Computational Design of Foldable Robots via Composition

by

Cynthia Rueyi Sung

B.S., Rice University (2011)
S.M., Massachusetts Institute of Technology (2013)

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

Author .....
Department of
Electrical Engineering and Computer Science
August 31, 2016

Certified by .....
Daniela Rus
Andrew (1956) and Erna Viterbi Professor
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Computational Design of Foldable Robots via Composition

by

Cynthia Rueyi Sung

Submitted to the Department of
Electrical Engineering and Computer Science
on August 31, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Recent advances in rapid fabrication technologies have given designers the ability to manufacture increasingly complex geometries with little increase in cost, making it easier than ever to build a robot. However, the process of designing a functional robot remains challenging. Robots are complex systems that tightly integrate mechanical, electrical, and software subsystems. As a result, traditional robot development often requires iterations of design and testing to ensure that the result is both functional and manufacturable.

This thesis explores intuitive design tools for robot design and proposes composition as a design approach. We leverage a print-and-fold paradigm of manufacturing, which has been shown to enable functional robots to be created within a day. The main challenge in using composition is that in general, even if two modules function correctly individually, the combination of the two may not be a valid design. We therefore develop algorithms and systems for robot composition that guarantee validity of the design geometry and that check the resulting kinematics.

Our main contributions include a database containing parameterized designs for printable joints and mechanisms, algorithms for composition of fold patterns and motion sequences that guarantee no self-intersection, automated generation of fabrication plans for multiple modes of print-and-fold fabrication, an interactive user interface in which users can compose custom robots and receive real-time feedback about their designs, and experimental verification in the form of functional mechanisms and robots. The results provide designers with a framework for rapid design exploration and bring us closer to a future of easy robot design and customization.

Thesis Supervisor: Daniela Rus
Title: Andrew (1956) and Erna Viterbi Professor

To my family,

with love.

Acknowledgments

This thesis would never have been completed without the help and support of many people.

First and foremost, I would like to thank my research advisor, Daniela Rus, who has been an awesome mentor throughout the course of my Ph.D. Daniela has always been available to answer any of my many questions, and she has taken a genuine interest in not only my research progress but also my motivation and general happiness. Her enthusiasm for robotics and its potential to change the world is infectious, and her drive to always "Keep the gradient!" has made her one of my greatest role models. Daniela, thank you so much for your guidance and support!

I would also like to thank my thesis committee members: Erik Demaine, Wojciech Matusik, and Vijay Kumar. Even before joining this committee, all three of them have provided insightful comments and advice on the research. Erik is primarily responsible for bringing me into the world of folding theory, while Wojciech introduced me to data-driven techniques, and Vijay has kept me grounded in practical application considerations. Their feedback has raised the level of this research considerably.

To all the members of the Distributed Robotics Lab extended family, I would like to express the greatest appreciation. On the technical side, I would particularly like to recognize Cagdas Onal and Michael Tolley, who introduced me to the idea of printable robots; Joseph DelPreto, who has let me "borrow" his electronics parts countless times; Brian Julian and Mehmet Dogar, who helped me to take motion capture measurements; and Ankur Mehta, Shuhei Miyashita, John Romanishin, Adriana Schulz, and Andrew Spielberg, who have all been valuable collaborators. I have been fortunate to have had the support of amazing technicians such as Ron Wiken, lab assistants Kathy Bates and Mieke Moran, and undergraduate researchers James Bern, Elizabeth Hadley, and Rhea Lin, whose high skills are only matched by their enthusiasm. Personally, I would also like to thank Andrew Marchese, who was the first to welcome me to the lab and who was always willing to answer any questions I had

about the Ph.D., as well as Nora Ayanian and Ross Knepper for their continuous motivational support.

Last but not least, I would like to thank my family: my mother, Chin Fong Pau, who has been there for all my successes and failures and who probably knows as much about this thesis as I do; my father, Chung Shin Sung, who is always willing to speculate about its potential impact on the world; and my siblings, Angela and Eric, who are always interested in talking about the latest in science and engineering. I dedicate this thesis to you.

This work was done in the Distributed Robotics Laboratory at MIT with support from NSF Grant Numbers 1240383 and 1138967, and from the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program. I am grateful for their support.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

A long-held goal in the robotics field has been to see our technologies enter the hands of the everyman. With the emergence of retailers such as Pololu [138] and Adafruit® [2], as well as products such as the iRobot® Roomba® [81] vacuum cleaner, this goal has recently started to become a reality. Despite these advances, customizing robotic technology to individual needs remains a challenge. Robots are complex systems that tightly integrate mechanical, electronic, and computational subsystems. As a result, customization at anything more than a superficial level often requires a nonnegligible amount of engineering skill. And yet, in order for robots to be able to address the individual needs of their users, they must allow for personalization.

Traditionally, robot development is a challenging and time-consuming process involving many iterations of design and testing, even for skilled engineers. Designers who decide to tackle this challenge must be able not only to devise and integrate physical and computational subcomponents, but also to evaluate manufacturability, usability, reliability, and other practical issues. With a required knowledge base this large, it should be no wonder that such projects often go through multiple prototypes before converging to a final design.

Given this design paradigm, a natural question is: *Is it possible to simplify design evaluation, cut down on the number of prototyping cycles, or otherwise make the design process more accessible?* The answer from a hardware perspective is a resounding *yes*. Recent advances in rapid fabrication [88, 201] have made creating complex 3-D

physical objects easier than ever, allowing people to realize their designs in hours or days instead of weeks or years. Unfortunately, when it comes to evaluation and design cycles, current design tools still present users with clear limitations, and the learning curve is steep for anyone who wishes to create a design from scratch.

In this thesis, we explore intuitive design tools that complement current rapid fabrication methods, with the goal of providing all roboticists, from dabblers to skilled professionals, with a framework for rapidly exploring, evaluating, and realizing their robotic designs. The tools incorporate simulations and interactive feedback with algorithms for design automation to streamline parts of the design cycle and give users the information they need to make design choices. We focus in particular on foldable robots, robots whose mechanical parts are fabricated as flat sheets and folded into their final 3-D form. As we will show, creating robots in this way accelerates the fabrication process and enables us to construct strong and lightweight structures well suited for robot designs, but it also presents additional fabrication constraints that complicate the design process. We tackle this challenge by proposing a design-by-composition approach to foldable robot design. In this approach, modules are composed together to form entire mechanisms and robots, and the validity of the design is maintained at every step. The approach not only finds practical use in an interactive design tool, but also provides insights into the space of feasible foldable designs.

1.1 New Capabilities

The results of this thesis will give rise to new capabilities for robot creators.

Exploratory Tools Tools that provide intuitive interfaces for creating and analyzing designs lower the cost of design exploration. In addition, analyzing users' designs before prototyping allows them to explore many different choices and find potential issues early before committing to a particular design. Since the effect of early design choices on a final product's performance is large compared to later choices [79], early

exploration of the design space is a valuable ability with the potential to significantly decrease time spent in later parts of the design process.

Rapid Iteration and Prototyping Tools that incorporate manufacturability constraints accelerate design iteration by decreasing the time to prototype. While advances in additive manufacturing have significantly increased prototyping capabilities, they suffer from a lack of corresponding design and modeling tools. The ability to design specifically for a fabrication technology will allow designers who otherwise would have had to wait weeks or months to finish a physical prototype to quickly evaluate their designs and make important design choices or changes.

Educational Tools Intuitive design interfaces provide valuable learning tools to novice engineers. Real-time design analysis and interactive feedback provide users with relevant information necessary to evaluate different design choices and to see how changes to the design affect its performance. Further, the ability to rapidly fabricate these designs allows engineering students to quickly receive real-world feedback about their robots while their design choices are still fresh.

Customized Technology Accessible design tools lower the barrier to entry to robot design. As a result, aspiring designers who are not trained in engineering will still be able to make the necessary design decisions and trade-offs to create customized robotic technologies to use in their own lives.

1.2 Challenges

The main challenge with robot design is that the design space is large, and there is a great amount of flexibility. Take, for example, the task of designing a robot to perform the simple locomotion task of moving across an empty room. There are many different robots that could do this task: cars that roll across the floor, legged robots that walk, robots that hop from one side of the room to the other, robots that fly, robots that slither, or robots that engage in myriad other possible types of locomotion. Further,

for each of these types of robots, the particular robot could take on any number of forms. A robot design tool must allow users to make interesting design choices while presenting the design space in an intuitive and manageable manner. In this thesis, we address this problem using a design-by-composition framework.

A second challenge is the interdependence of subsystems in a robotic design. Consider again the robot that must move across a room. The robot has a mechanical body that has a geometry, has a particular kinematic structure, and is made of certain materials; it has actuators, sensors, and electronics that control the robot's interactions with the environment at the low-level; and it has software that provides its high-level behaviors. Each of these components affects the other. For example, the mechanical body affects the actuator requirements and the efficiency of the programmed behaviors, while the electronics and software induce geometric and control constraints. A robot design tool must simultaneously consider these subsystems and allow users to explore how changes to one aspect of a robot design effect changes in another. In this thesis, we address the subproblem of geometry-motion interdependence.

A third challenge is the transition from a conceptual design to a physical robot. Any design that a user creates must be manufacturable, that is, each of its pieces must be able to be fabricated, and the disparate pieces must be able to be assembled. Therefore, a robot design tool should validate not only a robot's final form but also its fabrication plan. In this thesis, we address manufacturability in the context of foldable robots. In order to create foldable robots, designs must be able to be fabricated as 2-D fold patterns, a constraint that introduces its own set of challenges. In addition, transformable folded structures (i.e., folded structures that can move), which are the structures of interest for robot designs, are not well understood. We explore what kinds of robotic designs can be assembled by folding and, in the context of design by composition, composed into more complex patterns.

1.3 Design by Composition

Our approach to tackling these challenges is to use a design-by-composition framework. In this framework, users create entire robots by composing modules from a database of parameterized designs. The database focuses the design space to a set of designs that are manageable for a computational tool, while the parameterization of individual modules provides users with enough freedom to express themselves creatively [57]. In addition, the designs in the database, which can be augmented by experienced designers, provide users with proven examples with which they can build future designs.

Design by composition also tackles the second challenge of concurrent subsystem design. The main challenge of composition is that even when modules are individually valid, their combination may not be a valid design. We include both geometric- and motion-related modules in the database, and we incorporate simulations and optimization methods that evaluate design performance in the context of both geometric and motion data. As users compose modules from the database, constraints on the module parameters that result from the composition are automatically associated with the design. Our evaluation and optimization methods use the module parameterizations and composition constraints to allow users to automatically optimize over multiple aspects of the robot design.

Finally, design by composition addresses the third challenge of manufacturability. We ensure that every module in the database is independently manufacturable for any set of parameter values in the feasible set of the design. By developing methods for module composition and parameter manipulation that maintain manufacturability, composed designs are automatically manufacturable as well.

1.3.1 Scope and Limitations

As with any data-driven method, a design-by-composition approach restricts the possible output designs to those that can be composed from the database. Experienced

designers are needed to expand the set of building blocks from which users can compose their own designs.

In addition, this thesis addresses composition of only geometry and motion modules. Expanding this work to include other aspects of robot design requires introducing new composition algorithms to handle the new types of design constraints, as well as new simulation and evaluation capabilities.

Our design tool is driven by the user, who chooses modules from the database to compose. Optimization methods built on top of this tool can provide feedback about parameter changes, but not about module changes. Users can explore how the simulations and evaluations change as they add and remove parts manually.

Finally, this thesis focuses on foldable robots, and our database contains only foldable designs. Although the composition algorithms and data representation described are particular to folding, however, the underlying methods can be generalized to other fabrication methods. Augmenting this work to allow other fabrication methods requires new analysis of the constraints associated with those methods, as well as new composition algorithms.

1.4 Thesis Contributions

This thesis makes the following contributions:

- **A design-by-composition framework for foldable robots.** We present a framework for robot design by composition, in which modules from a database of foldable designs are composed into full robots. The approach incorporates new foldable modules that enable a wide variety of robot kinematics, algorithms for geometry and motion composition, simulations and optimization methods for evaluating composed designs, and methods for automatic fabrication of the results.

- **A database of parameterized and foldable modules.** We explore the space of possible motions achievable through foldable designs by creating parameter-

ized designs of foldable joints. We have analyzed these joints theoretically and verified their behavior through experiments on fabricated examples. Together with fold patterns for rigid bodies, the parts in this database can be composed to produce a variety of robot geometries and functionalities.

- **An algorithm for fold pattern composition.** We detail an algorithm for combining two fold patterns into a new one-piece non-self-intersecting composition consisting of the two originals attached at an edge. The algorithm is guaranteed to produce a valid output fold pattern. It runs in time polynomial in the number of vertices in the original patterns and adds a polynomial amount of extra waste material. We also show how the algorithm can be augmented to compose patterns at faces instead of edges.

- **Grammar-based gait suggestion for ground robots.** We have imposed a classification of ground robot parts into bodies, legs, wheels, and peripherals, and we use this classification to propose a grammar-based gait for composed designs. Individual joint motions for a composed robot are suggested based on the types of modules connected at that joint. We demonstrate these gaits on a variety of composed robots.

