\mathcal{C})$ should also be updated accordingly. For the second case, initialize the cost and the parent of this newly checked enclosed subspace, and update set $\mathcal{Q}$. There can be multiple ways to transit between two enclosed subspaces. For example, there are two edges between ${}^{t_2}\mathcal{C}_{\text{free}}^v$ and ${}^{t_3}\mathcal{C}_{\text{free}}^v$ shown in Figure 13.12. In the current setup, only the first one being found is saved. Since ${}^{\overline{\mathcal{C}}}\mathcal{S}$ and ${}^{\mathcal{C}}\mathcal{S}$ are randomly generated, it

is possible that the condition in *Line 12* is failed although there does exist ${}^{\overline{\mathcal{C}}}q$ and ${}^{\mathcal{C}}q$ that can pass this condition.

Once $\mathcal{C}^v_{\text{free}}(q^v_g)$ is visited, the algorithm ends. With p, a tree with visited enclosed subspaces as vertices, it is straightforward to find the optimal path connecting $\mathcal{C}^v_{\text{free}}(q^v_i)$ and $\mathcal{C}^v_{\text{free}}(q^v_g)$ as well as all the samples the node need to traverse. For example, in Figure 13.12, when node v traverses from ${}^{t_1}q^v_1$ to ${}^{t_3}q^v_1$, it first moves to ${}^{t_1}q^v_2$, then `Split` and `Merge` at ${}^{t_3}q^v_2$, and finally move to ${}^{t_3}q^v_1$. Moving a node inside one of its enclosed subspaces can be solved easily by geometry reconfiguration planning. After splitting the node into a pair, geometry reconfiguration planning can also be applied to move them to the computed positions in the next enclosed subspace for merging.

13.7 Test Scenarios

The motion planning framework is implemented in C++. Three example scenarios were conducted to measure the effectiveness of the approach. The performance of the framework is compared with the approach from [52] and Section 11.2. These experiments are also tested under different constraints and with different parameters to show the universality of the presented framework. All tests run on a laptop computer (Intel Core i7-8750H CPU, 16GB RAM) and the workspace is a cuboid.

13.7.1 Geometry Reconfiguration

The geometry reconfiguration planning test changes the cube shape of a VTT, Figure 12.10a, to a tower shape, Figure 12.10b. Recall that this VTT is composed of 21 members with initial positions of nodes listed in the following:

$$
\begin{aligned}
q^{v_0} &= [-1.605, -0.771, 2.075]^\intercal & q^{v_1} &= [0.7779, -0.7642, 2.075]^\intercal \\
q^{v_2} &= [-0.4756, -2.022, 0.075]^\intercal & q^{v_3} &= [-0.4142, 0.4228, 2.175]^\intercal \\
q^{v_4} &= [-1.605, -0.771, 0.075]^\intercal & q^{v_5} &= [0.3819, -0.3707, 0.125]^\intercal \\
q^{v_6} &= [-0.4314, -0.9559, 1.2321]^\intercal & q^{v_7} &= [-0.4756, -2.022, 2.075]^\intercal \\
q^{v_8} &= [0.1819, -0.1707, 0.075]^\intercal
\end{aligned}
$$

The constraints for this task are $\overline{L}_{\min} = 1.0\,\text{m}$, $\overline{L}_{\max} = 3.5\,\text{m}$, $\bar{\theta}_{\min} = 0.3\,\text{rad}$, and $\bar{\mu}_{\min} = 0.1$.

Four nodes v_1, v_3, v_5, and v_6 are involved in this motion task and their goal positions in the tower VTT are $q_g^{v_1} = [0.1819, -0.1707, 4.125]^\intercal$, $q_g^{v_3} = [-1.605, -0.771, 4.075]^\intercal$, $q_g^{v_5} = [-0.4756, -2.022, 4.075]^\intercal$, and $q_g^{v_6} = [0.1819, -0.1707, 2.125]^\intercal$. These four nodes are separated into two groups $\{v_3, v_5\}$ and $\{v_1, v_6\}$ which is randomly selected. First compute $\widehat{\mathcal{C}}_{\text{free}}^{v_3}(q_i^{v_3})$ and $\widehat{\mathcal{C}}_{\text{free}}^{v_5}(q_i^{v_5})$, and then do planning for these two nodes. The motions of node v_3 and v_5 is shown in Figure 13.13. Most of the obstacle region in this step is surrounded by the subspace $\widehat{\mathcal{C}}_{\text{free}}^{v_3}(q_i^{v_3})$ and $\widehat{\mathcal{C}}_{\text{free}}^{v_5}(q_i^{v_5})$, hence it is easier for them to extend outward first in order to navigate to the goal positions. This motion process moves the projected center of mass toward one edge of the support polygon but the planner can constrain the projected center of mass within the support polygon. After planning for v_3 and v_5, the truss is updated, and $\widehat{\mathcal{C}}_{\text{free}}^{v_1}(q_i^{v_1})$ and $\widehat{\mathcal{C}}_{\text{free}}^{v_6}(q_i^{v_6})$ are computed accordingly. Finally the planning for v_1 and v_6 finishes this motion task with the result shown in Figure 13.14. For v_1 and v_6 in this updated truss,

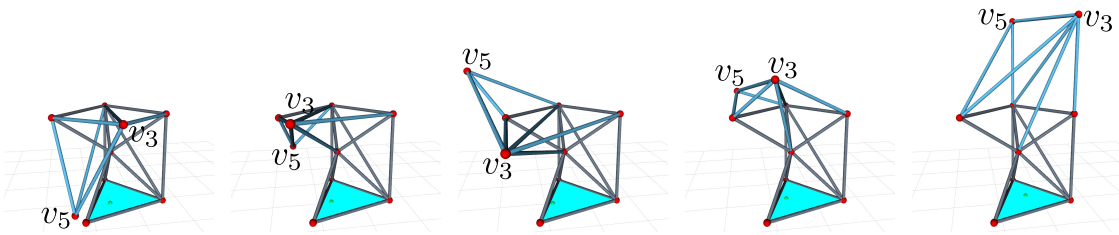


Figure 13.13: v_3 and v_5 firstly extend outward, and then move upward to their goal positions. The support polygon is formed by three nodes (v_2, v_4, v_5) on the ground shown as the aqua region (▲) and the green dot (●) is the center of mass represented on the ground.

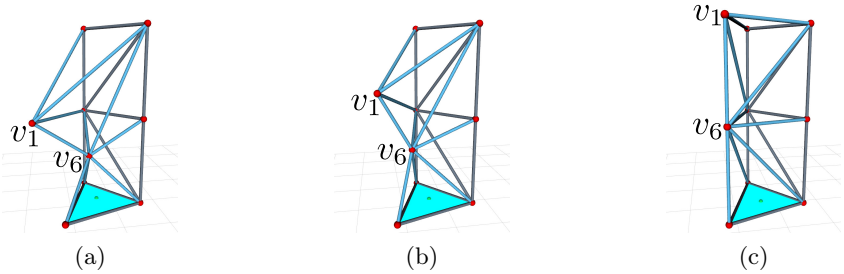


Figure 13.14: v_1 and v_6 can navigate to their goal positions easily since $\widehat{\mathcal{C}}_{\text{free}}^{v_1}(q_i^{v_1})$ and $\widehat{\mathcal{C}}_{\text{free}}^{v_6}(q_i^{v_6})$ almost cover the whole workspace.

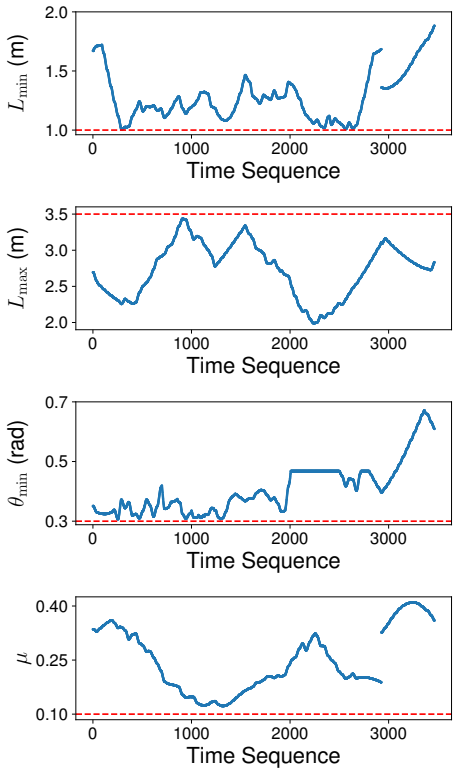


Figure 13.15: The minimum length ($L_{\min}$) and the maximum length ($L_{\max}$) of all moving edge modules, the minimum angle between every pair of edge modules ($\theta_{\min}$), and the motion manipulability (μ) are measured throughout the geometry reconfiguration process in Figure 13.13 and Figure 13.14.

the enclosed subspace $\widehat{\mathcal{C}}^{v_1}_{\text{free}}(q_i^{v_1})$ and $\widehat{\mathcal{C}}^{v_6}_{\text{free}}(q_i^{v_6})$ almost covers the whole workspace so it is also easy for them to navigate to the goal positions. The minimum length ($L_{\min}$) and the maximum length ($L_{\max}$) of all moving edge modules, the minimum angle between every pair of edge modules ($\theta_{\min}$), and the motion manipulability (μ) are shown in Figure 13.15. Note that $L_{\min}$, $L_{\max}$, and μ are not necessarily to be continuous because nodes that are under control are changing. This motion task is also demonstrated by [52] using the retraction-based RRT algorithm. This algorithm cannot solve this motion planning task in 100 trials unless an intermediate waypoint is manually specified to mitigate the narrow passage issue. The success rate for the planning from the initial to the waypoint is 99% and 98% from the waypoint to the goal. In comparison, the presented algorithm doesn't need any additional waypoints, and for 1000 trials, the mean running time is 4.294 s with a standard deviation of

1.992 s and the success rate is 100%. In these trials, the maximum planning time is 7.556 s and the minimum is 1.014 s.