- **Algorithms for automated generation of fabrication plans.** We describe methods for converting user-specified 3-D designs into valid fabrication plans. We have addressed a variety of origami-inspired methods for fabricating mechanical components at a variety of scales. We also describe an approach for converting designed gaits into electronics and software components.

- **An interactive end-to-end system for compositional robot design.** We have created an interactive tool in which users can manipulate and compose modules from the database to create custom ground robots. The tool incorporates the algorithms in this thesis to produce functional designs with valid fabrication plans. We have conducted a preliminary user study with the tool

and determined that users found interacting with the tool to be both intuitive and enjoyable.

- **Robot designs fabricated at multiple scales.** We have used the proposed framework to design and create robots ranging in size from 146 mm long to 620 mm long and fabricated using a variety of techniques. The robots are all demonstrably functional. Our results show that the design-by-composition framework has the potential to scale to many different applications.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, we compare our work to previous research in rapid fabrication, design methods, and origami-inspired robotics. In Chapter 3, we present important notation and symbols used in the thesis. In Chapter 4, we describe our foldable joint designs and their analysis. In Chapter 5, we discuss fold pattern composition and describe an algorithm for composing two fold patterns at an edge. In Chapter 6, we describe our grammar-based approach to gait suggestion for ground robots, our metrics for evaluating these gaits, and methods for converting these gaits into electronics and software. In Chapter 7, we describe an interactive tool that we have developed to aid casual users in composing foldable robot designs and a user study we conducted with the tool. In Chapter 8, we describe several results incorporating other fabrication approaches. We conclude and discuss directions of future research in Chapter 9.

Chapter 2

Related Work

This thesis is enabled by technical advances in the fields of rapid fabrication, computational design, and origami-inspired engineering, as well as by techniques in design composition. In this chapter, we outline previous results in the literature to set the context of our work.

2.1 Rapid Fabrication

With the growing interest from hobbyists and makers in creating custom robots and mechanisms [18, 56], there has been a corresponding boom in rapid fabrication and prototyping capabilities. Among these, 3-D printing has emerged as a convenient method for creating general geometries quickly. Unlike traditional manufacturing, in which complex geometries require in-depth analysis to ensure fabricability [70], 3-D printing processes are independent of the fabricated object, meaning that designers are now able to instantiate increasingly complex geometries without corresponding increases in cost or fabrication time. The most common methods for printing 3-D geometries are fused deposition modeling, in which spools of thermoplastic filament are melted and fused into layers of the fabricated solid object, and stereolithography, in which liquid material is cured using ultraviolet light and merged to form the object layers. A review of existing 3-D printing technologies can be found in [61, 201].

As a result of these technologies, multiple groups have started to investigate how mechanisms and linkages can be fabricated as single prints without requiring post-fabrication assembly. Work in [9, 24] explored a wide variety of printed joint designs, including universal and spherical joints, that could be used in full mechanisms [180], manipulators [107], and robots [110, 162]. Fuge et al. [56] expanded on this body of work by including methods for automated calibration of joint designs to the fabrication machine. Multi-material printers have further increased the diversity of printed objects by allowing designers to control properties such as stiffness [78], damping [105], and texture and transparency [27,189], giving them the ability to print such functional objects as flexure hinges [41, 63], hydraulic drive systems [104], and full robots [72].

2.1.1 Origami-Inspired Fabrication

Although recent advancements in machining practices and 3-D printing technology have produced significant speedup in manufacturing of mechanical structures, these methods are still costly and time consuming when compared with planar fabrication alternatives [132]. Origami-based design methods aim to augment existing 2-D fabrication techniques with folding algorithms to rapidly fabricate 3-D structures. Fabricating structures in plane and then folding them into their final shape enables complex devices to be created more quickly and efficiently, providing engineers with greater opportunities to prototype, test, and refine their designs.

Folding in engineering design has existed in mechanical designs for decades [179, 184] and more recently is being applied to electromechanical devices and foldable circuits [71, 131, 157]. The ability to integrate sensors and circuitry directly into the physical body of a device has enabled the creation of fully functional robots in less than a day [53, 74, 132]. In addition, folding thin materials can create strong, lightweight components, which are ideal for robot designs [127, 161]. Folding as a vehicle for physical change has given rise to sheets with programmable shape [71], manipulators [46,58,155,167,198,205], sensors [122,156], and support structures that precisely assemble small devices [197]. The transformation abilities of assembled folded structures have also been used to control how a robot crawls [44, 52, 54, 89],

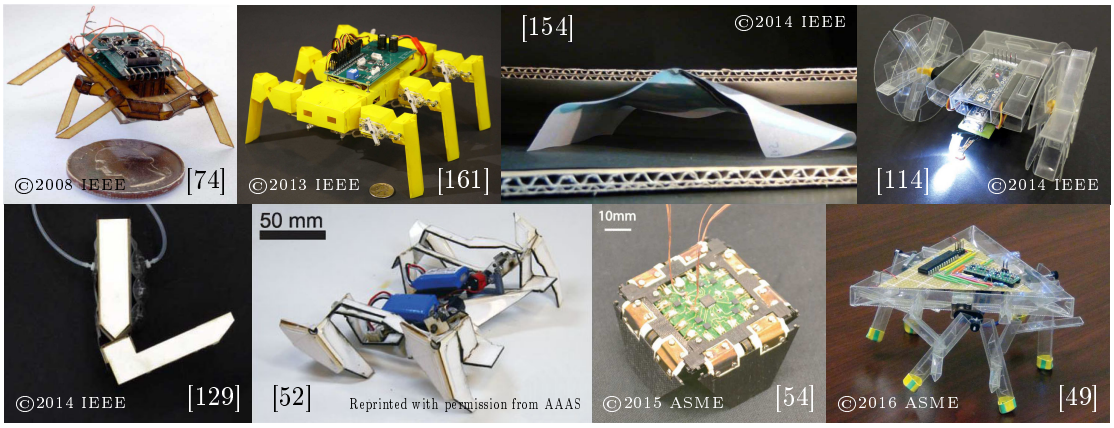


Figure 2-1: Previous foldable robots

jumps [50], or navigates obstacles [97]. Figure 2-1 shows some previously developed foldable robots.

In its most basic form, origami-inspired fabrication consists of cutting thin sheets of flexible material, such as paper or plastic, and deforming them into a 3-D shape. This method is often used when creating simple models [203] or paper toys [66, 120, 147, 177]. Stiffer designs can be created by laminating layers of different materials and cutting away parts of individual layers to create a structure combining both rigid and flexible parts [73, 74]. The approach has been used with great success to create folded structures with up to 15 layers [197] that assemble and self-align similarly to pop-up books, but it requires more complex methods for converting fold patterns into fabrication plans that take into account the thickness of the materials used [29, 93, 175]. Still more complex models can be created through a combination of additive manufacturing and folding, as shown in [40], where multi-material printers provided the designers with a simpler solution to mixing material properties to produce foldable structures.

The benefits of origami-inspired fabrication are tempered by the complexity of subsequent assembly steps. Folded structures often present more complex fabrication plans than the equivalent structure made through traditional 3-D manufacturing. As a result, many groups have also been searching for ways to automate the assembly of folded structures. For example, work in [12, 178] used a robotic manipulator to

fold origami patterns from a sheet of paper. Later, work in [124] took advantage of heating and gravitational forces to show how plastic sheets heated along a fold line could deform into a folded shape naturally. Work in [16, 28] bypassed the folding problem completely by cutting planar faces individually and assembling them using custom connectors.

Other approaches to automated folding focus on incorporating folding information into the structure itself. These "active origami" structures [137] rely on active materials to fold and unfold a pattern. The advantage of these approaches is that the same infrastructure used for assembly can also be used for actuation during operation [53]. Programmable sheets [4,71] encode generalized crease patterns with the ability to fold into arbitrary voxelized shapes using shape-memory alloys (SMA). Instance-specific approaches have also emerged, producing a variety of designs that fold using residual stress [13, 202], heat [41, 52, 53, 64, 90, 122, 182, 183], exposure to water [69, 154, 155], air pressure [129], and magnetic fields [84, 103, 204], particularly in the creation of microscale structures.

In this thesis, we primarily fold mechanical structures out of sheets of a single material, although we show extensions to other fabrication methods as well.

2.2 Computational Design of Robots and Functional Mechanisms

The incredible growth of rapid fabrication technology has incited a corresponding growth in design algorithms and tools. Approaches to design can be categorized into fully automated methods and interactive methods.

2.2.1 Fully Automated Design

Fully automated methods for robot design aim to create entire designs with minimal user intervention. These problems are generally formulated as optimizations over design and function parameters. In certain cases, the form of the robot is well

specified [42, 134, 191], and optimization methods are used to ensure efficiency. In other branches of work, users specify less about the robot. For example, work in [25] considered the robot design problem as a set of constraints on inputs, outputs, and inter-component dependencies, and it chose the components such as motors and chassis that satisfied the problem. It did not consider how the physical components should be connected. Work in [75,158,159], by comparison, focused on the physical structure over the actuation and used genetic algorithms to generate virtual linkage designs to perform various tasks such as locomotion and manipulation. The work was extended to fabricated robot designs in [30, 99].

The difficulty of designing entire robots from scratch has caused many research efforts to focus on automated design for particular subsystems and goals. For example, work in [82] considered modular robot design from a task perspective. A high-level task was broken down into a sequence of subtasks that were matched to existing full designs in a database, and transitions between subtasks occurred through reconfiguration. Motion plans were the subject of [92], which used linear temporal logic specifications to define high level task requirements and search for valid plans given a robot model and environment. Motion of a linkage mechanism was considered in [85, 87]. This work proved that there always exists a linkage able to follow a bounded subset of an algebraic curve and showed how to construct one. Finally, there exist systems that focus on fabrication aspects, such as those that automatically choose materials given desired properties [20, 27, 189] or that optimize geometry to account for particular fabrication methods [28, 100, 102].

2.2.2 Interactive Design

Compared to software compilers, or even work on automatic circuit design [126], fully automated mechatronic design systems have not gained wide acceptance in the engineering community. Progress on this front has been slowed by the high amount of coupling between different subcomponents of the same system [5, 51, 193, 194]. Therefore, more recent systems have been turning towards interactive design approaches that require more user input in the design process.

Interactive tools have been useful in designing a variety of physical objects, including architectural structures [190, 196], urban spaces [188], furniture [91, 96, 149, 152, 185], statuettes [128, 140], clothes [109, 186], toys [11, 123, 160, 166, 187], pop-ups [6, 7], and general geometries [59, 151]. This success has encouraged their use for mechanical assemblies as well. Previously, systems such as those in [48, 163] used textual interfaces to ask users a series of questions and produce linkages based on required inputs, outputs, and degrees of freedom. These systems used targeted queries to narrow down the space of designs to present to users. Since Mitra et al. proposed visual cues for mechanical assembly visualization in [121], multiple graphical tools have emerged for designing other types of mechanisms, including cam mechanisms [207], gear systems [34], linkages [10, 180, 206], and walking automata [19]. Reconfigurable objects such as furniture [62] and fold patterns [67, 80] have also been treated to not only analyses of their motion, but also checks for stability, collisions, and structural properties. The designed objects can be fabricated with various rapid fabrication methods by using complementary interactive tools that allow users to specify the fabrication method [16, 125] or object stiffness [200], and that break down the designs into parts that can be individually manufactured and assembled [101].

More recently, interactive approaches to design have been applied to robots and other mechatronic systems. Tools such as those in [17, 143] have aimed at addressing electromechanical constraints and producing user-customized but verifiable electronics plans. Simple programming interfaces integrated into the system allowed users to check their proposed connections against simulated behavior. Work in [111, 112] extended these methods to robot designs, allowing users to generate combined mechanical and electronic designs. Work in [45, 110] further added simulations enabling users to visualize robot behavior.

These existing tools provide users with the ability to specify designs but provide little guidance to users during the design process. In this thesis, we leverage a parameterized database to enable this feedback, allowing users to explore how design changes affect the simulated performance to produce more effective interactions.

2.3 Design by Composition

Simplifying design by composing existing modules is an approach that has been proposed for robot hardware before. Modular robots are systems of identical functional modules that can be connected together in various ways to form robots with complex functionalities [165]. Multiple types of modular robot hardware have been developed [22, 37, 65, 116, 144–146, 164, 195], but these systems have suffered from the idea that all modules must be the same. While using identical modules for all parts of a robot increases versatility and robustness, it also vastly increases the complexity requirements for individual modules.

An alternate approach is to compose modules of different types, each with their own intended functionality. In engineering practice, complex systems are commonly designed by breaking them down into smaller components that are matched to building blocks in a knowledge base [26, 32]. Systems such as Topobo [142], LEGO® Mindstorms® [68], and Fischertechnik® [55] emulate this process by providing users with existing hardware pieces that they can snap together to create interesting new designs. Parameterizing the pieces so that they can also be customized on the fly has proven to be an effective strategy for computational tools to provide users with creative freedom while maintaining a simplified design process [57]. As a result, design by composition has been used for such objects as furniture [91, 152], puppets [26], clothing [109], and electromechanical systems [17].