13.7.2 Topology Reconfiguration

Scenario 1

The VTT configuration used for this topology reconfiguration example is similar to the configuration shown in Figure 11.6 with the following nodes' positions:

$$
\begin{aligned}
q^{v_0} &= [0.05, 0, 0.075]^\intercal \quad & q^{v_1} &= [0.1, 1.8, 0.075]^\intercal \\
q^{v_2} &= [2.1, 1.9, 0.075]^\intercal \quad & q^{v_3} &= [2.1, 0, 0.075]^\intercal \\
q^{v_4} &= [0, 2.1, 3.225]^\intercal \quad & q^{v_5} &= [1.95, 0.9, 3]^\intercal \\
q^{v_6} &= [0, 0, 3.025]^\intercal
\end{aligned}
$$

The constraints for this task are $\overline{L}_{\min} = 1.0\,\text{m}$, $\overline{L}_{\max} = 5.0\,\text{m}$, $\bar{\theta}_{\min} = 0.2\,\text{rad}$, and $\bar{\mu}_{\min} = 0.1$.

The motion task is to move v_5 from its initial position $q_i^{v_5} = [1.95, 0.9, 3]^\intercal$ to a goal position $q_g^{v_5} = [1, 1.2, 0.9]^\intercal$ (Figure 11.9). This motion cannot be executed with only geometry reconfiguration because $\mathcal{C}_{\text{free}}^{v_5}(q_i^{v_5})$ and $\mathcal{C}_{\text{free}}^{v_5}(q_g^{v_5})$ shown in Figure 13.16 are separated by the

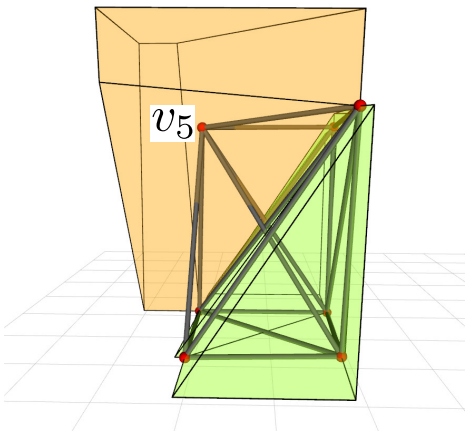


Figure 13.16: $\mathcal{C}_{\text{free}}^{v_5}(q_i^{v_5})$ is the yellow space on the upper left and $\mathcal{C}_{\text{free}}^{v_5}(q_g^{v_5})$ is the green space on the lower right. They are not connected and separated by the obstacle region generated from edge (v_3, v_4).

obstacle region generated from edge module (v_3, v_4). For this task, the minimum distance between two nodes is $d_{\min} = 1.0\,\text{m}$ and $N_{\max}$ is constrained to be greater than or equal to 3, namely it is expected to have at least 3 samples for every enclosed subspace, and the maximum number of iterations $K = 5N_{\max}$.

With the topology reconfiguration planning algorithm (Algorithm 11), one pair of `Split` and `Merge` actions is sufficient. v_5 is moved to a new location, then split into a pair of nodes (v_5' and v_5'') so that both of these two newly generated nodes can navigate to $\mathcal{C}_{\text{free}}^{v_5}(q_g^{v_5})$ and merge into a single node. Then the geometry motion planning is used to plan the motions of v_5' and v_5'' and control them to the target positions for merging. Finally, merge them back to an individual node and then move the node to $q_g^{v_5}$. The detailed process is shown in Figure 13.17. The minimum length ($L_{\min}$) and the maximum length ($L_{\max}$) of all moving edge modules, the minimum angle between every pair of edge modules ($\theta_{\min}$), and the motion

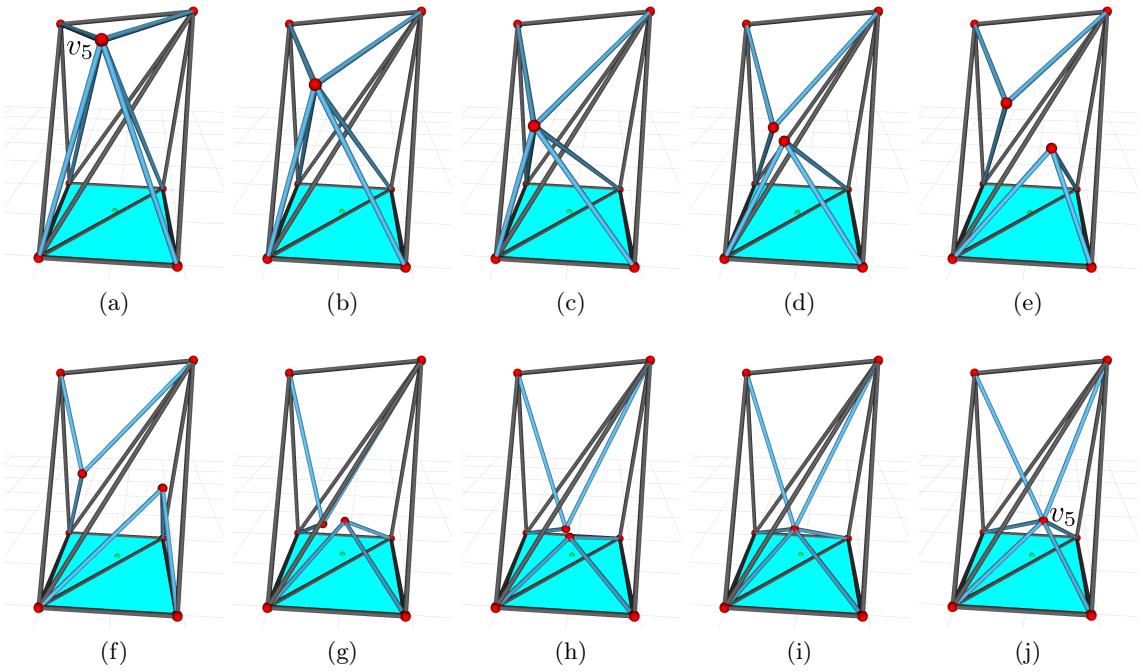


Figure 13.17: The sequence to move v_5 from $q_i^{v_5}$ to $q_g^{v_5}$ is shown. The support polygon is the aqua region (■) and the green dot (●) is the center of mass represented on the ground. (a) — (c) First move v_5 to a new location. (d) Split v_5 into a pair. (e) — (h) Move these two newly generated nodes in different directions to go around the edge module (v_3, v_4). (i) — (j) Merge them into an individual node and move this node to $q_g^{v_5}$.

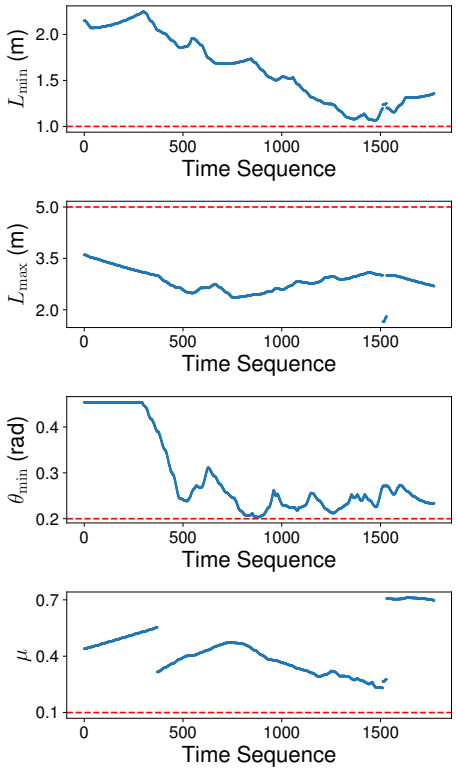


Figure 13.18: The minimum length ($L_{\min}$) and the maximum length ($L_{\max}$) of all moving edge modules, the minimum angle between every pair of edge modules ($\theta_{\min}$), and the motion manipulability (μ) are measured throughout the topology reconfiguration process in Figure 13.17.

manipulability (μ) are shown in Figure 13.18.

This motion task has been solved in Section 11 with the graph search algorithm exploring 8146 VTT configurations with a more complex action model in order to find a valid sequence of motion actions. With the proposed framework, the free space of v_5 is partitioned into 53 enclosed subspaces and it takes on average 56.614 s to solve this motion task when using the first transition model with a standard deviation of 5.654 s in 1000 trials, including the enclosed subspace computation, topology reconfiguration planning, and two-node geometry reconfiguration planning, and the success rate is 100%. In these trials, the maximum planning time is 77.880 s and the minimum is 38.973 s. With the transition model based on the group free space, the average planning time can be as fast as 20 s.

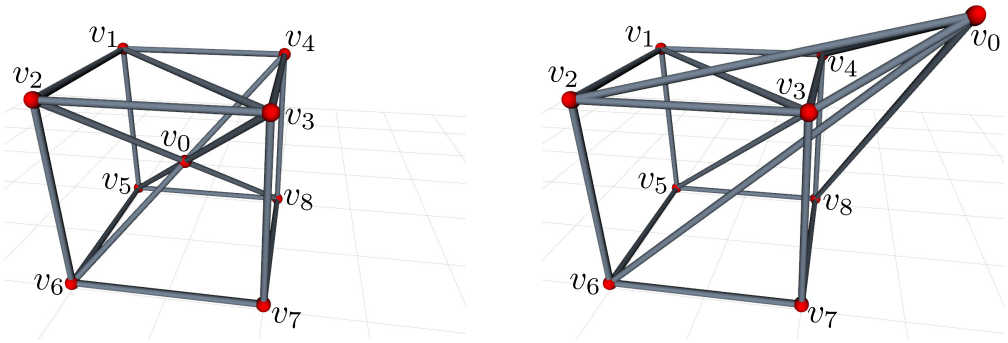


Figure 13.19: (a) A VTT is constructed from 19 members with 9 nodes. (b) The task is to move v_0 from its initial position to a position outside the cubic truss.