In robot control, composition of motion primitives is also an effective strategy for producing complex high-level behaviors and is often combined with hierarchical control. Work in [130] applied this approach to a hexapod robot, using a high level controller to determine the parameters for individual controller modules at each of the joints. Work in [181] used a similar approach for a manipulator arm, but rather than controlling individual joints, it used a low-level controller to decrease the effective dimensionality of the system and a high-level controller to set the values of this smaller set of parameters.

Systems that leverage a design-by-composition approach not only simplify the design space but have also been shown to facilitate rapid fabrication. For example, in [106], an automated pick-and-place machine was used to assemble a carefully chosen set of electromechanical building blocks. Similarly, in [111], a composition approach was proposed for foldable robot design that automatically generated the fold pattern for the robot body, although no guarantees or analysis of the designs were provided. In [45], existing hardware was combined with 3-D printed joints and freeform components to create customized multicopters that achieved stable flight. At a smaller scale, composition of microstructures in 3-D printing allows digital fabrication methods to produce material properties not achievable through solid blocks of a single material alone [27, 189].

In this thesis, we explore design of foldable robots by composition, as well as algorithms and analysis methods that can be incorporated into the design process to produce robots of guaranteed validity.

2.4 Origami-Inspired Engineering

Origami-inspired fabrication approaches introduce additional complexity to the design process. In particular, fabricated objects must have valid fold patterns. For static objects, this is a well studied problem, while questions about transformable structures remain largely unanswered.

2.4.1 Static Structures

The problem of edge-unfolding polyhedra involves flattening a polyhedron by cutting along some of its edges and unfolding the rest (See [39] for a survey of results). The resulting planar figure should be a simple, non-overlapping polygon. A number of algorithms have been proposed for approaching this problem (ref. [150]), although none succeed in the general case. Since the traditional problem statement for edge unfolding requires that every face be covered exactly once and that no extra material be added, there exist polyhedra such as that in [15] for which an edge unfolding

does not exist. As a result, some heuristics [120, 177] have been developed that allow multiple pieces to be used or minor changes to the 3-D geometry to be made.

In contrast, extra material can be added in origami design problems [94, 95, 173], where an arbitrary polyhedral surface is folded from a convex 2-D sheet of paper. Work in [148] characterized these types of fold patterns and provided necessary conditions for the patterns to be foldable, and systems in [67, 80] provided methods for users to design, simulate, and visualize new fold patterns. Algorithms for automatically constructing designs have been proposed on multiple fronts. In [173, 174], arbitrary simple polyhedral surfaces were converted into origami designs by positioning faces of the surface on the 2-D plane and tucking away excess material to bring neighboring faces together. Composition of origami designs is an idea that appeared in [117, 118], in which rotational symmetry allowed the goal surface to be decomposed into slices, each of which could be achieved by the same fold pattern. Similarly, Cheng and Cheong [31] decomposed a surface into horizontal slices and construct its unfolding one level at a time. Both of these algorithms were limited in the types of surfaces that they could produce.

In practical design, additional constraints also have to be met. For example, objects folded from sheet metal have constraints on bend radius or design complexity [35]. To deal with these constraints, work in [23] proposed a multi-stage approach for generating a sheet metal part that covers all user-specified regions while avoiding user-specified obstacles. In [135, 136], the authors solved a similar problem and enforced manufacturability of the part by generating only designs that could be produced using a set of allowed sheet metal operations. In both of these cases, the user inputted only constraints on what should or should not be covered, but the actual shape of the part was left to the algorithm, allowing greater flexibility to reduce cost and manufacturing time. Wang [192] considered the case where the desired shape of the sheet metal part was entirely known. When the shape was not manufacturable as a single piece, then it was decomposed into simpler manufacturable components that could be assembled via welding, riveting, etc.

2.4.2 Transformable Structures

For foldable robots, it is desirable that fold patterns produce mechanisms, that is, structures that move. Unfortunately, the space of transformable folded structures is not well characterized. Exploration of transformable folded structures seems to have been limited to single designs that are created by individuals (ref. Section 2.1.1) are are difficult to generalize. Progress beyond these structures is complicated by a lack of understanding of what types of motions can result from folding. The most promising results have come from the study of pop-ups. For example, algorithms proposed by [1] demonstrated how to design pop-ups for general polygons, and work in [98,119] tackled automated pop-up design for models of buildings. Pop-up mechanisms were analyzed and categorized in [199], and the results were used to propose new pop-up joint types. In contrast, a similar analysis of general folded designs as spherical mechanisms was performed in [21], but although many models from popular literature were classified, no new principles were derived or new mechanisms found.

A small set of parameterized transformable designs can be found in the literature. For example, work in [60] proposed a new rotational joint type that can be composed into closed loops of arbitrary size. A collapsible wheel was proposed in [97] to enable robots to shrink in height and roll under obstacles. A continuously foldable cylinder that could be tessellated to produce cellular structures that expand and contract was demonstrated in [176], and work in [133] showed a 3-D analog with three degrees of freedom. Work in [86] demonstrated a pattern based on the Kresling pattern for controlling not only the shape but also the stiffness of cylindrical structures. Finally, work in [49, 113] suggested methods for combining designs for parameterized rigid robot body parts to form entire robot designs.

Although these pieces of work present interesting new degrees of freedom for robots, none of them systematically explores the foldable mechanism design space. In this thesis, we propose parameterized joint designs for joints commonly used in robot designs, as well as methods for them to be combined to create more complex joints and full mechanisms.

Chapter 3

Notation

This chapter provides a brief summary of the notation and symbols used in the thesis.

Chapters 4 and 5 are primarily concerned with fold patterns. Briefly, a fold pattern $(P, \mathcal{F})$ consists of a non-self-intersection 2-D polygon P and a set of folds $\mathcal{F}$. Each fold $f \in \mathcal{F}$ is a line segment on the interior of P and is associated with a fold angle. The 3-D shape that results from bending the fold pattern at each of the folds is called the folded state. The following symbols refer to properties of a fold pattern or a folded state.

Table 3.1: Symbols pertaining to fold patterns

Symbol	**Definition**
$(P, \mathcal{F})$	A fold pattern consisting of a polygon P and a set of folds $\mathcal{F}$
e	An edge of an unfolding $(P, \mathcal{F})$
$f = (e, \phi)$	A fold consisting of a fold line e and a fold angle ϕ
$\mathcal{E}(P, \mathcal{F})$	Edge set of the fold pattern $(P, \mathcal{F})$
$\mathcal{V}(P, \mathcal{F})$	Vertex set of the fold pattern $(P, \mathcal{F})$
N	Number of vertices
M	Number of edges
$\mathrm{CH}(P)$	Convex hull of the polygon P
Q	Polyhedral complex, image of a folded state
$\mathcal{E}(Q)$	Edge set of the polyhedral complex Q
$\mathcal{V}(Q)$	Vertex set of the polyhedral complex Q
$\mathbf{p}$	A path defined by vertices $v_1, v_2, \ldots v_n$

Additionally, Chapter 4 presents foldable rotational and translational joint designs and a model of the mechanics of the proposed designs. The following symbols are relevant to these joints.

Table 3.2: Symbols pertaining to foldable joint designs

Symbol	Definition
R	Joint range of motion
N_s	Number of sides of joint
N_ℓ	Number of layers of joint
r, r_i, r_o	Radius, inner radius, and outer radius of rotational joints
d, w, h	Depth, width, and height of one segment of prismatic joint
k_i	Stiffness of fold i
θ	Rotational joint angle
τ_θ	Holding torque required to maintain joint at angle θ
x	Prismatic joint position
R_x	Holding force required to maintain joint at position x

Chapters 6 and 7 discuss designing robots within the composition framework. A robot design $\mathbf{D}$ consists of a hierarchy of connected modules with geometric parameters $\mathbf{q}$ and gait parameters θ_i and g_i. The robot designs are evaluated by discretizing time, iterating through a sequence of rigid body simulations, and repeating the gait until the robot reaches a steady state behavior. The following symbols refer to a robot design and the parameters related to its geometry, gait, or simulation.

Table 3.3: Symbols pertaining to robot design

Symbol	Definition
$\mathbf{D}$	Robot design
$\mathbf{q}$	Geometric parameters of a robot design
Q	Feasible set for parameters of a robot design
N_g	Number of steps in a gait
g_i	Step parameter for limb i in a gait
θ_i	Angle parameters for limb i in a gait
Δt	Time step used for robot simulation
$\mathbf{c}_i$	Contact points of robot with the ground at time t_i
t_{i_0}	Start time for robot steady state behavior
t_{i_f}	End time for robot steady state behavior

Our system evaluates multiple performance metrics during the simulation and reports these metrics to the designer. The following symbols are the relevant variables and metrics used.

Table 3.4: Symbols pertaining to robot performance metrics

Symbol	Definition
$\mathbf{x}(t)$	Position of robot center of mass projected on ground at time t
$\tilde{\mathbf{x}}(t)$	Expected position of robot at time t given a circular trajectory
S	Stability (mm)
V	Velocity (mm/s)
W	Wobbliness (radians)
$\Delta\gamma$	Combined pitch-roll angular change (radians)
E	Slip (mm)
φ	Heading (radians)
ρ	Curvature (1/m)
σ^2	Variance (mm^2)

Finally, Chapter 8 discusses alternate fabrication methods. The following symbols refer to variables used in the generation of the fabrication plans for these methods.

Table 3.5: Symbols pertaining to fabrication

Symbol	Definition
t	Thickness of material used
ℓ_i, ℓ_o	Inner and outer positions for teeth generation along a fold
w_g	Gap width between two faces of a fold pattern

Chapter 4

Transformable Foldable Modules

In a design-by-composition approach, a key element is the modules that make up the database. For robots, those modules must consist of both rigid components and movable joints. However, as discussed in Chapter 2, transformable folded structures have not been well studied, and the space of transformable foldable designs remains uncharacterized.

In this chapter, we address the question of what motions are achievable through folding, and we contribute fold patterns for revolute and prismatic joints that are parameterized to achieve a user-specified size and range of motion. The designs provide users with greater flexibility in fold pattern design compared to previous modules [60, 113]. Since the joints are meant to be used in foldable robots, all of the designs provide natural attachment points for other modules, as well as space for actuators. We have fabricated our joints and compared their force-displacement curves to a pseudo-rigid-body model [77]. Together with rigid shapes, these joints form a base set of modules from which robots with any desired kinematics can be created out of a folded sheet.[1]

[1] The majority of this chapter was published in [170, 172].

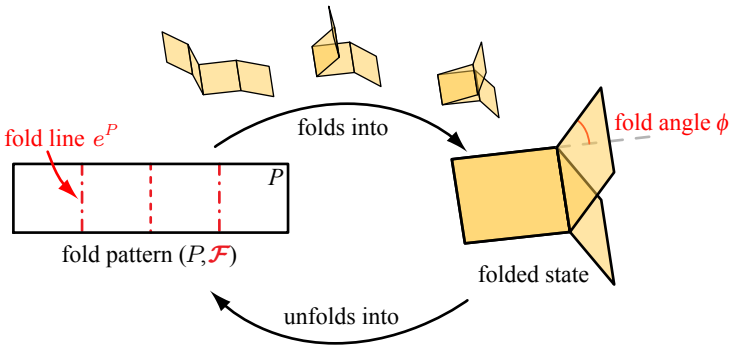


Figure 4-1: A fold pattern $(P, \mathcal{F})$ consists of a polygon P and a set of folds, each of which has a fold line and a fold angle. The pattern folds into a 3-D folded state.

4.1 Parametric Designs

Our mechanisms and robots are based on three joint types, a hinge joint, a prismatic joint, and a pivot joint. In this section, we present the parameterized fold patterns for each.

4.1.1 Definitions

We begin with informal definitions for the terms used in the following descriptions. A more formal treatment can be found in [39]. Consider a non-self-intersecting 2-D polygon P, possibly with holes. A *fold line* on P is a line segment such that both endpoints are on the boundary of P and the segment itself lies on the interior of P. A *fold pattern* $(P, \mathcal{F})$ consists of such a polygon P and a set of folds $\mathcal{F}$ on P. Each fold is a fold line associated with a fold angle in the range $[-\pi, \pi]$. The folds in a fold pattern divide the original polygon P into *faces*, smaller polygons that overlap only at the fold lines. The *folded state* is the 3-D shape resulting from bending the fold pattern to the given fold angle at each of the folds and keeping the faces flat (ref. Figure 4-1). The folded state must be non-crossing and should also be distance preserving (i.e., not stretch or tear the material).