Scenario 2

Another motion task in which topology reconfiguration actions are involved is shown in Figure 13.19 that is to move v_0 from a position $q_i^{v_0}$ inside the cubic truss (Figure 13.19a) to a new position $q_g^{v_0}$ (Figure 13.19b). The initial positions of all nodes are

$$
\begin{aligned}
q^{v_0} &= [0, 0, 1.075]^\intercal & q^{v_1} &= [1, 1, 0.075]^\intercal \\
q^{v_2} &= [-1, 1, 2.075]^\intercal & q^{v_3} &= [-1, -1, 2.075]^\intercal \\
q^{v_4} &= [1, -1, 2.075]^\intercal & q^{v_5} &= [1, 1, 0.075]^\intercal \\
q^{v_6} &= [-1, 1, 0.075]^\intercal & q^{v_7} &= [-1, -1, 0.075]^\intercal \\
q^{v_8} &= [1, -1, 0.075]^\intercal
\end{aligned}
$$

The constraints for this task are $\overline{L}_{\min} = 0.5\,\text{m}$, $\overline{L}_{\max} = 5.0\,\text{m}$, $\bar{\theta}_{\min} = 0.15\,\text{rad}$, and $\bar{\mu}_{\min} = 0.1$.

In this task, $q_g^{v_0} = [-0.64, -2.19, 2.78]^\intercal$. Similarly, $q_i^{v_0}$ and $q_g^{v_0}$ are in two separated enclosed subspaces, and one solution is to apply topology reconfiguration actions twice to traverse three enclosed subspaces in $\mathcal{C}_{\text{free}}^v$ shown in Figure 13.20. For this task, $d_{\min} = 0.5$, $N_{\max}$ is constrained to be greater than or equal to 3, and $K = 8N_{\max}$.

The detailed planning result is shown in Figure 13.21. The planner first moves the node v_0 to a location outside the cubic truss (Figure 13.21a — Figure 13.21c), and then

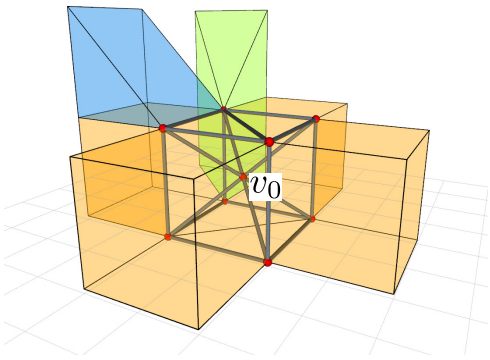


Figure 13.20: v has to move from $\mathcal{C}^v_{\text{free}}(q^v_i)$ that is the yellow enclosed subspace to the green enclosed subspace, and then $\mathcal{C}^v_{\text{free}}(q^v_g)$ that is the blue enclosed subspace.

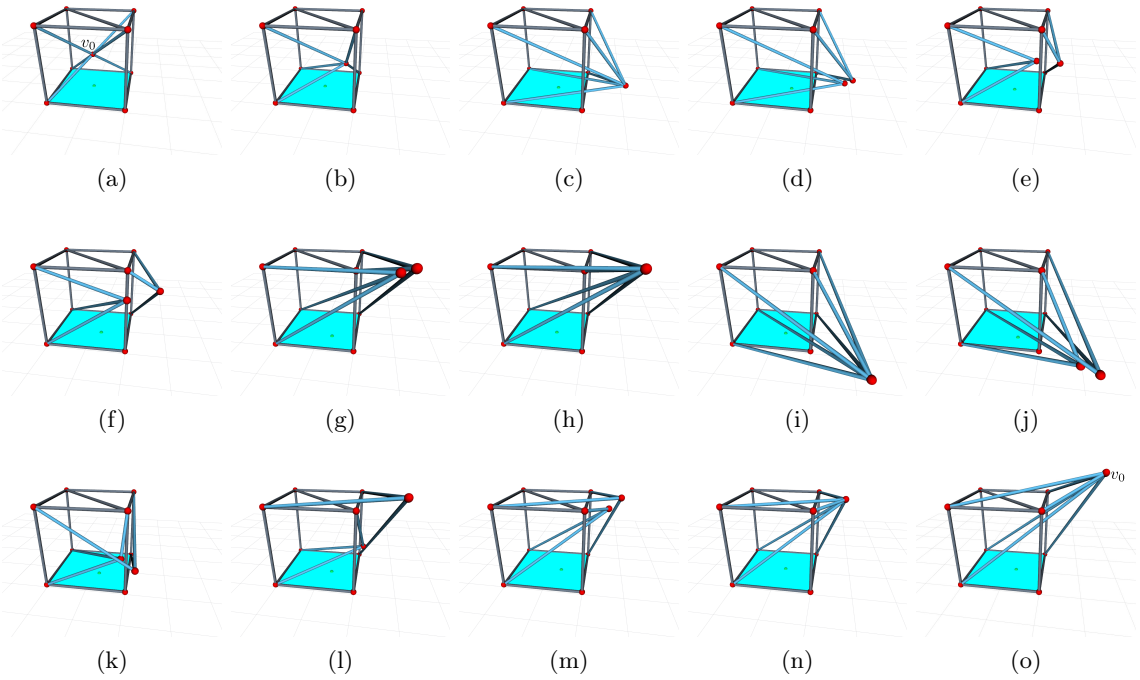


Figure 13.21: The sequence to move v_0 from $q^{v_0}_i$ to $q^{v_0}_g$ by traversing three enclosed subspaces in $\mathcal{C}^v_{\text{free}}$ is shown. The support polygon is the aqua region (■) and the green dot (●) is the center of mass represented on the ground. (a) — (c) Move v_0 to traverse the plane formed by v_3, v_4, v_7, and v_8. (d) Split v_0 into a pair here so that both newly generated nodes can move around edge module (v_3, v_7) while avoiding singular configuration. (e) — (h) Two newly generated nodes are moved to a location inside the green enclosed subspace and merge. (i) — (j) Move the merged node to a new location and split it in a different way to generate two new nodes. (k) — (n) One node traverses the space inside the cubic truss to go to the blue enclosed subspace, and the other node moves upward. Then these two nodes merge at a location inside the blue enclosed subspace. (o) Move the node to the target location.

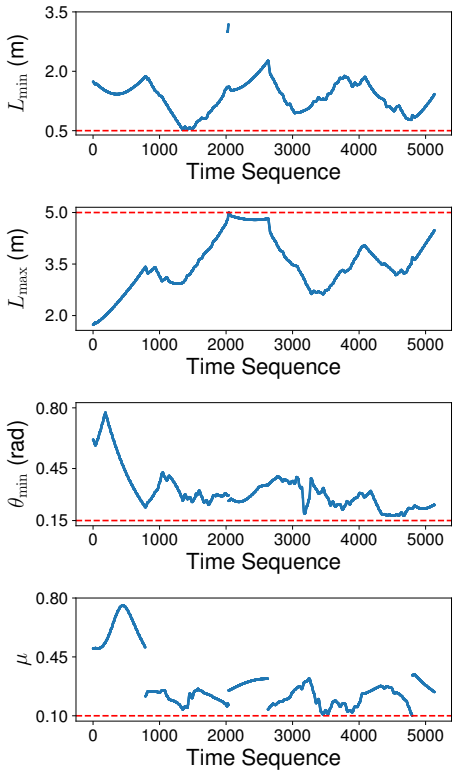


Figure 13.22: The minimum length ($L_{\min}$) and the maximum length ($L_{\max}$) of all moving edge modules, the minimum angle between every pair of edge modules ($\theta_{\min}$), and the motion manipulability (μ) are measured throughout the topology reconfiguration process in Figure 13.21.

split it into two. If the node is split inside the cubic truss, there is no way to merge them inside the green enclosed subspace (Figure 13.21h) because one node has to traverse a region formed by node v_3, v_4, v_7, and v_8 resulting in singular configurations. After this `Split` action, do geometry reconfiguration planning to control them to the green enclosed subspace (Figure 13.21d — Figure 13.21g) and merge them back. Then move the node to a different location inside the green enclosed subspace (Figure 13.21i) and then split the node in a different way (Figure 13.21j) in order to navigate these two nodes to $\mathcal{C}^v_{\text{free}}(q_g^{v_0})$ (Figure 13.21k — Figure 13.21m) and merge them back (Figure 13.21n). Eventually move the node to $q_g^{v_0}$ (Figure 13.21o). The minimum length ($L_{\min}$) and the maximum length ($L_{\max}$) of all moving edge modules, the minimum angle between every pair of edge modules ($\theta_{\min}$), and the motion manipulability (μ) are shown in Figure 13.22. Using the strict transition model,

the average planning time is 319.825 s with a standard deviation being 21.312 s in 1000 trials, and the success rate is 74%. In these trials, the maximum planning time is 392.466 s and the minimum is 172.051 s. The search space is larger and more samples are generated in order to find the sequence of topology reconfiguration actions which consumes more time. All the failures are from the graph search phase, and more samples can solve this issue but also make the search process more time consuming. With the same parameters, when using the other transition model based on the group free space, the average planning time is 86.482 s with a standard deviation being 2.048 s in 1000 trials, and the success rate is 95.9%. The maximum planning time is 92.853 s and the minimum planning time is 79.978 s. The performance is improved a lot. All the failures in these trials are from the path planning for a group of node, namely even the transition model between two enclosed subspace is valid, but it is not guaranteed for the planner to find the solution.

13.8 Conclusion

A reconfiguration planning framework for VTT robots is presented in this chapter. The configuration space of a node is studied considering the physical size of the robot components and singularity avoidance constraints. Geometry reconfiguration planning involves the motions of multiple nodes which are strongly coupled. An efficient algorithm to compute the obstacle regions and free space of a group is developed so that RRT can be applied to solve the geometry reconfiguration planning problem easily with all physical constraints considered. A fast algorithm to compute all enclosed subspaces in the free space of a node is presented so that whether topology reconfiguration actions are needed can be verified. A sample generation method is introduced to generate samples that can efficiently provide valid topology reconfiguration actions over a wide space. With the topology reconfiguration planning algorithm based on Dijkstra's algorithm, a sequence of topology reconfiguration actions can then be computed with geometry reconfiguration planning for a group of nodes, and the motion tasks requiring topology reconfiguration can then be solved efficiently.

Chapter 14

Locomotion

This chapter presents an approach for VTTs to perform locomotion tasks which can also be extended to other truss robots. This locomotion planner allows a VTT to navigate in an environment without receiving impacts from the ground. This chapter excerpts heavily from [83].

Locomotion capability is important for VTTs to perform tasks. While a VTT is locomoting, it is crucial to avoid impact from the environment in order to protect the mechanical components of the robot. This chapter presents a non-impact rolling locomotion planner for an arbitrary VTT by making use of the geometry reconfiguration actions. During the locomotion process, the motions of nodes are computed automatically so that a VTT first expands its support polygon and then tumbles in a stable manner. Some experiments are shown in the end.

14.1 Introduction

Self-reconfiguration enables a VTT to be better suited for a variety of tasks by dramatically changing its shape and topology and locomotion is one important task in many scenarios. For example, in a search and rescue mission, a VTT may need to flexibly locomote over some terrain to reach a destination and then reconfigure into a tower for shoring. The locomotion process can be achieved by moving nodes sequentially that can interact with the environment. One way to move would be quasi-statically to reduce potentially damaging

impacts with the ground [107].

The VTT locomotion process is similar to a locomotion mode of some VGTs which is accomplished by tipping and contacting the ground. Lee and Sanderson [68] simulated the rolling locomotion for TETROBOT systems with generated paths for moving nodes. Abrahantes *et al.* [1] designed the gait for a cubic truss. These works divide the locomotion gait into several steps, but they have to compute the motion of nodes beforehand which cannot be applied to arbitrary configurations. Optimization approaches have been used for locomotion planning. Usevitch *et al.* [148] formulates the locomotion process as a quadratic program by constraining the motion of the center of mass. The objective function is related to the velocity of each node. A more complete quadratic programming approach to locomotion is presented in [149] to generate discrete motions of nodes in order to follow a given trajectory or compute a complete gait cycle. More hardware constraints were considered, including length and collision avoidance. However, the approach has to solve an optimization problem in high dimensional space, and it also has to deal with non-convex and nonlinear constraints which may cause numerical issues and is limited to a fully connected five-node graph in order to avoid incorporating the manipulability constraints into the quadratic program. Park *et al.* [107] extends this optimization-based approach by preventing a robot from receiving impacts from the ground. Incorporated with a polygon-based random tree search algorithm to output a sequence of supporting polygons in [106], a VTT can execute a locomotion task in an environment. However, in these quadratic programs, numerical differentiation is required to relate some physical constraints with these optimized parameters whenever solving the problem. A locomotion step has to be divided into multiple phases leading to more constraints. Also, these approaches are not guaranteed to provide feasible solutions and they are also time-consuming to solve.

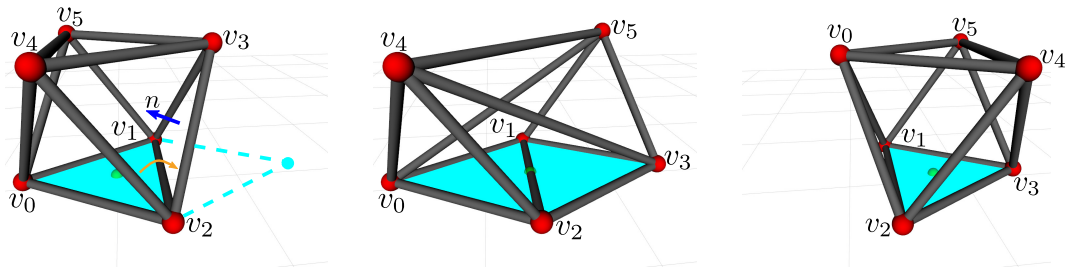
This chapter presents a new locomotion planning solution based on the efficient geometry reconfiguration planning algorithm. The solution can solve the problem much faster and more reliably under several hardware constraints compared with previous works. In addition, this approach can be applied to any arbitrary VTT. The shape of a VTT $G = (V, E)$ can be

regarded as a polyhedron with flat polygonal facets formed by members. This polyhedron consists of vertices (a subset of V), edges E^G, facets F^G, and an incidence relation on them (e.g. every edge is incident to two vertices and every edge is incident to two facets [60]). One of the facets $f \in F^G$ is the current support polygon. In a rolling locomotion step, the support polygon is changed from f to an adjacent facet f'. As mentioned, impact from the ground can damage the robot, so the stability criterion has to be maintained during the whole process. The locomotion problem for a given VTT $G = (V, E)$ can be stated as the following:

- **Non-impact Rolling Locomotion**. Compute the motions of a set of nodes in V to change the support polygon from $f \in F^G$ to a desired adjacent facet $f' \in F^G$ without receiving impacts from the ground while satisfying all constraints.

14.2 Locomotion

Unlike the previous reconfiguration planning, the center of mass during locomotion moves over a large range by continuously interacting with the environment. In a rolling locomotion step, a VTT rolls from one support polygon to an adjacent support polygon (Figure 14.1a).



(a) (b) (c)

Figure 14.1: A VTT in octahedron configuration executes a single rolling locomotion step. (a) Initially node v_0, v_1, and v_2 forms the support polygon shown as the aqua region (▲), and the center of mass projected onto the ground (●) is within this support polygon. The truss wants to roll from its current support polygon to an adjacent support polygon formed by node v_1, v_2, and the new tipping location. (b) v_3 and v_5 are moved so that the support polygon is expanded and the center of mass projected onto the ground is on member (v_1, v_2). (c) v_0 and v_4 are moved to their destinations to finish this locomotion step, and the center of mass projected onto the ground is within the new support polygon formed by node v_1, v_2, and v_3.

In this process, it is useful to maintain statically stable locomotion to prevent the system from receiving impacts from the ground as repeated impacts may damage the reconfiguration nodes. Due to this requirement, a rolling locomotion step has to be manually divided into several phases to have control over the center of mass [106, 149]. The approach presented here can deal with this issue automatically while planning the motion of nodes that need to be moved. A high-level path planner can generate a support polygon trajectory given an environment and a locomotion task, and the locomotion planner can ensure that the truss can follow this support polygon trajectory without violating constraints and receiving impacts from the ground.

14.2.1 Truss Polyhedron

The boundary representation of a VTT is modeled as a convex polyhedron which can be either defined initially or obtained from the set of node positions by computing the convex hull of them using existing computational geometry algorithms, such as Quickhull [6]. Given a VTT $G = (V, E)$, its polyhedron representation can be fully defined as $P^G = (E^G, F^G)$ in which F^G is the set of facets forming the boundary of this VTT and E^G is the set of edges forming all facets, namely each facet is composed of multiple edge members in E^G and each edge of E^G is incident to two facets. For example, the boundary representation of the VTT in Figure 14.1a is an octahedron in which edge (v_1, v_2) is incident to facet (v_1, v_2, v_3) and facet (v_0, v_1, v_2). In this example, all edges are contained in its polyhedron representation. However, for the VTT in Figure 13.7, v_0 and its corresponding edges in E^{v_0} are not involved in its polyhedron representation.

Suppose the set of edges forming a facet $f \in F^G$ is $E_f^G \subset E^G$, and $\forall e \in E_f^G$, the other incident facet of edge e is $f^e \in F^G$. Every facet can be a support polygon as long as all of its vertices are on the ground. Assume $f \in F^G$ is the current support polygon and a single rolling locomotion step is equivalent to rotating the truss with respect to an edge of this facet $e \in E_f^G$ until the other incident facet of this edge f^e becomes the new support polygon. An example of this process is shown in Figure 14.1. Initially facet $f = (v_0, v_1, v_2)$ is the support polygon, and the other incident facet of edge $e = (v_1, v_2) \in E_f^G$ is $f^e = (v_1, v_2, v_3)$.

If the objective is to control the robot to do locomotion from Figure 14.1a to Figure 14.1c, the result of this process is equivalent to rotating the truss with respect to the fixed edge $e = (v_1, v_2)$ until the support polygon becomes f^e. This means the input command for each locomotion step can be simply an edge of the current support polygon, e.g. (v_1, v_2) for the scenario in Figure 14.1.