When all fold angles are single values, then a fold pattern produces a static structure. Positive fold angles correspond to what we common term valley folds, and negative fold angles are mountain folds. Since we are interested in patterns for joints,

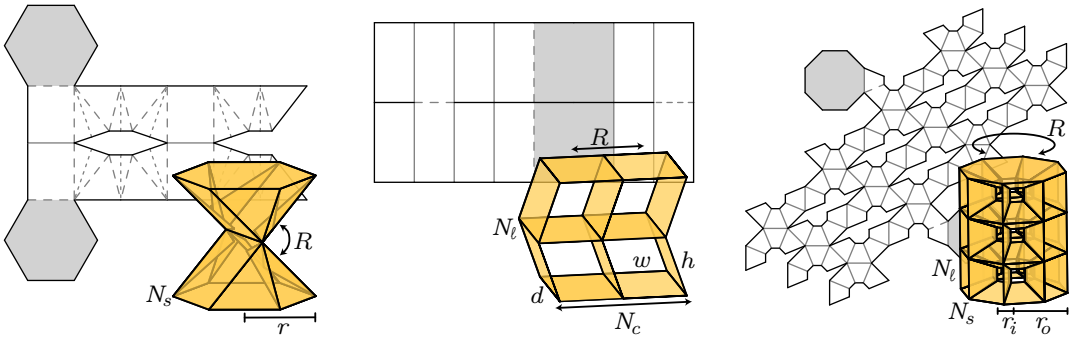
which allow movement, some folds must be able to achieve entire fold angle ranges, which are connected subsets of $[-\pi, \pi]$. We call these folds the *active* folds. In this thesis, figures depicting fold patterns will use the convention that active folds are drawn as solid lines, valley folds as dashed lines, mountain folds as dashed-dotted lines, and cut lines forming the boundary of P as solid black lines.

Since the purpose of joints is to connect other structures to each other, our joints have faces that exist specifically to attach to other bodies. We call these faces the *bases* of the joint. All of our joints are designed to connect two structures to each other and so each have two bases.

4.1.2 Hinge Joint

Hinge joints enable rotation about an axis parallel to a base and can be implemented as a single active fold on the base itself. This is the approach taken in many previously designed foldable robots [49, 113, 161]. However, when hinge joints are created in this way, the joint itself consists of a single fold and occupies zero volume. As a result, the joint limits depend on the geometry of the bodies being connected and cannot be independently specified.

Joint description We have designed a hinge joint of a more general form, shown in Figure 4-2(a). The base of the joint is a regular polygon with N_s sides ($N_s = 6$ in



(a) Hinge joint (b) Prismatic joint (c) Pivot joint

Figure 4-2: Summary of fold patterns and folded states for three basic joint types with input parameters indicated. The base faces are shaded gray.

the example), where N_s is even. From two opposite sides, sloped faces angle to meet at the axis of rotation. Triangular faces are attached to all other sides of the bases to provide structural support.

When the joint moves, the top base rotates about the axis of rotation relative to the bottom base. The limits of the rotational motion are when two rectangular faces touch. Thus, the angle between a rectangular face and the base it is connected to is determined by the joint limits as $\frac{R}{4}$, where R is the total desired angular range of motion.

Fold pattern The associated fold pattern is a strip that attaches to the outer edges of the base polygons and contains the rectangular and triangular faces. Additional folds that tuck away extra material help form the hinge shape. Input parameters to the pattern are the number of sides N_s, the radius r of the base, and the total range of motion R. The basic hinge joint design is always symmetric. If asymmetric joint limits are desired, users can extend the base by attaching a sloped polyhedron and tilting the joint. Note that the joint angle is restricted to be between $-\pi$ and π radians, and that the length of the joint increases with the range of motion.

4.1.3 Prismatic Joint

Joint description A prismatic joint is created by composing parallelogram linkages to produce linear motion of a desired distance without increasing the size of the joint. Each base of the joint is a rectangular face that forms one side of a parallelogram linkage. In a single parallelogram linkage, horizontal and vertical translation are coupled. By connecting linkages in a grid, these two degrees of freedom can be decoupled. Figure 4-2(b) shows a two-by-two grid of linkages of height h. By restricting horizontal motion, the joint enables vertical translation by as much as $2h$. By restricting the vertical distance between the bases to be h, the joint enables horizontal translation by a distance $h\sqrt{3}$ in either direction.

The resulting structure enables 2 degree-of-freedom motion in a plane. To produce a true prismatic joint, which allows linear motion in one direction but not in any other,

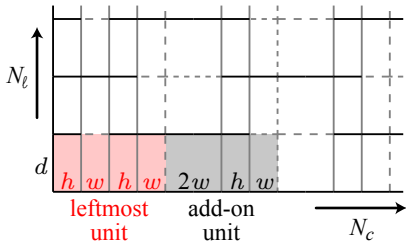


Figure 4-3: Prismatic joint construction

we impose a height constraint to the joint using additional faces added in a subsequent stage (ref. mechanism in Figure 5-12). This yields a joint with translational motion parallel to the base.

Fold pattern The fold pattern for the grid of parallelogram linkages is a grid of rectangular faces, built by repeating and connecting units as shown in Figure 4-3. On the leftmost side is a unit consisting of four faces that fold into a parallelogram linkage. To add columns, three-face add-on units are attached to the right, with the fold angles of each unit opposite in sign to the one before. For each layer, the entire row of units is duplicated and attached above the previous layer to the faces corresponding to the top link of the linkage below. Input parameters are the dimensions h, w, and d of one linkage in the grid, the joint's horizontal range of motion R, and the number of columns N_c in the grid. Assuming that the height of the linkage is constrained to be h, the theoretical number of layers N_ℓ can be computed using the relationship $R = 2h\sqrt{N_\ell^2 - 1}$, although practically we approximate N_ℓ using the simpler expression $R = 2h\,(N_\ell - 1)$. The joint's range of motion can be increased by changing either h or N_ℓ.

4.1.4 Pivot Joint

Joint description Pivot joints allow rotation about an axis perpendicular to the base. To achieve this twisting motion, we use spherical parallelogram linkages arranged in a cycle. Figure 4-4 shows an example of 5 folded square linkages in this configuration. The black outlines indicate the corresponding 3-D linkage, and the

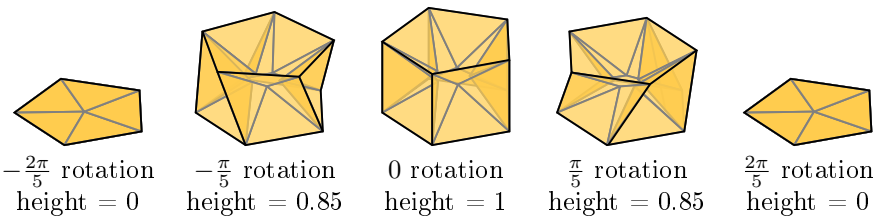


Figure 4-4: Motion of 1 layer of a pivot joint with all black lines having unit length

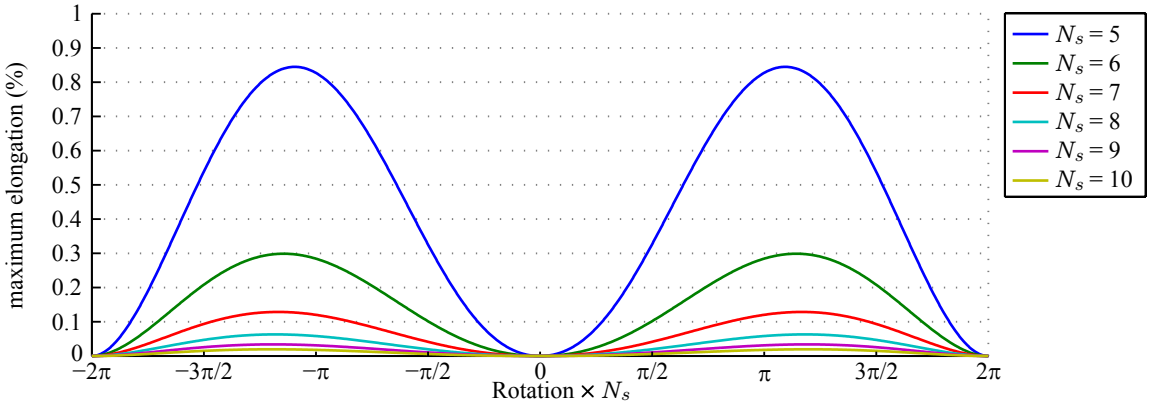


Figure 4-5: Elongation of folds in a pivot joint over the course of rotation for N_s between 5 and 10. Maximum elongation decreases as N_s increases.

gray lines indicate folds, which correspond to the axes of rotation for the joints of the black linkage. When this arrangement of faces is used, a twist-and-collapse motion can be performed. Note that this motion is theoretically not rigid since it requires elongation of some of the fold lines. It is possible to create a rigidly foldable structure by adding triangular pleats to the corners of each of the square linkages to allow stretch. Since our simulations show that the amount of elongation is at most 1% (ref. Figure 4-5) and foldable robots have generally been fabricated from flexible materials, we use the simpler design shown and accommodate some stretch by replacing one fold out of the four in each square linkage with a cut. The bases of the pivot joint are regular polygons, and each side is attached to one square linkage.

In the same way as for the prismatic joint, we stack linkages in series to decouple vertical translation from twisting motion. Figure 4-2(c) shows an 8-sided 3-layer stacked linkage structure. When the relative rotation of the two bases is maintained at 0, the linkage can achieve a vertical translation of up to $2N_\ell r_o \sin\frac{\pi}{N_s}$. Restricting the distance between the two bases to be the height of one layer using additional

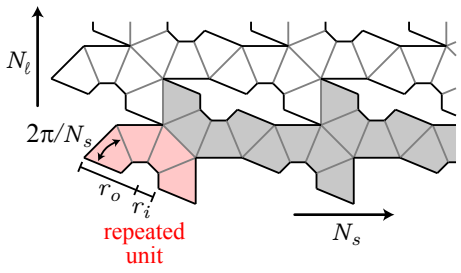


Figure 4-6: Pivot joint construction

faces enables pure twisting motion. We add faces for height constraints in subsequent steps.

Fold pattern The pattern is built similarly to the prismatic joint, by repeating and connecting identical units. The unit, shown in Figure 4-6, consists of four isosceles trapezoids, each with an angle equal to $\frac{2\pi}{N_s}$ between the legs of the trapezoid, connected along the legs. A total of N_s units are attached to each other at the faces corresponding to the side links of the linkages in order to produce the N_s linkages that form one layer of the joint (shaded gray). The resulting strip of units is duplicated once for each layer and attached to the adjacent layers using the faces corresponding to the top and bottom links of the linkages. Input parameters to this design are the number of sides $N_s > 4$ of the base, the range of motion R, and the inner and outer radii r_i and r_o of the joint. Similarly to the prismatic joint, we approximate the number of layers N_ℓ using the expression $R = \frac{4\pi}{N_s}(N_\ell - 1)$. In Figure 4-2(c), the joint has 8 sides and 3 layers and can twist π radians.

4.2 Pseudo-Rigid Body Analysis

Folded structures are compliant mechanisms whose faces and folds all deform with external forces. However, such compliant mechanisms can be modeled as mechanisms where rigid faces are joined at hinges with torsion springs [76, 77], a method that was applied to folded cartons in [36, 141]. In this section, we present a model for our folded joints, which we later compare to experimental measurements.

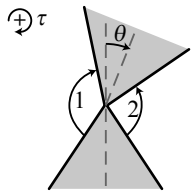


Figure 4-7: Cross-sectional view of hinge joint with active folds labeled

4.2.1 Hinge Joint

A hinge joint is two rigid bodies connected at two active folds. Figure 4-7 shows a cross-sectional view of the hinge joint. The angle θ is the joint angle and folds 1 and 2 are the active folds. Let ϕ_1 and ϕ_2 represent the fold angles of the folds 1 and 2 respectively. These fold angles are functions of the joint angle

$$\phi_1 = R/2 + \theta \tag{4.1}$$

$$\phi_2 = R/2 - \theta \tag{4.2}$$

where R is the range of motion of the joint.

Then the holding torque τ_θ necessary to maintain the joint at an angle θ is

$$\tau_\theta = -k_1 \left(\phi_1 - \phi_1^o\right) + k_2 \left(\phi_2 - \phi_2^o\right) \tag{4.3}$$

where stiffnesses k_1 and k_2 may be functions of θ and ϕ_1^o and ϕ_2^o are the equilibrium positions of the active folds. When the stiffness $k_1 = k_2 = k$ and using Equations (4.1) and (4.2), this expression simplifies to

$$\tau_\theta = -2k \left(\theta - \frac{\phi_1^o - \phi_2^o}{2}\right) \tag{4.4}$$

That is to say, the joint as a whole acts as a torsion spring with spring constant equal to the sum of the spring constants of the active folds.