14.2.2 Locomotion Planning

Given a VTT $G = (V, E)$, its truss polyhedron model $P^G = (E^G, F^G)$ and its current support polygon $f \in F^G$ can be derived. Given a locomotion command $e \in f$, f^e can be found and its normal vector n_{f^e} pointing inward P^G can be computed (e.g. vector n for facet (v_1, v_2, v_3) in Figure 14.1a), then the rotation angle α between n_{f^e} and the normal vector of the ground $[0, 0, 1]^\intercal$ is simply

$$\alpha = \arccos\left(\frac{n_{f^e} \bullet [0, 0, 1]^\intercal}{\|n_{f^e}\|}\right) \tag{14.1}$$

The rotation axis is along edge $e = (v_1^e, v_2^e)$ and its direction can be easily determined by the right-hand rule. For the example shown in Figure 14.1, the rotation axis is simply $\frac{q^{v_1} - q^{v_2}}{\|q^{v_1} - q^{v_2}\|}$, namely the unit vector pointing from node v_2 to node v_1. Assume the rotation axis is pointing from v_1^e to v_2^e, then the rotation angle and the rotation axis determine a rotation matrix R^e, and for every node $v \in V \setminus \{v_1^e, v_2^e\}$ during this locomotion process, its initial position q_i^v is known as q^v and its goal position q_g^v can be derived as

$$q_g^v = R^e(q^v - q^{v_1^e}) + q^{v_1^e} \tag{14.2}$$

Then the approach presented in Section 13.5 can be used to plan the motion for all of these involved nodes to execute the locomotion step. Due to the stability constraint that at least three nodes need to contact the ground to form a support polygon and the center of mass represented on the ground has to be within this support polygon, the planner will automatically move nodes first to expand the support polygon so that the center of mass can be moved in a larger range. An example of this non-impact rolling locomotion planning

result is shown in Figure 14.1. In this task, the initial support polygon is facet (v_0, v_1, v_2) and the task is to roll the truss so that facet (v_1, v_2, v_3) becomes the new support polygon. The locomotion command is simply (v_1, v_2), the rotation axis and the rotation angle can be computed as mentioned, and four nodes (v_0, v_3, v_4, and v_5) have to move to new locations which can be derived from Eq. (14.2). Because of the stability constraint, node v_0 cannot be moved at the beginning, or the robot won't be stable since there will be no support polygon. The planner chooses to move v_3 and v_5 first to expand the support polygon which is formed by node v_0, v_1, v_2, and v_3, and the center of mass is moved toward the target support polygon (Figure 14.1b). The center of mass projected onto the ground is always within the initial support polygon. After the support polygon is expanded, v_0 and v_4 are moved, and, in the meantime, the projected center of mass enters the target support polygon (Figure 14.1c).

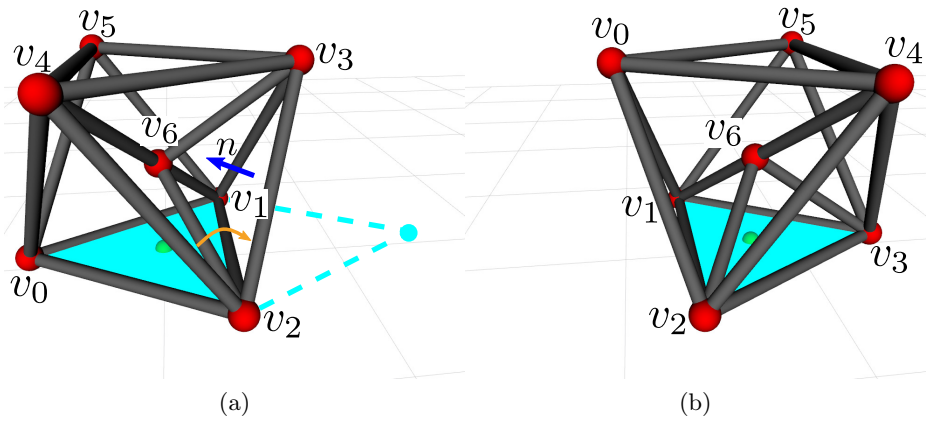
14.3 Test Scenarios

The VTT used for the locomotion test is shown in Figure 14.2a that is an octahedron with three additional edge modules internally. The initial locations of all nodes are

$$
\begin{aligned}
q^{v_0} &= [-0.7217, 0, 0.075]^\intercal & q^{v_1} &= [0.3608, 0.6250, 0.075]^\intercal \\
q^{v_2} &= [0.3608, -0.6250, 0.075]^\intercal & q^{v_3} &= [0.7217, 0, 1.0956]^\intercal \\
q^{v_4} &= [0.7217, 0, 1.0956]^\intercal & q^{v_5} &= [-0.3608, -0.6250, 1.0956]^\intercal \\
q^{v_6} &= [0, 0, 0.5853]^\intercal
\end{aligned}
$$

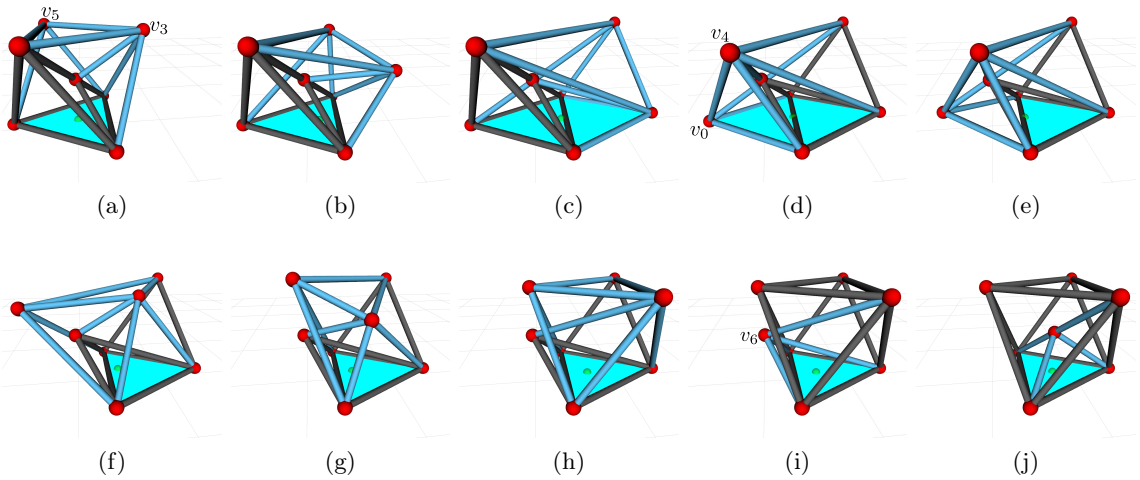
The constraints for this locomotion task are $\overline{L}_{\min} = 0.3\,\text{m}$, $\overline{L}_{\max} = 5.0\,\text{m}$, $\bar{\theta}_{\min} = 0.3\,\text{rad}$, and $\bar{\mu}_{\min} = 0.1$. The task is to roll this VTT to an adjacent support polygon shown in Figure 14.2b, and the locomotion command is (v_1, v_2). Five nodes (v_0, v_3, v_4, v_5, v_6) are involved in this process and their goal locations can be computed by Eq. (14.2).

The detailed locomotion process is shown in Figure 14.3. Five nodes are divided into three groups and the whole process is also divided into three phases automatically. In the first phase, v_3 and v_5 are moved to expand the support polygon (Figure 14.3c). Then v_0 and



(a) (b)

Figure 14.2: The locomotion task is to roll the truss from (a) to (b).



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 14.3: The planned motion for this locomotion task is shown. (a) — (c) Move node v_3 and v_5 to first expand the support polygon that is the aqua region. (d) Move v_0 and v_4 to move the center of mass represented on the ground (•) toward the target support polygon. (e) — (h) Once the center of mass represented on the ground is inside the target support polygon, lift v_0 and move both v_0 and v_4 to their target locations. (i) — (j) Finally move v_6 to its target location to finish the locomotion process.

v_4 are moved gradually to move the center of mass toward the adjacent support polygon shown in Figure 14.3d. Once the center of mass represented on the ground is inside the target support polygon, v_0 can be lifted off the ground and both v_0 and v_4 can be moved to their target locations (Figure 14.3e—Figure 14.3h). Finally, move node v_6 to its target location shown in Figure 14.3j to finish the whole process.

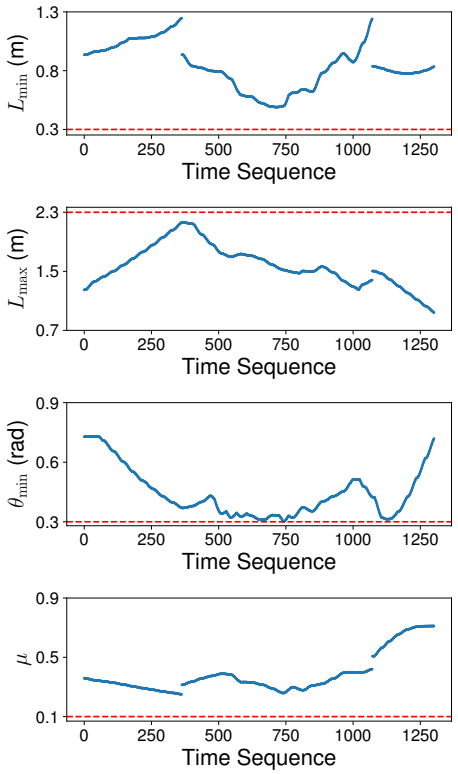


Figure 14.4: The minimum length ($L_{\min}$) and the maximum length ($L_{\max}$) of all moving edge modules, the minimum angle between every pair of edge modules ($\theta_{\min}$), and the motion manipulability (μ) are measured throughout the locomotion process in Figure 14.3.

The minimum length ($L_{\min}$) and the maximum length ($L_{\max}$) of all moving edge modules, the minimum angle between every pair of edge modules ($\theta_{\min}$), and the motion manipulability (μ) are shown in Figure 14.4. This rolling step is tested with 1000 trials and the average planning time is 8.297 s with the standard deviation being 5.935 s in 1000 trials, and the success rate is 100%. In these trials, the maximum planning time is 24.459 and the minimum planning time is 0.473 s. The locomotion of this VTT is also tested in [106] and the performance comparison is shown in Table 14.1. The presented planner here has better performance in terms of both efficiency and robustness.

An octahedron rolling gait is computed based on an optimization approach by [149]. Impacts are not considered in this approach and it takes 166 s to compute the whole process. The octahedron configuration is also tested with the presented framework and the comparison is shown in Table 14.2. The solution (Figure 14.1) can be derived as fast as

Table 14.1: Comparison with the optimization approach from [106].

	Average Planning Time	Success Rate
Optimization	18.834 s	38.3% (820 trials)
Sampling-Based	8.297 s	100% (1000 trials)

Table 14.2: Comparison with the optimization approach from [149].

	Average Planning Time	Success Rate
Optimization	166 s	Not provided
Sampling-Based	1.181 s	100% (1000 trials)

1.181 s in average with a standard deviation being 1.033 s in 1000 trials, and the success rate is 100%. In these trials, the maximum planning time is 3.437 s and the minimum planning time is 0.103 s.