Since each of our joints are constructed from a single sheet of material, the stiffness k as a function of θ can be computed as [76]

$$k = K_\Theta \frac{EI}{\ell} \tag{4.5}$$

where K_Θ is the nondimensionalized stiffness, E is the Young's modulus of the material, I is the area moment of inertia of the section of material involved in the fold, and ℓ is the length of the pivot. Since the folds are perforated using the laser cutter, it is difficult to calculate the moment of inertia I for a fold exactly. However, we expect a constant proportion of the material is removed during perforation, so

$$I \propto \ell_f t^3 \tag{4.6}$$

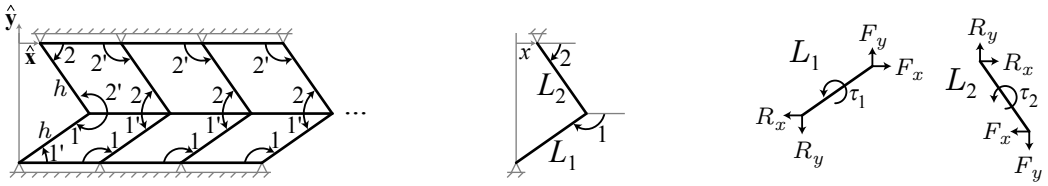
where ℓ_f is the length of the fold and t is the thickness of the material. That is, for a given material,

$$k \propto \ell_f t^3 \tag{4.7}$$

4.2.2 Prismatic Joints

Prismatic joints are formed by stacking parallelogram units. We first consider a 2-layer prismatic joint. To formulate the pseudo-rigid-body model, we use the equivalent linkage shown in Figure 4-8(a), where the fold angles of the active folds are labeled. Since each unit is a parallelogram, all folds labeled 1 must have the same fold angle. In addition, it is reasonable to assume for the sake of simplicity that all of these folds labeled 1 have the same stiffness and equilibrium position since they are all fabricated identically and they all experience the same motions. This is similarly true for the folds labeled 1', 2, and 2' respectively.

In that case, it is possible to simplify the prismatic joint to the diagram in Figure 4-8(b), where the folds in the bottom row of units are reduced to a single



(a) $N_\ell = 2$ (b) Simplified diagram (c) Individual links

Figure 4-8: (a) Diagram of 2-layer prismatic joint with active folds labeled. (b) Simplified linkage diagram. (c) Free-body diagram for individual links in (b).

equivalent active fold that produces a torque

$$\tau_1 = k_1 \left(\phi_1 - \phi_1^o\right) \tag{4.8}$$

on link L_1, where ϕ_1 is the angle at fold 1, and the top row of units are reduced to a single active fold that produces a torque

$$\tau_2 = k_2 \left(\phi_2 - \phi_2^o\right) \tag{4.9}$$

on link L_2, where ϕ_2 is the angle at fold 2. The distance x is the joint position. To enable horizontal translation, the total height of the linkage is maintained at h, so the following relations must hold.

$$h\cos\phi_1 + h\cos\phi_2 = x \tag{4.10}$$

$$h\sin\phi_1 + h\sin\phi_2 = h \tag{4.11}$$

To compute the holding force R_x necessary to maintain the joint at a specific position x, we use the following force balance equations. For link L_1,

$$F_x - R_x = 0 \tag{4.12}$$

$$F_y - R_y = 0 \tag{4.13}$$

$$\tau_1 - hR_x\sin\phi_1 - hR_y\cos\phi_1 = 0 \tag{4.14}$$

Similarly, for link L_2,

$$\tau_2 - hR_x \sin\phi_2 - hR_y \cos\phi_2 = 0 \tag{4.15}$$

So the holding force R_x can be found by solving the following matrix equation for F_x.

$$h \begin{bmatrix} \sin\phi_1 & \cos\phi_1 \\ \sin\phi_2 & \cos\phi_2 \end{bmatrix} \begin{bmatrix} R_x \\ R_y \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{4.16}$$

It is simple to see that adding more layers to the prismatic joint just adds more rows to the matrix equation and more variables to the distance constraints. Thus the relationship between holding force and fold torques is

$$h \begin{bmatrix} \sin\phi_1 & \cos\phi_1 \\ \sin\phi_2 & \cos\phi_2 \\ \sin\phi_3 & \cos\phi_3 \\ \cdots & \cdots \end{bmatrix} \begin{bmatrix} R_x \\ R_y \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \cdots \end{bmatrix} \tag{4.17}$$

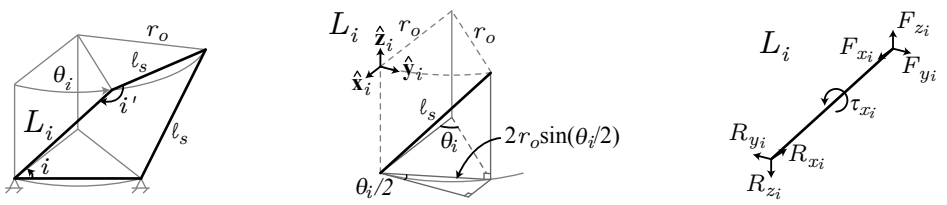
subject to

$$\sum_{i=1}^{N_\ell} h\cos\phi_i = x \tag{4.18}$$

$$\sum_{i=1}^{N_\ell} h\sin\phi_i = h \tag{4.19}$$

4.2.3 Pivot Joint

Since the pivot joint consists of layers of parallelogram units similarly to the prismatic joint, we expect it to follow a similar model. In particular, Figure 4-9(a) shows a single square linkage on a pivot joint. Since the joint is rotationally symmetric, the analysis for every link L_i in a single layer of the joint is the same. Let ϕ_i be the measure of fold angle i as indicated in the figure, and let $\theta_i(\phi_i)$ denote the rotation about the center axis of the pivot joint that is contributed by layer i. The length $\ell_s = 2r_o \sin\frac{\pi}{N_s}$



(a) One square linkage (b) Link L_i geometry (c) Link L_i

Figure 4-9: (a) Diagram of a single square linkage on a pivot joint. (b) Relevant dimensions for analysis of link L_i. (c) Free-body diagram of link L_i.

is the length of one side of the pivot joint and also the length of each link on the corresponding linkage.

To construct a pseudo-rigid-body model, we approximate the torque on the link L_i contributed by the folds at i and i' as

$$\tau_i = k_i \left(\phi_i - \phi_i^o\right) \tag{4.20}$$

and assume all links rigid [76]. The height of layer i is $\sqrt{\ell_s^2 - \left(2r_o \sin\frac{\theta_i(\phi_i)}{2}\right)^2} = 2r_o\sqrt{\sin^2\frac{\pi}{N} - \sin^2\frac{\theta_i(\phi_i)}{2}}$. When the joint is constrained to a total height of ℓ_s, the following geometrical relations must hold:

$$\sum_{i=1}^{N_\ell} \theta_i(\phi_i) = \theta \tag{4.21}$$

$$2r_o \sum_{i=1}^{N_\ell} \sqrt{\sin^2\frac{\pi}{N} - \sin^2\frac{\theta_i(\phi_i)}{2}} = \ell_s \tag{4.22}$$

To calculate the holding torque, we use the free-body diagram of link L_i shown in Figure 4-9(c). The joint produces rotational motion. For a single link, we align the $\hat{\mathbf{x}}_i$ direction radially outward from the center of the joint, according to the bottom attachment point, the $\hat{\mathbf{z}}_i$ direction parallel to the center axis of the joint, and the $\hat{\mathbf{y}}_i$ direction according to the right-hand convention. Then the force balance equations

are as follows:

$$F_{x_i} - R_{x_i} = 0 \tag{4.23}$$

$$F_{y_i} - R_{y_i} = 0 \tag{4.24}$$

$$F_{z_i} - R_{z_i} = 0 \tag{4.25}$$

$$\tau_{x_i} - 2r_o R_{y_i} \sqrt{\sin^2 \frac{\pi}{N} - \sin^2 \frac{\theta_i(\phi_i)}{2}} - r_o R_{z_i} \sin\theta_i(\phi_i) = 0 \tag{4.26}$$

The total holding torque for that layer is equal to $\tau_{\theta_i} = N_s r_o R_{y_i}$.

Between adjacent layers, the holding torque for those two layers must be equal at equilibrium

$$\tau_{\theta_i} = \tau_{\theta_{i-1}} \tag{4.27}$$

which implies $R_{y_i} = R_{y_{i-1}} = R_y$.

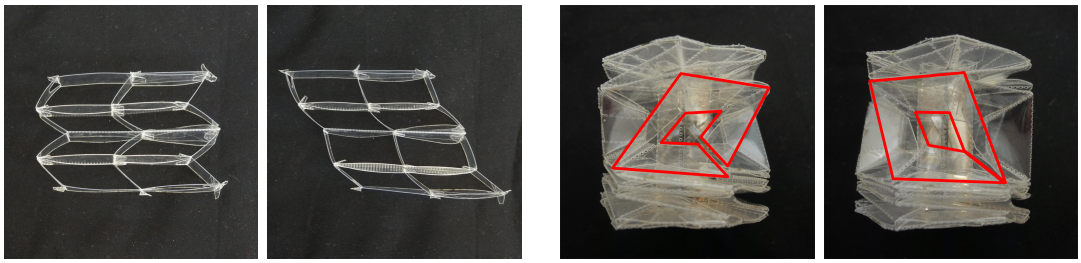
We can therefore compute the reaction forces by solving the following matrix equation

$$r_o \begin{bmatrix} 2\sqrt{\sin^2 \frac{\pi}{N} - \sin^2 \frac{\theta_1(\phi_1)}{2}} & \sin\theta_1(\phi_1) \\ 2\sqrt{\sin^2 \frac{\pi}{N} - \sin^2 \frac{\theta_2(\phi_2)}{2}} & \sin\theta_2(\phi_2) \\ 2\sqrt{\sin^2 \frac{\pi}{N} - \sin^2 \frac{\theta_2(\phi_2)}{2}} & \sin\theta_3(\phi_3) \\ \cdots & \cdots \end{bmatrix} \begin{bmatrix} R_y \\ R_z \end{bmatrix} = \begin{bmatrix} \tau_{x_1} \\ \tau_{x_2} \\ \tau_{x_3} \\ \cdots \end{bmatrix} \tag{4.28}$$

subject to constraints Equations (4.21) and (4.22). The net holding torque for the joint is then equal to $\tau_\theta = N_s r_o R_y$.

4.3 Experimental Comparisons

We have generated fold patterns for our basic joints using multiple different parameters and constructed them out of 0.051 mm and 0.127 mm thick polyester film, cutting them using a laser cutter and perforating the folds for easier assembly. Before printing, we added tabs and slots to the pattern to attach edges that should remain coincident in the folded state.



(a) Prismatic joint (b) Pivot joint

Figure 4-10: Joints folded from polyester film in two different positions

Figure 4-10 shows two of our folded joints. Figure 4-10(a) shows the result for a $N_c = 2$, $N_\ell = 4$ prismatic joint with every layer 15 mm tall. Without height constraints, it is capable of 80 mm horizontal motion and 60 mm vertical motion. Figure 4-10(b) shows a 6-sided pivot joint with $r_o = 44$ mm and a 2π radian (360°) range of motion. The faces corresponding to one square linkage are outlined. The joint has 4 layers and is constrained in height by additional faces subsequently added to the fold pattern. In particular, we added a tube of the desired height down the center of the joint; we attached it to the base on one side of the joint and to an added perpendicular face on the other side (ref. Figure 5-10 for example fold pattern). In this way, the bases can rotate relative to each other but cannot move farther apart than the length of the tube. Since plastic film has thickness, adding layers to increase a joint's range of motion increases the size of the joint: each layer adds the thickness of five sheets of plastic in the case of the prismatic joint, and four sheets of plastic for the pivot joint. For the joint in Figure 4-10(b), the additional thickness corresponds to almost half the joint length. This is not a concern for the hinge joint, which does not rely on layers to control the range of motion.

4.3.1 Electronics Integration

Print-and-fold manufacturing enables using a single uniform process to fabricate entire robots. Actuation, sensing, and computation can be simultaneously incorporated into a robot during the fabrication process by printing circuitry and mounting components directly onto the fold pattern before folding. All of our joints provide natural

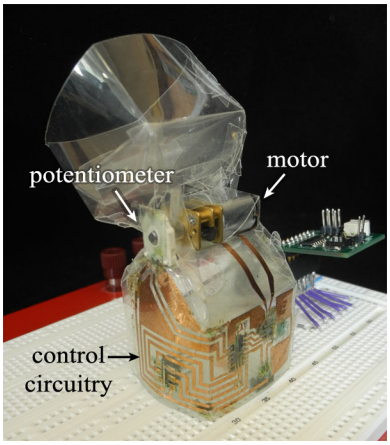


Figure 4-11: Hinge joint with integrated electronics

placement for actuators and circuitry to be integrated into the folded structure, obviating the need for a post-folding stage of attaching circuit boards, actuators, and additional wires.

We used the following procedure to fabricate an actuated hinge joint. We used the design for a 4-sided π radian (180°) hinge joint and added a motor and potentiometer, as well as the control circuitry of a standard servomotor, to produce a hinge joint with position control. We placed the motor at the center of the hinge joint, with the output shaft aligned with the axis of rotation and directly connected to the input of the potentiometer. We manually designed a motor mount to keep the motor in place. The control circuit was designed by hand and line the faces of the bottom half of the joint.