14.4 Conclusion

A non-impact rolling locomotion algorithm is presented to move an arbitrary VTT easily by a simple command with the efficient geometry reconfiguration planner. This locomotion algorithm outperforms other approaches in terms of efficiency and robustness. This approach can also be applied to other truss robots.

Part V

Conclusions

Chapter 15

Contributions and Future Work

15.1 Contributions

Self-reconfigurable modular robots are promising to handle a much wider range of tasks through self-adaption in which they can reconfigure their morphologies whenever needed. This reconfiguration capability benefits the robot flexibility significantly but also complicates the design, control, and planning of these robots. This thesis focuses on a hierarchy framework to deploy a modular robotic system, including configuration modeling and recognition, morphology transformation, and task execution.

Modular robot morphologies are modeled as graphs, and several algorithms are presented for efficient configuration identification and comparison. Mobile-style reconfiguration behaviors are explored and shown to be efficient to achieve dexterous morphology transformation in which modules can alter existing connections to form a new morphology or several separated modules are able to join together into a connected morphology. The reconfiguration actions during a morphology transformation process are guaranteed by the phased docking controller and executed in parallel for efficiency. Manipulation tasks can be executed by modular robots using the proposed real-time trajectory planner in which the control and motion planning can be handled simultaneously. This is particularly useful for whole-body manipulation in which kinematic chains may share DOFs.

This thesis also contributes to the development of VTTs which are reconfigurable truss

systems. The modeling approach and the control framework are developed for an arbitrary VTT or truss robot. The complex configuration space is studied and can be computed by a fast computational geometry solution leading to efficient motion planning by manipulating truss nodes. The reconfiguration strategy is developed to transform a VTT significantly through geometry reconfiguration and topology reconfiguration. A non-impact rolling locomotion behavior can be computed by the presented planner to navigate a VTT in an environment. This locomotion planner outperforms other locomotion strategies for truss robots in efficiency and robustness.

These works are built for SMORES-EP, CKBot, and VTT, but can be generalized for other robots. The graph model of SMORES-EP can be applied to other modular robots with minor modifications, and the algorithm for configuration recognition can be used to identify a robot morphology. The morphology transformation strategy is useful for modular robotic systems as long as modules can move independently. The manipulation planning framework introduces a universal kinematics model and a motion planning tool for modular robots in chain structures for manipulation tasks. This manipulation planning tool can also be applied to whole-body manipulation with multi-limb robots. The modeling and motion control for VTT can be applied to other truss robots. The reconfiguration planning framework is able to handle the shape morphing tasks for truss robots by not only changing member lengths but also rearranging the connections of members. The locomotion planner is also a general tool for truss robots to achieve non-impact locomotion activities.

15.2 Future Work

Self-reconfigurable modular robots are promising and their flexibility is demonstrated in this thesis. However, it is still a challenge to come up with a good hardware design solution for this type of robot. For example, the docking interface is important, but it is usually limited to provide strong connection and robust communication between modules. Individual module dexterity also needs to be considered. Very dexterous module design can increase the flexibility of the system, but usually requires larger space and results in a more complicated mechanism. The SMORES-EP system is a new hybrid self-reconfigurable modular robot

which has been used in a variety of applications. However, the versatility of the hardware morphologies is limited due to the docking force of the EP-Face connectors. The VTT system brings about the topology reconfiguration capability for truss robots but the design of the reconfigurable node is complicated that also makes the topology reconfiguration process difficult.

In addition to the hardware design, task-driven morphology transformation is the fundamental to make use of the reconfiguration capability. This requires the motion capability evaluation of a morphology, the understanding of the environment, and also the compatibility between the task and the given morphology. There are several challenges. The number of possible morphologies can be very large and it is necessary to search this large space efficiently. The environment can be very different in different scenarios and characterizing an environment for morphology selection is challenging. This is also true for tasks. Additionally, the compatibility evaluation between a task and a morphology is hard to be general.

An important theme in this work is robot-to-robot collaboration. In this thesis, a robot is a module and several modules can work together. They can behave independently and execute tasks by collaboration, such as carrying an object. Furthermore, these modules can execute "fusion" behavior to be a more powerful individual. This capability can significantly improve their performance in an unstructured environment to handle unknown tasks. This can be very useful not just for modular robots but also for other types of robots and exploring this new collaboration behavior for robotic systems can be exciting future work.

Bibliography

[1] M. Abrahantes, L. Nelson, and P. Doorn, "Modeling and gait design of a 6-tetrahedron walker robot," in *2010 42nd Southeastern Symposium on System Theory (SSST)*, 2010, pp. 248–252.

[2] S. K. Agrawal, L. Kissner, and M. Yim, "Joint solutions of many degrees-of-freedom systems using dextrous workspaces," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 3, 2001, 2480–2485 vol.3.

[3] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, 1996, 113–120 vol.1.

[4] M. Asadpour, M. H. Z. Ashtiani, A. Sproewitz, and A. Ijspeert, "Graph signature for self-reconfiguration planning of modules with symmetry," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 5295–5300.

[5] J. Baca, B. Woosley, P. Dasgupta, and C. Nelson, "Real-time distributed configuration discovery of modular self-reconfigurable robots," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1919–1924.

[6] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, 469–483, Dec. 1996.

[7] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991.

[8] C. Bererton and P. K. Khosla, "Towards a team of robots with repair capabilities: A visual docking system," in *Experimental Robotics VII*, D. Rus and S. Singh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 333–342.

[9] S. Bonardi, M. Vespignani, R. Moeckel, J. V. den Kieboom, S. Pouya, A. Sproewitz, and A. Ijspeert, "Automatic generation of reduced CPG control networks for locomotion of arbitrary modular robot structures," in *Proceedings of Robotics: Science and Systems*, Berkeley, USA, 2014.

[10] G. Bradshaw and C. O'Sullivan, "Adaptive medial-axis approximation for sphere-tree construction," *ACM Trans. Graph.*, vol. 23, no. 1, 1–26, 2004.

[11] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[12] H. B. Brown, J. M. Vande Weghe, C. A. Bererton, and P. K. Khosla, "Millibot trains for enhanced mobility," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 452–461, 2002.

[13] Z. Butler, R. Fitch, D. Rus, and Y. Wang, "Distributed goal recognition algorithms for modular robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 1, 2002, 110–116 vol.1.

[14] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic decentralized control for a class of self-reconfigurable robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 1, 2002, 809–816 vol.1.

[15] A. Casal and M. H. Yim, "Self-reconfiguration planning for a class of modular robots," in *Sensor Fusion and Decentralized Control in Robotic Systems II*, G. T. McKee and P. S. Schenker, Eds., International Society for Optics and Photonics, vol. 3839, SPIE, 1999, pp. 246 –257.

[16] A. Castano, A. Behar, and P. M. Will, "The Conro modules for reconfigurable robots," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 403–409, 2002.

[17] A. Castano and P. Will, "Representing and discovering the configuration of Conro robots," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 4, 2001, 3503–3509 vol.4.

[18] J. J. Champoux, "DNA topoisomerases: Structure, function, and mechanism," *Annual Review of Biochemistry*, vol. 70, no. 1, pp. 369–413, 2001, PMID: 11395412.

[19] B. Chazelle, "Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm," *SIAM Journal on Computing*, vol. 13, no. 3, pp. 488–507, 1984.

[20] I. Chen and J. W. Burdick, "Enumerating the nonisomorphic assembly configurations of modular robotic systems," in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, vol. 3, 1993, 1985–1992 vol.3.

[21] G. S. Chirikjian, "Kinematics of a metamorphic robotic system," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994, 449–455 vol.1.

[22] G. S. Chirikjian and J. W. Burdick, "A geometric approach to hyper-redundant manipulator obstacle avoidance," *Journal of Mechanical Design*, vol. 114, no. 4, pp. 580–585, Dec. 1992.

[23] D. T. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A MoveIt! case study," *Journal of Software Engineering for Robotics*, vol. 5, no. 4, pp. 3–16, 2014.

[24] F. Collins and M. Yim, "Design of a spherical robot arm with the spiral zipper prismatic joint," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2137–2143.

[25] J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell, "An integrated system for perception-driven autonomy with modular robots," *Science Robotics*, vol. 3, no. 23, 2018.

[26] J. Davey, N. Kwok, and M. Yim, "Emulating self-reconfigurable robots — design of the SMORES system," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4464–4469.

[27] A. Dutta, P. Dasgupta, and C. Nelson, "Distributed configuration formation with modular robots using (sub)graph isomorphism-based approach," *Autonomous Robots*, vol. 43, no. 4, pp. 837–857, 2019.

[28] N. Eckenstein and M. Yim, "Modular robot connector area of acceptance from configuration space obstacles," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3550–3555.

[29] ——, "The X-Face: An improved planar passive mechanical connector for modular self-reconfigurable robots," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 3073–3078.

[30] R. Fitch and Z. Butler, "Million module march: Scalable locomotion for large self-reconfiguring robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 331–343, 2008.

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

[32] T. Fukuda, M. Buss, H. Hosokai, and Y. Kawauchi, "Cell structured robotic system cebot: Control, planning and communication methods," *Robotics and Autonomous Systems*, vol. 7, no. 2, pp. 239 –248, 1991, Special Issue Intelligent Autonomous Systems.

[33] D. A. Furcy, "Speeding up the convergence of online heuristic search and scaling up offline heuristic search," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, 2004.

[34] K. Gilpin, A. Knaian, and D. Rus, "Robot pebbles: One centimeter modules for programmable matter through self-disassembly," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 2485–2492.

[35] K. Gilpin, K. Kotay, and D. Rus, "Miche: Modular shape formation by self-dissasembly," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 2241–2247.

[36] A. Giusti and M. Althoff, "Automatic centralized controller design for modular and reconfigurable robot manipulators," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3268–3275.