Circuit traces were printed on copper tape using a solid-ink printer. The tape was then affixed to a sheet of 0.127 mm thick polyester film, and the circuit etched out using a ferric chloride solution. The fold pattern was cut out and fold lines perforated on the reverse side of the polyester film using a laser cutter. Actuators, sensors, and other circuit components were soldered directly onto the circuit traces by hand. Finally, the device was folded into shape.

The final hinge is pictured in Figure 4-11. Despite the motor and potentiometer leads being on opposite sides of the hinge joint, both components were able to be soldered directly into the circuit without additional wires. We sent a PWM signal to

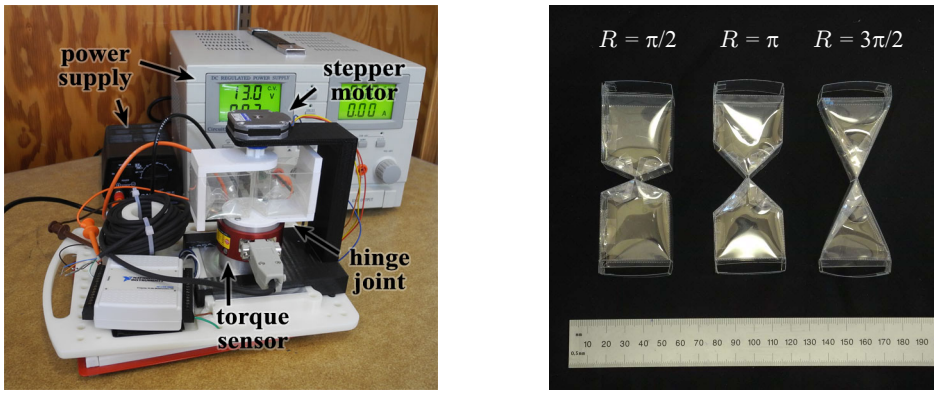
the microcontroller on the joint to control its angle. The joint was able to achieve the entire π radian (180°) range for which it was designed. This joint demonstrates the feasibility of integrating sensors and actuators directly with our fold patterns to produce foldable robots.

4.3.2 Force and Torque Measurements

To check the spring-based models derived in Section 4.2, we generated and folded designs for multiple hinge, prismatic, and pivot joints and measured the holding forces and torques for a range of positions.

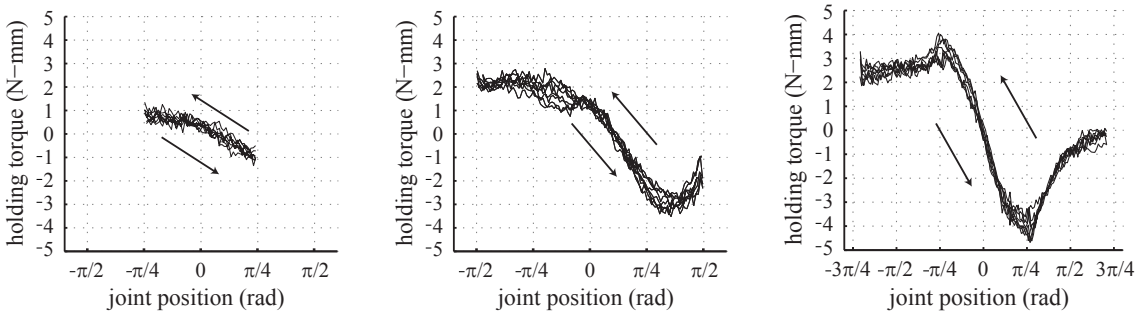
Hinge Joint

For the hinge joint, we generated three hinges with 4 sides ($r = 25$ mm) and varying ranges of motion: $\frac{\pi}{2}$ radians, π radians, and $\frac{3\pi}{2}$ radians. We folded each design out of both 0.051 mm and 0.127 mm thick polyester film and measured the holding torque of each joint over its entire range of motion. Figure 4-12 shows the experimental setup and the three joint designs tested. A Futek TFF500 torque sensor was attached to one base of the hinge joint and measured the torque as a stepper motor moved the other base. A LABVIEW program was used to control the stepper motor and to record torque measurements at 1 kHz. Each hinge was moved one step (200 steps/rev.) every 50 ms, first in a clockwise direction to the positive joint limit, then counterclockwise



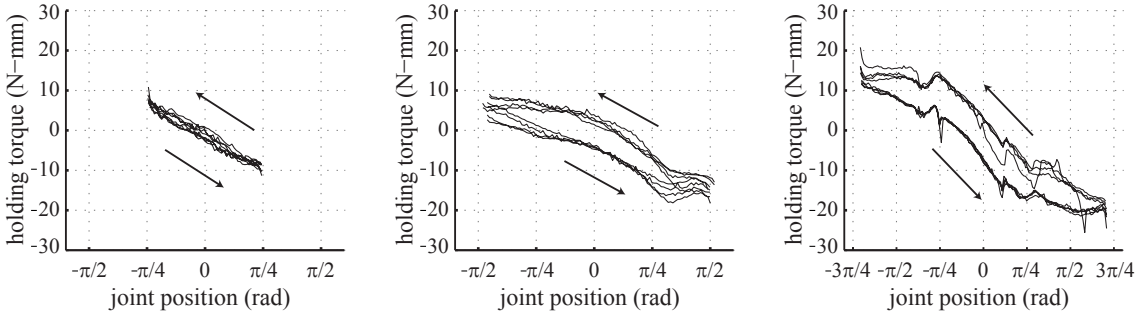
(a) Experimental setup (b) Tested hinge joints

Figure 4-12: Experimental setup used to measure holding torque of hinge joints



(a) $R = \frac{\pi}{2}$ (b) $R = \pi$ (c) $R = \frac{3\pi}{2}$

Figure 4-13: Holding torque vs. joint position for hinge joints folded from 0.051 mm thick polyester film, with joint ranges $R = \frac{\pi}{2}$, $R = \pi$, and $R = \frac{3\pi}{2}$



(a) $R = \frac{\pi}{2}$ (b) $R = \pi$ (c) $R = \frac{3\pi}{2}$

Figure 4-14: Holding torque vs. joint position for hinge joints folded from 0.127 mm thick polyester film, with joint ranges $R = \frac{\pi}{2}$, $R = \pi$, and $R = \frac{3\pi}{2}$

to the negative joint limit, for a total of 4 repetitions. For each step of the motor, the torque measurements obtained at that step were averaged.

Figures 4-13 and 4-14 show the results. Measurements were consistent across runs. For small angular deflections, folds can be approximated by a linear spring model. However, for larger angular deflections, folds start to exhibit nonlinear behavior. Holding torque levels off as the joint angle increases, similarly to the findings in [36], and even decreases for the thinner 0.051 mm thick material. For the thicker material (0.127 mm thick) the folds show hysteresis, with the amount of hysteresis increasing with the joint range. In the context of our model, this means that the equilibrium position of the joint changes depending on the motion of the joint.

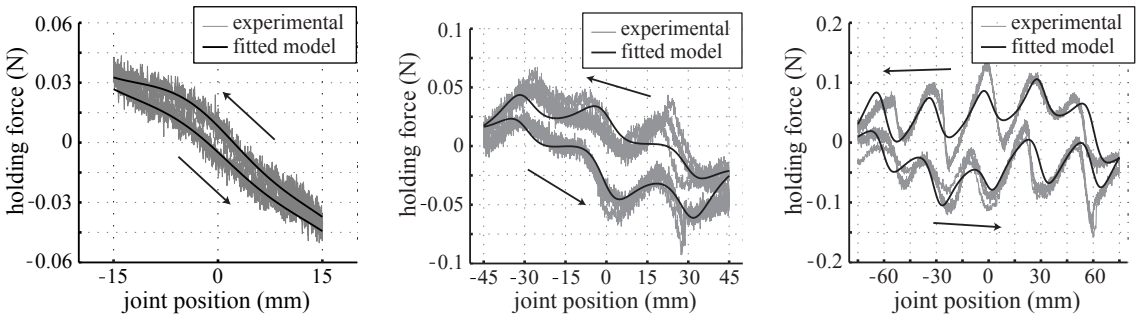
Prismatic Joint

To test our model of the prismatic joint, we generated and folded prismatic joints with $N_c \in \{1, 2, 4\}$, $N_\ell \in \{2, 4, 6, 8\}$, for a total of 12 joints. Each joint had $h = d = 15$ mm with $N_c w = 60$ mm, and we tested a joint range of $2(N_\ell - 1)\times 15$ mm. We used an Instron 5944 machine to measure the holding force vs. joint displacement for each of the joints. The height of the joint was maintained at 15 mm using the setup shown in Figure 4-15. We moved the joint to the negative joint limit at a rate of 10 mm/min., then to the positive joint limit, for a total of 3 repetitions.

Figures 4-16 and 4-17 show the resulting curves for three of the joints folded from 0.051 mm and 0.127 mm thick polyester film respectively. Active folds in a prismatic joint have fold angles that range from 0 to π radians, similar to the hinge joint with a range of motion of π radians, so it is expected that the curves exhibit hysteresis. The figures also show the model proposed in Section 4.2 fitted to the data using a least squares fit over the stiffness and equilibrium positions for each row. For simplicity, we assumed that the stiffness values k are constant and the same for each row of units. The shape of the experimental values matches well with the fitted

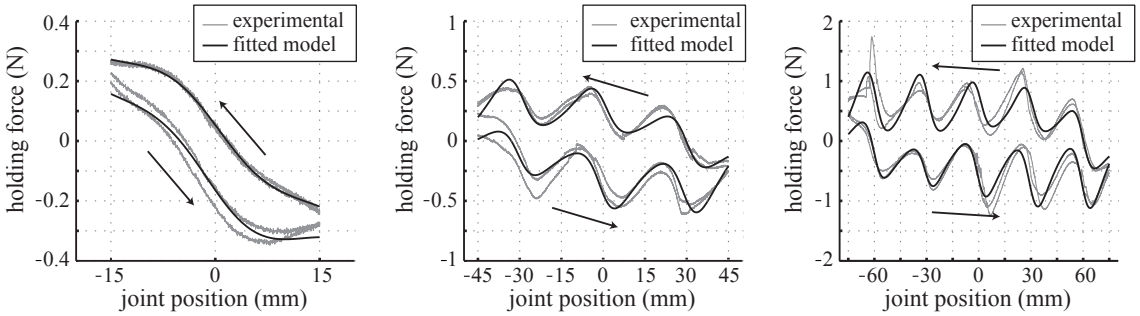
(a) Experimental setup (b) Close-up of joint

Figure 4-15: Experimental setup used to measure holding force of prismatic joints



(a) $N_c = 1$, $N_\ell = 2$ (b) $N_c = 2$, $N_\ell = 4$ (c) $N_c = 4$, $N_\ell = 6$

Figure 4-16: Holding force vs. joint position for prismatic joints folded from 0.051 mm thick polyester film, overlaid with curves fitted from the model



(a) $N_c = 1$, $N_\ell = 2$ (b) $N_c = 2$, $N_\ell = 4$ (c) $N_c = 4$, $N_\ell = 6$

Figure 4-17: Holding force vs. joint position for prismatic joints folded from 0.127 mm thick polyester film, overlaid with curves fitted from the model

curves. In general, the stiffness value was the same between forward and backward motion, while the equilibrium positions changed, which is consistent with our findings for the hinge joint. We suspect most of the approximation error can be attributed to nonlinearities in the spring torque. Large spikes in force such as those near $x = 30$ mm in Figure 4-16(b) and near $x = -60$ mm in Figure 4-17(c) occurred when the tabs that we added to facilitate folding snagged on other parts of the joint.

Table 4.1 shows the average stiffness values k for each of the fitted curves. Stiffnesses for the thinner material are lower than for the thicker material by a factor of 13.1 in the mean. This is close to the theoretical factor of $\left(\frac{0.127 \text{ mm}}{0.051 \text{ mm}}\right)^3 = 15.4$ derived from Equation (4.7). Finally, the stiffness of a row increases with N_c. This is unsurprising since as N_c increases, the number of active folds that contribute to the overall force also increases.

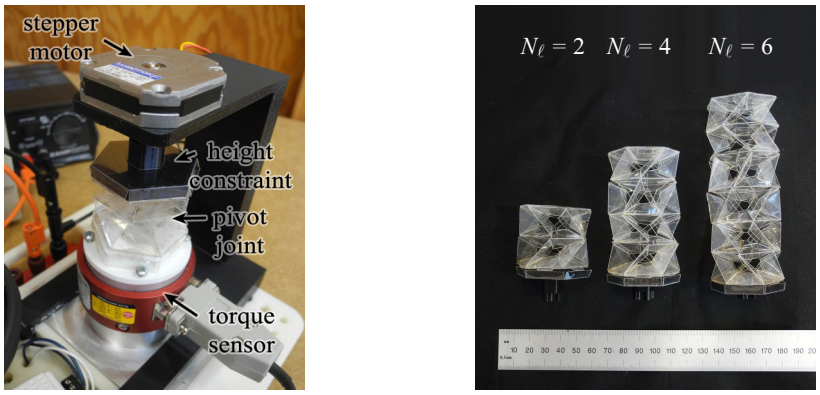
Table 4.1: Fitted fold stiffness values k for prismatic joints made from two thickness of polyester film

	0.051 mm thick			**0.127 mm thick**		
	$N_c = 1$	$N_c = 2$	$N_c = 4$	$N_c = 1$	$N_c = 2$	$N_c = 4$
$N_\ell = 2$	0.008	0.011	0.020	0.13	0.15	0.21
$N_\ell = 4$	0.013	0.018	0.022	0.20	0.34	0.38
$N_\ell = 6$	0.023	0.038	0.050	0.24	0.36	0.71
$N_\ell = 8$	0.017	0.044	0.041	0.22	0.33	0.46

Pivot Joint

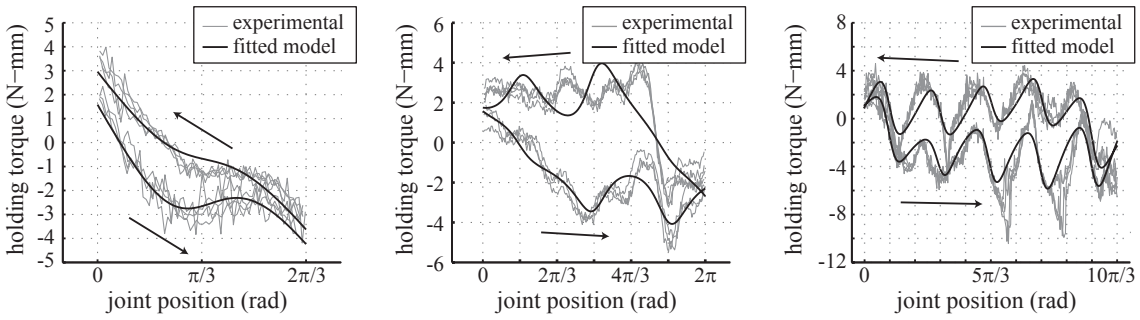
We generated and folded three 6-sided pivot joints ($r_o = 44$ mm) with 2, 4, and 6 layers, and we measured the holding torque for the joints using a similar setup to the one used for the hinge joint, with slight modifications as shown in Figure 4-18. The height of the joint was maintained at $\ell_s = 2r_o \sin\frac{\pi}{6} = 44$ mm throughout the tests. The joint was rotated clockwise at 1 step (200 steps/rev.) every 50 ms to the positive joint limit, then counterclockwise to the negative joint limit, for a total of 4 repetitions.

Plots of the results for the joints tested are shown in Figures 4-19 and 4-20. Again, we fitted our model to the resulting data using a least squares fit over the stiffness and the equilibrium positions for each layer. We assumed the stiffness was the same for all layers and constant over the entire run. The model accurately captures the behavior of a pivot joint with 2 layers. As for $N_\ell = 4$ and $N_\ell = 6$, the model is able to capture



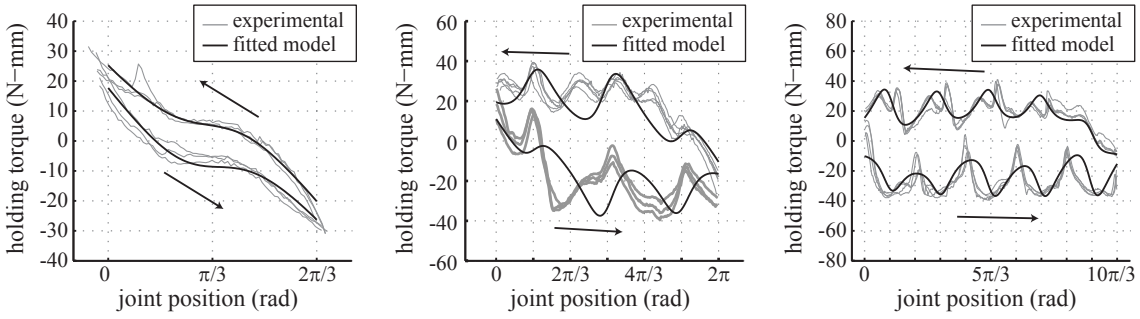
(a) Experimental setup (b) Tested pivot joints

Figure 4-18: Experimental setup used to measure holding torque of pivot joints



(a) $N_\ell = 2$ (b) $N_\ell = 4$ (c) $N_\ell = 6$

Figure 4-19: Holding torque vs. joint position for pivot joints folded from 0.051 mm thick polyester film, overlaid with curves fitted from the model



(a) $N_\ell = 2$ (b) $N_\ell = 4$ (c) $N_\ell = 6$

Figure 4-20: Holding torque vs. joint position for pivot joints folded from 0.127 mm thick polyester film, overlaid with curves fitted from the model

the general structure of the model, such as its periodicity. We believe the local errors to be due to nonlinearities in the spring torque and to deformation of the faces during motion, which was not included in the model. This is supported by how the pivot joint appears to exhibit different behavior depending on the direction of rotation. As discussed in Section 4.1.4, the pivot joint cannot move rigidly, and so we have added cuts to the fold pattern, which more easily accommodate stretch when rotating clockwise than counterclockwise (ref. Figure 4-10(b)). For small deviations, the pseudo-rigid-body model we employed should be able to account for some elongation of faces [76]. Our results indicate that a deeper understanding of the joint requires more detailed models.

4.4 Summary

We have presented fold patterns for multiple joints common in robot designs. Our fold patterns are parameterized to deliver the range of motion needed by the robot application, and they are designed to be composed with other modules to achieve the full range of joint types. The joints are controllable using electronic components that can be integrated in the printing process. We have also performed an analysis of our joints to determine their behavior under driving forces and torques, and we have shown here that simple spring-based models may not be sufficient since a fold's behavior is dependent on the movements it has previously experienced. Future work includes better characterization of the mechanical properties of folded structures and the dynamics of folded joints, as well as investigations of other common mechanisms used in robotic designs.

Chapter 5

Composition of Foldable Mechanisms

A necessary part of a design-by-composition approach is ensuring that compositions are valid. In this chapter, we address composition of foldable modules, and we present an algorithm for automatically composing multiple foldable designs together so that the end product has the combined geometry of the originals. Figure 5-1 illustrates the problem addressed. Given two fold patterns, in this case a walking robot and a gripper, our algorithm automatically generates a composite fold pattern for a walking robot with a gripper on one end.

In particular, we address the problem of edge-compositions, compositions involving two surfaces connected at one edge via a fold. For ease of assembly of the final product, we require that the fold pattern be one piece. In addition, for structural reasons, we would like for the composite fold pattern to contain the original fold patterns, rather than for it to be a completely new one. This is because in a folded state, cut edges, edges corresponding to edges on the boundary of a fold pattern that have been glued together, are mechanically weaker than folds. Cutting along an edge that will be subjected to large stresses may drastically weaken the final product. We assume that the two inputted fold patterns perform their intended functions and therefore require that the composite folding pattern contain the original fold patterns in their entirety as subsets.[1]

Our main result is the following:

[1]The majority of this chapter was published in [168, 169, 172].

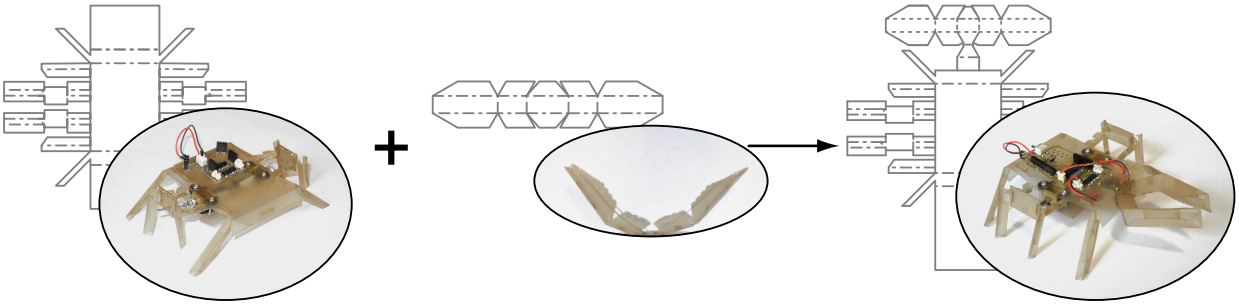


Figure 5-1: *Left*: Walking and gripper robots folded out of the patterns shown. *Right*: The *composition*, a walking-gripping robot, whose fold pattern was designed manually. Our goal is to generate such a fold pattern automatically. *Credit: Robots were designed by Cagdas Onal and Michael Tolley.*

Theorem 5.2.3. *Any edge-composition of two folded surfaces has a one-piece non-self-intersecting fold pattern consisting of 1) the fold patterns of the two original folded surfaces connected by 2) a bridge of linking material.*

This result provides more efficient material usage than previous work in origami design [31, 117, 173, 174], which requires that all designs be folded from a convex polygon, and more easily accommodates non-static shapes. By adding linking material, it also improves upon previous work in polyhedron unfolding [15, 39] since it is guaranteed to compose any two valid input designs. We provide a polynomial-time algorithm for generating the composite fold pattern, and we demonstrate the algorithm on various input surfaces to create foldable linkage mechanisms.

5.1 Problem Statement

Consider again a fold pattern $(P, \mathcal{F})$ consisting of a non-self-intersecting polygon P and a set of folds $\mathcal{F}$ (ref. Section 4.1.1). We denote the folds $f_i = (e_i^P, \phi_i)$, where e_i^P is the fold line and ϕ_i is the fold angle. The fold line and the line segments forming the boundary of P together are the edges $\mathcal{E}(P, \mathcal{F}) = \{e_1^P, e_2^P, \ldots\}$ of the fold pattern, and their endpoints are the *vertices* $\mathcal{V}(P, \mathcal{F}) = \{v_1^P, v_2^P, \ldots\}$. We call the space that is not part of P, $\mathbb{R}^2 \setminus P$, the *free space*.

Now let Q denote the polyhedral complex that is the image of the folded state of the fold pattern $(P, \mathcal{F})$. We say that the pattern $(P, \mathcal{F})$ folds into Q and that Q can be unfolded into $(P, \mathcal{F})$. Let $\mathcal{E}(Q)$ denote the edges of Q. Then the problem we would like to solve is as follows.

Problem 5.1.1. *Given two polyhedral complexes Q_1 and Q_2 with corresponding fold patterns $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$, and two edges $e^{Q_1} \in \mathcal{E}(Q_1)$ and $e^{Q_2} \in \mathcal{E}(Q_2)$, find a fold pattern $(P_3, \mathcal{F}_3)$ such that*

1. *$(P_3, \mathcal{F}_3)$ folds into the union of Q_1 and Q_2 translated and rotated so that e^{Q_1} is coincident to e^{Q_2}, and*

2. *$(P_3, \mathcal{F}_3)$ contains translated, rotated, and/or reflected instances of $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$ as subsets.*

The selected edges e^{Q_1} and e^{Q_2} act as a hinge in the combined surface. In order for a solution to Problem 5.1.1 to make sense, the range of fold angle of this hinge must not cause Q_1 and Q_2 to collide. We assume that this range is nonempty.

The edges e^{Q_1} and e^{Q_2} can be mapped to edges on the fold patterns $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$. Because the pattern contains cuts and faces can be multiply covered, it is possible for multiple edges in $(P_1, \mathcal{F}_1)$ to correspond to e^{Q_1}. Let $\mathcal{E}^{P_1}$ be the set of these edges, and similarly for $\mathcal{E}^{P_2}$. By definition, if we are able to guarantee for a $(P_3, \mathcal{F}_3)$ that one edge in $\mathcal{E}^{P_1}$ will coincide with one edge in $\mathcal{E}^{P_2}$ in the folded state, then $(P_3, \mathcal{F}_3)$ will satisfy condition (1) of Problem 5.1.1. We therefore modify the problem statement slightly to deal with the fold patterns.

Problem 5.1.2. *Given two fold patterns $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$, an edge e^{P_1} in $(P_1, \mathcal{F}_1)$, and an edge e^{P_2} in $(P_2, \mathcal{F}_2)$, find a fold pattern $(P_3, \mathcal{F}_3)$ such that*

1. *$(P_3, \mathcal{F}_3)$ folds into the union of translated and rotated instances of Q_1 and Q_2, the images of folded states of $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$ respectively,*

2. *in the folded state, e^{P_1} and e^{P_2} coincide, and*

3. $(P_3, \mathcal{F}_3)$ *contains rotated, translated, and/or reflected instances of* $(P_1, \mathcal{F}_1)$ *and* $(P_2, \mathcal{F}_2)$ *as subsets.*

Solving this problem for any combination of $e^{P_1} \in \mathcal{E}^{P_1}$ and $e^{P_2} \in \mathcal{E}^{P_2}$ will satisfy Problem 5.1.1. The remainder of this paper is concerned with solving Problem 5.1.2.