[37] R. Graham, "An efficient algorith for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, no. 4, pp. 132 –133, 1972.

[38] R. Groß, M. Bonani, F. Mondada, and M. Dorigo, "Autonomous self-assembly in a swarm-bot," in *Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2005)*, K. Murase, K. Sekiyama, T. Naniwa, N. Kubota, and J. Sitte, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 314–322.

[39] Gurobi Optimization. "Gurobi Optimizer." version 9.1, [Online]. Available: `https://www.gurobi.com/products/gurobi-optimizer/`.

[40] B. Haghighat, E. Droz, and A. Martinoli, "Lily: A miniature floating robotic platform for programmable stochastic self-assembly," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1941–1948.

[41] G. J. Hamlin and A. C. Sanderson, "TETROBOT: A modular approach to parallel robotics," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 42–50, 1997.

[42] ——, "TETROBOT modular robotics: Prototype and experiments," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*, vol. 2, 1996, 390–395 vol.2.

[43] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[44] M. Herman, "Fast, three-dimensional, collision-free motion planning," in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, 1986, pp. 1056–1063.

[45] S. Hirose, T. Shirasu, and E. F. Fukushima, "Proposal for cooperative robot "Gunryu" composed of autonomous segments," *Robotics and Autonomous Systems*, vol. 17, no. 1, pp. 107 –118, 1996.

[46] P. Holmes, S. Kousik, B. Zhang, D. Raz, C. Barbalata, M. J. Roberson, and R. Vasudevan, "Reachable sets for safe, real-time manipulator trajectory design," in *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, 2020.

[47] F. Hou, "Self-reconfiguration planning for modular robots," Ph.D. dissertation, University of Southern California, Los Angeles, CA, 2011.

[48] F. Hou and W.-M. Shen, "Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 3135–3140.

[49] ——, "Graph-based optimal reconfiguration planning for self-reconfigurable robots," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1047 –1059, 2014, Reconfigurable Modular Robotics.

[50] D. Hsu, J. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *Proceedings of International Conference on Robotics and Automation*, vol. 3, 1997, 2719–2726 vol.3.

[51] R. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Information Processing Letters*, vol. 2, no. 1, pp. 18 –21, 1973.

[52] S. Jeong, B. Kim, S. Park, E. Park, A. Spinos, D. Carroll, T. Tsabedze, Y. Weng, T. Seo, M. Yim, F. C. Park, and J. Kim, "Variable topology truss: Hardware overview, reconfiguration planning and locomotion," in *2018 15th International Conference on Ubiquitous Robots (UR)*, 2018, pp. 610–615.

[53] G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit, "An end-to-end system for accomplishing tasks with modular robots," in *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, 2016.

[54] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund, "Modular ATRON: Modules for a self-reconfigurable robot," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 2, 2004, 2068–2073 vol.2.

[55] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 4569–4574.

[56] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[57] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji, "Automatic locomotion design and experiments for a modular robotic system," *IEEE/ASME Transactions on Mechatronics*, vol. 10, no. 3, pp. 314–325, 2005.

[58] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[59] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[60] L. Kettner, "Using generic programming for designing a data structure for polyhedral surfaces," *Computational Geometry*, vol. 13, no. 1, pp. 65–90, 1999.

[61] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[62] B. T. Kirby, B. Aksak, J. D. Campbell, J. F. Hoburg, T. C. Mowry, P. Pillai, and S. C. Goldstein, "A modular robotic system using magnetic force effectors," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2787–2793.

[63] E. Klavins, R. Ghrist, and D. Lipsky, "A grammatical approach to self-organizing robotic systems," *IEEE Transactions on Automatic Control*, vol. 51, no. 6, pp. 949–962, 2006.

[64] A. N. Knaian, "Electropermanent magnetic connectors and actuators: Devices and their application in programmable matter," Ph.D. dissertation, Massachusetts Institute of Technology, Boston, 2010.

[65] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata, "Distributed self-reconfiguration of M-TRAN III modular robotic system," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 373–386, 2008.

[66] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, Available at http://planning.cs.uiuc.edu/.

[67] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, pp. 473–479.

[68] W. H. Lee and A. C. Sanderson, "Dynamic rolling locomotion and control of modular robots," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 1, pp. 32–41, 2002.

[69] H. Li, T. Wang, and G. S. Chirikjian, "Self-assembly planning of a shape by regular modular robots," in *Advances in Reconfigurable Mechanisms and Robots II*, X. Ding, X. Kong, and J. S. Dai, Eds., Cham: Springer International Publishing, 2016, pp. 867–877.

[70] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA* : Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Schölkopf, Eds., vol. 16, MIT Press, 2004.

[71] M. Likhachev and A. Stentz, "R* search," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, ser. AAAI'08, Chicago, Illinois: AAAI Press, 2008, 344–350.

[72] C. Liu, A. Bera, T. Tsabedze, D. Edgar, and M. Yim, "Spiral zipper manipulator for aerial grasping and manipulation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 3179–3184.

[73] C. Liu, M. Whitzer, and M. Yim, "A distributed reconfiguration planning algorithm for modular robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4231–4238, 2019.

[74] C. Liu and M. Yim, "A quadratic programming approach to modular robot control and motion planning," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, Taichung, Taiwan, 2020, pp. 1–8.

[75] ——, "Reconfiguration motion planning for variable topology truss," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1941–1948.

[76] C. Liu, S. Yu, and M. Yim, "A fast configuration space algorithm for variable topology truss modular robots," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 8260–8266.

[77] ——, "Shape morphing for variable topology truss," in *2019 16th International Conference on Ubiquitous Robots (UR)*, Jeju, Korea, 2019.

[78] C. Liu, Q. Lin, H. Kim, and M. Yim, "SMORES-EP, a modular robot with parallel self-assembly," *arXiv preprint arXiv:2104.00800*, 2021.

[79] C. Liu, T. Tosun, and M. Yim, "A low-cost, highly customizable solution for position estimation in modular robots," *Journal of Mechanisms and Robotics*, vol. 13, no. 6, 2021.

[80] C. Liu and M. Yim, "A quadratic programming approach to manipulation in real-time using modular robots," *The International Journal of Robotic Computing*, vol. 3, no. 1, pp. 121–145, 2021.

[81] C. Liu and M. Yim, "Configuration recognition with distributed information for modular robots," in *IFRR International Symposium on Robotics Research*, Puerto Varas, Chile, 2017.

[82] C. Liu, S. Yu, and M. Yim, "Motion planning for variable topology truss modular robot," in *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, 2020.

[83] ——, "Motion planning for variable topology trusses: Reconfiguration and locomotion," *arXiv preprint arXiv:2108.00309*, 2021.

[84] G. Liu, S. Abdul, and A. A. Goldenberg, "Distributed control of modular and reconfigurable robot with torque sensing," *Robotica*, vol. 26, no. 1, 75–84, 2008.

[85] W. Liu and A. F. T. Winfield, "Self-assembly in heterogeneous modular robots," in *Distributed Autonomous Robotic Systems*, M. Ani Hsieh and G. Chirikjian, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 219–232.

[86] A. Lyder, R. F. M. Garcia, and K. Stoy, "Mechanical design of Odin, an extendable heterogeneous deformable modular robot," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 883–888.

[87] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 109–117, 1985.

[88] G. L. McColm, "On the structure of random unlabelled acyclic graphs," *Discrete Math.*, vol. 277, no. 1, 147–170, 2004.

[89] W. W. Melek and A. A. Goldenberg, "Neurofuzzy control of modular and reconfigurable robots," *IEEE/ASME Transactions on Mechatronics*, vol. 8, no. 3, pp. 381–389, 2003.

[90] G. Mermoud, M. Mastrangeli, U. Upadhyay, and A. Martinoli, "Real-time automated modeling and control of self-assembling systems," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 4266–4273.

[91] K. Miura, "Design and operation of a deployable truss structure," in *NASA. Goddard Space Flight Center The 18th Aerospace Mech. Symp.*, Greenbelt, Maryland, May 1984, pp. 49–63.

[92] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *Journal of Graphics Tools*, vol. 2, no. 1, pp. 21–28, 1997.

[93] K. Motomura, A. Kawakami, and S. Hirose, "Development of arm equipped single wheel rover: Effective arm-posture-based steering method," *Autonomous Robots*, vol. 18, pp. 215–229, 2005.

[94] S. Murata, K. Kakomura, and H. Kurokawa, "Docking experiments of a modular robot by visual feedback," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 625–630.

[95] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji, "A 3-D self-reconfigurable structure," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 1, 1998, 432–439 vol.1.

[96] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-TRAN: Self-reconfigurable modular robotic system," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 431–441, 2002.

[97] R. M. Murray, S. S. Sastry, and L. Zexiang, *A Mathematical Introduction to Robotic Manipulation*. USA: CRC Press, Inc., 1994.

[98] L. E. Navarro-Serment, R. Grabowski, C. J. J. Paredis, and P. K. Khosla, "Modularity in small distributed robots," in *Sensor Fusion and Decentralized Control in Robotic Systems II*, G. T. McKee and P. S. Schenker, Eds., International Society for Optics and Photonics, vol. 3839, SPIE, 1999, pp. 297 –306.

[99] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein, "A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots," in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, 2016, pp. 254–263.

[100] C. A. Nelson, "A framework for self-reconfiguration planning for unit-modular robots," Ph.D. dissertation, Purdue University, West Lafayette, IN, 2005.

[101] M. Nilsson, "Heavy-duty connectors for self-reconfiguring robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 4, 2002, 4071–4076 vol.4.

[102] I. O'Hara, J. Paulos, J. Davey, N. Eckenstein, N. Doshi, T. Tosun, J. Greco, J. Seo, M. Turpin, V. Kumar, and M. Yim, "Self-assembly of a swarm of autonomous boats into floating structures," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1234–1240.

[103] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.

[104] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments," in *Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'12, Atibaia, São Paulo, Brazil: AAAI Press, 2012, pp. 207–215.

[105] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 403–421, 2008.

[106] S. Park, J. Bae, S. Lee, M. Yim, J. Kim, and T. Seo, "Polygon-based random tree search planning for variable geometry truss robot," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 813–819, 2020.

[107] S. Park, E. Park, M. Yim, J. Kim, and T. W. Seo, "Optimization-based nonimpact rolling locomotion of a variable geometry truss," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 747–752, 2019.

[108] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, 1993, 802–807 vol.2.

[109] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 489–494.

[110] E. Rimon and D. E. Koditschek, "Exact robot navigation using cost functions: The case of distinct spherical boundaries in E^n," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, 1988, 1791–1796 vol.3.

[111] "Rosbridge," [Online]. Available: `https://wiki.ros.org/rosbridge_suite`.

[112] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 3293–3298.

[113] M. Rubenstein, K. Payne, P. Will, and Wei-Min Shen, "Docking among independent and autonomous CONRO self-reconfigurable robots," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, 2004, 2877–2882 Vol.3.

[114] D. Rus and M. Vona, "Crystalline robots: Self-reconfiguration with compressible unit modules," *Autonomous Robots*, vol. 10, no. 1, pp. 107–124, 2001.

[115] *S1791-42r customer information sheet*, Harwin Inc., 2016.

[116] D. Saldaña, B. Gabrich, G. Li, M. Yim, and V. Kumar, "ModQuad: The flying modular structure that self-assembles in midair," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 691–698.

[117] D. Saldaña, B. Gabrich, M. Whitzer, A. Prorok, M. F. M. Campos, M. Yim, and V. Kumar, "A decentralized algorithm for assembling structures with modular robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2736–2743.

[118] D. Saldaña, P. M. Gupta, and V. Kumar, "Design and control of aerial modules for inflight self-disassembly," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3410–3417, 2019.

[119] B. Salemi, M. Moll, and W. Shen, "SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 3636–3641.

[120] B. Salemi, Wei-Min Shen, and P. Will, "Hormone-controlled metamorphic robots," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 4, 2001, 4194–4199 vol.4.

[121] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *2001 European Control Conference (ECC)*, 2001, pp. 2603–2608.

[122] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[123] J. Seo, M. Yim, and V. Kumar, "Assembly sequence planning for constructing planar structures with rectangular modules," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 5477–5482.

[124] J. Seo, S. Gray, V. Kumar, and M. Yim, "Reconfiguring chain-type modular robots based on the Carpenter's Rule Theorem," in *Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics*, D. Hsu, V. Isler, J.-C. Latombe, and M. C. Lin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 105–120.

[125] J. Seo, J. Paik, and M. Yim, "Modular reconfigurable robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 63–88, 2019.

[126] J. Seo, M. Yim, and V. Kumar, "A theory on grasping objects using effectors with curved contact surfaces and its application to whole-arm grasping," *The International Journal of Robotics Research*, vol. 35, no. 9, pp. 1080–1102, 2016.

[127] H. Seraji, "Configuration control of redundant manipulators: Theory and implementation," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 4, pp. 472–490, 1989.

[128] K. Shankar, J. W. Burdick, and N. H. Hudson, "A quadratic programming approach to quasi-static whole-body manipulation," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, H. L. Akin, N. M. Amato, V. Isler, and A. F. van der Stappen, Eds. Cham: Springer International Publishing, 2015, pp. 553–570.

[129] W. Shen, R. Kovac, and M. Rubenstein, "SINGO: A single-end-operative and genderless connector for self-reconfiguration, self-assembly and self-healing," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 4253–4258.

[130] Z. Shiller and S. Dubowsky, "On computing the global time-optimal motions of robotic manipulators in the presence of obstacles," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 785–797, 1991.

[131] M.-C. Shiu, L.-C. Fu, and Y.-J. Chia, "Graph isomorphism testing method in a self-recognition Velcro strap modular robot," in *2010 5th IEEE Conference on Industrial Electronics and Applications*, 2010, pp. 222–227.

[132] A. Spinos, D. Carroll, T. Kientz, and M. Yim, "Variable topology truss: Design and analysis," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2717–2722.

[133] A. Sproewitz, R. Moeckel, J. Maye, and A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 423–443, 2008.

[134] A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi, and A. Ijspeert, "Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1016 –1033, 2014, Reconfigurable Modular Robotics.

[135] K. Stoy, D. Brandt, and D. Christensen, *Self-Reconfigurable Robots*. Cambridge, MA: The MIT Press, 2010.

[136] K. Stoy, Wei-Min Shen, and P. M. Will, "Using role-based control to produce locomotion in chain-type self-reconfigurable robots," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 410–417, 2002.

[137] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[138] J. W. Suh, S. B. Homans, and M. Yim, "Telecubes: Mechanical design of a module for self-reconfigurable robotics," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 4, 2002, 4095–4101 vol.4.

[139] "The Boost Graph Library (BGL)." (2019), [Online]. Available: `https://www.boost.org/doc/libs/1_71_0/libs/graph/doc/index.html`.

[140] "The computational geometry algorithms library." (2019), [Online]. Available: `https://www.cgal.org/`.

[141] "The robot operating system (ros)," [Online]. Available: `https://www.ros.org/`.

[142] M. T. Tolley and H. Lipson, "On-line assembly planning for stochastically reconfigurable systems," *The International Journal of Robotics Research*, vol. 30, no. 13, pp. 1566–1584, 2011.

[143] T. Tosun, J. Davey, C. Liu, and M. Yim, "Design and characterization of the EP-Face connector," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 45–51.

[144] T. Tosun, D. Edgar, C. Liu, T. Tsabedze, and M. Yim, "PaintPots: Low cost, accurate, highly customizable potentiometers for position sensing," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1212–1218.

[145] *Total ground carbon conductive coating 838 technical data sheet*, Ver. 1.04, MG Chemicals, 2013.

[146] M. Turpin, N. Michael, and V. Kumar, "Concurrent assignment and planning of trajectories for large teams of interchangeable robots," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 842–848.

[147] "Unity," [Online]. Available: `https://unity.com/`.

[148] N. Usevitch, Z. Hammond, S. Follmer, and M. Schwager, "Linear actuator robots: Differential kinematics, controllability, and algorithms for locomotion and shape morphing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 5361–5367.

[149] N. S. Usevitch, Z. M. Hammond, and M. Schwager, "Locomotion of linear actuator robots through kinematic planning and nonlinear optimization," *IEEE Transactions on Robotics*, pp. 1–18, 2020.

[150] S. Vassilvitskii, M. Yim, and J. Suh, "A complete, local and parallel reconfiguration algorithm for cube style modular robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 1, 2002, 117–122 vol.1.

[151] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, 2000, pp. 153–158.

[152] H. Wei, Y. Chen, J. Tan, and T. Wang, "Sambot: A self-assembly modular robot system," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 4, pp. 745–757, 2011.

[153] J. Werfel, D. Ingber, and R. Nagpal, "Collective construction of environmentally-adaptive structures," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2345–2352.

[154] J. Werfel and R. Nagpal, "Three-dimensional construction with mobile robots and modular blocks," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 463–479, 2008.

[155] P. J. White, K. Kopanski, and H. Lipson, "Stochastic self-reconfigurable cellular robotics," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, 2004, 2888–2893 Vol.3.

[156] P. White, V. Zykov, J. Bongard, and H. Lipson, "Three dimensional stochastic reconfiguration of modular robots," in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, 2005.

[157] M. Yim, "A reconfigurable modular robot with many modes of locomotion," in *Porc. of JSME Intl. Conf. on Advanced Mechatronics*, Tokyo, Japan, 1993.

[158] M. Yim, "New locomotion gaits," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994, 2508–2514 vol.3.

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

[160] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[161] M. Yim, B. Shirmohammadi, J. Sastra, M. Park, M. Dugan, and C. J. Taylor, "Towards robotic self-reassembly after explosion," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2767–2772.

[162] M. Yim, Ying Zhang, K. Roufas, D. Duff, and C. Eldershaw, "Connecting and disconnecting for chain self-reconfiguration with PolyBot," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 442–451, 2002.

[163] M. Yim, P. White, M. Park, and J. Sastra, "Modular self-reconfigurable robots," in *Encyclopedia of Complexity and Systems Science*, R. A. Meyers, Ed. New York, NY: Springer New York, 2009, pp. 5618–5631.

[164] M. H. Yim, D. Goldberg, and A. Casal, "Connectivity planning for closed-chain reconfiguration," in *Sensor Fusion and Decentralized Control in Robotic Systems III*, G. T. McKee and P. S. Schenker, Eds., International Society for Optics and Photonics, vol. 4196, SPIE, 2000, pp. 402 –412.

[165] Ying Zhang, M. Yim, C. Eldershaw, D. Duff, and K. Roufas, "Phase Automata: A programming model of locomotion gaits for scalable chain-type modular robots," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 3, 2003, 2442–2447 vol.3.

[166] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji, "A self-reconfigurable modular robot: Reconfiguration planning and experiments," *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 903–915, 2002.

[167] Y. Zhao, H. Lin, and M. Tomizuka, "Efficient trajectory optimization for robot motion planning," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2018, pp. 260–265.

[168] W. Zhu, T. Lamarche, E. Dupuis, D. Jameux, P. Barnard, and G. Liu, "Precision control of modular robot manipulators: The VDC approach with embedded FPGA," *IEEE Transactions on Robotics*, vol. 29, no. 5, pp. 1162–1179, 2013.

[169] Y. Zhu, G. Li, X. Wang, and X. Cui, "Automatic function-isomorphic configuration recognition and control for ubot modular self-reconfigurable robot," in *2012 IEEE International Conference on Mechatronics and Automation*, 2012, pp. 451–456.

[170] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.