5.1.1 Representation

For analysis of our algorithms, we assume a random-access machine that can perform real arithmetic and store real numbers. We store a fold pattern $(P, \mathcal{F})$ on this machine in four parts:

1. an $N \times 2$ array $\mathbf{V}_{(P,\mathcal{F})}$, where N is the total number of unique vertices in P and $\mathcal{F}$. Row i of $\mathbf{V}_{(P,\mathcal{F})}$ contains the (x, y) coordinate values of the ith vertex.

2. an $M \times 2$ array $\mathbf{E}_{(P,\mathcal{F})}$, where M is the total number of edges in P and $\mathcal{F}$. Row i of $\mathbf{E}_{(P,\mathcal{F})}$ contains the indices from $\mathbf{V}_{(P,\mathcal{F})}$ of the endpoints of edge i.

3. a list $\mathbf{B}_{(P,\mathcal{F})}$ of length M. Element i is an indicator equal to 1 if edge i is an edge on P and 0 if edge i is a fold line in $\mathcal{F}$.

4. a list $\mathbf{\Theta}_{(P,\mathcal{F})}$ of length M. Element i is the fold angle corresponding to edge i if edge i is a fold line and 0 otherwise.

Thus storage of $(P, \mathcal{F})$ is $\mathcal{O}(N + M)$. For the remainder of this paper, we will use the notation that N_i is the number of vertices in $(P_i, \mathcal{F}_i)$ and M_i the number of edges. Note that for a fold pattern, $M_i \sim \mathcal{O}(N_i)$ by Euler's formula for planar graphs.

5.2 Composition Algorithm

5.2.1 Main Insight: Edges on the Convex Hull Boundary

In order to construct $(P_3, \mathcal{F}_3)$, the input fold patterns $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$ must be arranged in the plane without intersection. Extra material can be added so that the

fold patterns connect as long as the edges to be joined, e^{P_1} and e^{P_2}, are coincident in the folded state. We use the following insight.

Lemma 5.2.1. *If e^{P_1} and e^{P_2} are on the boundaries of the convex hulls of their respective fold patterns, then they may be placed coincident in the plane and will not cause $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$ to intersect.*

In this case, the fold pattern $(P_3, \mathcal{F}_3)$ is simply the union of $(P_1, \mathcal{F}_1)$ and the transformed $(P_2, \mathcal{F}_2)$, with the edge e^{P_1} (also e^{P_2}) converted from a boundary edge of P_1 (respectively P_2) to a fold in $\mathcal{F}_3$.

When e^{P_1} or e^{P_2} is not on the boundary of the convex hull of its fold pattern, then naïvely following the above procedure may lead to self-intersection of $(P_3, \mathcal{F}_3)$. However, it is possible to modify $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$ without changing their corresponding folded structures so that the above procedure may be used. Taking the case of $(P_1, \mathcal{F}_1)$, this modification would consist of constructing an additional fold pattern $(P^b, \mathcal{F}^b)$ and attaching it to $(P_1, \mathcal{F}_1)$ so that in the folded state e^{P_1} becomes coincident to an edge on the boundary of the combined fold pattern's convex hull. We call $(P^b, \mathcal{F}^b)$ a *bridge* and say that e^{P_1} has been *bridged* to the boundary of the convex hull.

As we will show in Section 5.2.2,

Lemma 5.2.2. *Given a fold pattern $(P, \mathcal{F})$ and an edge e^P, it is always possible to bridge e^P to the boundary of the convex hull. The bridge can be constructed in $\mathcal{O}(N^2 \log(N))$ time.*

Combining Lemmas 5.2.1 and 5.2.2 yields our algorithm (Algorithm 1) and main result.

Theorem 5.2.3. *For any fold patterns $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$, and edges e^{P_1} in $(P_1, \mathcal{F}_1)$ and e^{P_2} in $(P_2, \mathcal{F}_2)$, there exists a fold pattern $(P_3, \mathcal{F}_3)$ that satisfies Problem 5.1.2. Furthermore, this fold pattern can be computed using Algorithm 1 in time polynomial in the number of vertices and edges in the original fold patterns.*

Algorithm 1: COMPOSE$((P_1, \mathcal{F}_1), e_1^P, (P_2, \mathcal{F}_2), e_2^P)$

Data: $(P_1, \mathcal{F}_1)$ = first input fold pattern
e^{P_1} = edge on $(P_1, \mathcal{F}_1)$ to connect to $(P_2, \mathcal{F}_2)$
$(P_2, \mathcal{F}_2)$ = second input fold pattern
e^{P_2} = edge on $(P_2, \mathcal{F}_2)$ to connect to $(P_1, \mathcal{F}_1)$
Result: $(P_3, \mathcal{F}_3)$ = composite fold pattern satisfying Problem 5.1.2

1 $\left\{(P_1', \mathcal{F}_2'), e^{P_1'}\right\} \leftarrow$ BRIDGE$\left((P_1, \mathcal{F}_1), e^{P_1}\right)$; `// Algorithm 2 or 4`
2 $\left\{(P_2', \mathcal{F}_2'), e^{P_2'}\right\} \leftarrow$ BRIDGE$\left((P_2, \mathcal{F}_2), e^{P_2}\right)$; `// Algorithm 2 or 4`
3 Rotate, translate, and reflect $(P_2', \mathcal{F}_2')$ so that $e^{P_2'}$ is coincident to $e^{P_1'}$ and $(P_1', \mathcal{F}_1')$ and $(P_2', \mathcal{F}_2')$ do not intersect;
4 $P_3 \leftarrow P_1' \cup P_2'$; $F_3 \leftarrow \mathcal{F}_1' \cup \mathcal{F}_2' \cup \{(e^{P_1'}, \phi)\}$;

5.2.2 Constructing the Bridge

This section describes the algorithms and analysis that establish Lemma 5.2.2. There are two cases to consider:

1. e^P is an exterior edge (on the outer boundary of P)

2. e^P is an interior edge (on the inner boundary of P or a fold line in $\mathcal{F}$)

We show that in either case, it is possible to construct a bridge such that in the folded state, an edge on the boundary of the convex hull of the modified fold pattern collapses onto e^P.

Case 1: Exterior Edges

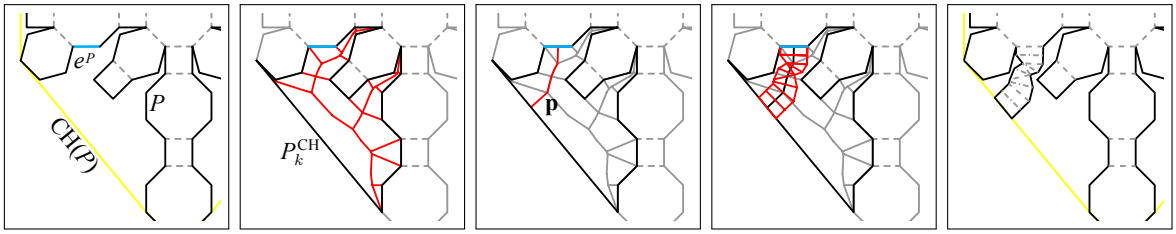
The bridge constructing algorithm for exterior edges is given in Algorithm 2 and illustrated in Figure 5-2. Starting with a fold pattern $(P, \mathcal{F})$, the procedure consists of finding a path through the free space and creating pleats along that path to fold onto the edge e^P.

Line 1: Find a Path to the Convex Hull Boundary In the first step, we search for a path through the free space from e^P to the boundary of the convex hull. Theoretically, any such path will suffice. We restrict the path to be a simple polygonal chain. In so doing, it becomes possible to represent a path $\mathbf{p}$ of length n as a finite

Algorithm 2: BRIDGEFROMBOUNDARY$((P, \mathcal{F}), e^P)$

Data: $(P, \mathcal{F})$ = input fold pattern
e^P = boundary edge to bridge
Result: $(P_{new}, \mathcal{F}_{new})$ = fold pattern containing $(P, \mathcal{F})$
e^P_{new} = edge on boundary of convex hull of P_{new} that folds onto e^P

1 $\mathbf{p} \leftarrow$ path from e^P to CH(P); `// Lemma 5.3.4`
2 $(P^b, \mathcal{F}^b) \leftarrow$CREATEPLEATS$(e^P, \mathbf{p})$; `// Algorithm 3`
3 $P_{new} \leftarrow P \cup P^b$; $\mathcal{F}_{new} \leftarrow \mathcal{F} \cup \mathcal{F}^b \cup \{(e^P, \pi)\}$;
4 Remove intersections;
5 $e^P_{new} \leftarrow e_{n_e}$ created during bridge construction in line 2;



(a) Input (b) Medial axis of P_e^{CH} (c) Line 1 (d) Line 2 (e) Output

Figure 5-2: Algorithm 2: Bridging an edge on the boundary of the fold pattern to the boundary of the convex hull. (a) Original fold pattern. The edge to join e^P is highlighted in blue and the convex hull is outlined in yellow. (b) The region P_k^{CH} (outlined in black) and its straight skeleton (red). (c) The path $\mathbf{p}$ from e^P to the boundary of the convex hull. (d) Pleats tiled along the path. (e) Output fold pattern with the bridge added.

list of the vertices $\mathbf{p} = v_1 v_2 \dots v_n$ in the order they appear in the chain. Since the goal is to overlay the path with pleats, we choose a path surrounded on both sides by as much free space as possible. A path along the straight skeleton of the free space satisfies this criterion.

Take the convex hull of P and let P_e^{CH} be the region of space in CH $\setminus P$ that is bounded by e^P. The straight skeleton of P_e^{CH} is the union of trajectories of vertices of P_e^{CH} when the boundary of P_e^{CH} is shrunk in such a way that all edges retain their orientation and move towards the interior of P_e^{CH} at the same speed. The straight skeleton was introduced in [3] as an alternative to the medial axis, which can contain parabolic arcs when P_e^{CH} is nonconvex. As its name suggests, the straight skeleton contains only line segments.

The straight skeleton separates P_e^{CH} into regions, one containing each edge on the boundary of P_e^{CH}. The path that we use to construct the pleats is the path in the straight skeleton from the region containing e^P to the region containing the convex hull edge (Figure 5-2(c)). Since the straight skeleton is a tree, there will only be one such path. Let $\mathbf{p} = v_1 v_2 \dots v_{n_e}$ be this path, with the first vertex v_1 located at the midpoint of e^P, intermediate vertices $v_2, \dots, v_{n_e-1}$ at vertices in the straight skeleton, and the final vertex v_{n_e} on the convex hull edge.

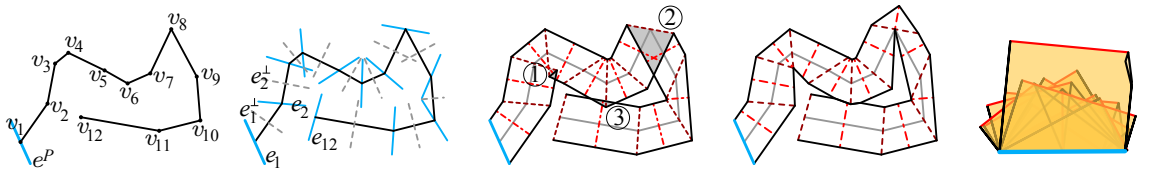
Line 2–3: Overlay Pleats Accordion style pleats are overlaid on the path $\mathbf{p}$ using Algorithm 3 (Figure 5-3). These pleats are a fold pattern consisting of a sequence of non-intersecting folds with fold angles alternating between π and $-\pi$ so that the edge at the end of the path (the convex hull edge) collapses exactly onto e^P in the folded state. For the last pleat, rather than following the procedure in lines 3–5 of Algorithm 3, the point of intersection between the line common to edge e_{n_e-1} and the line common to the convex hull edge is found. Then, e_{n_e-1} is rotated about this point of intersection onto the convex hull edge to create e_{n_e}. This guarantees that the last edge added to the pleated structure lies on the boundary of the fold pattern's convex

---

Algorithm 3: CREATEPLEATS($e^P, \mathbf{p}$)

Data: e^P = starting edge
$\mathbf{p} = v_1 v_2 \dots v_{n_e}$ = path to follow
Result: $(P^b, \mathcal{F}^b)$ = pleats fold pattern satisfying Lemma 5.3.1

`// Beginning with` e^P`,`
1 $e_1 = e^P$;
`// Compute fold line locations`
2 **for** $i = 2, \dots, n_e$ **do**
3 $\quad \ell_i^\perp \leftarrow$ perpendicular bisector of segment $v_{i-1}v_i$;
4 $\quad e_i \leftarrow$ reflection of e_{i-1} over $\ell_i^\perp$;
5 $\quad e_i^\perp \leftarrow$projection of e_{i-1} onto $\ell_i^\perp$;
6 **end**
7 Remove intersections;
`// Fold pattern of pleated structure`
8 $P^b \leftarrow$ polygon strip containing the edges $e_1, e_2^\perp, e_2 \dots e_{n_e}^\perp, e_{n_e}$ in order;
9 $\mathcal{F}^b \leftarrow \left\{(e_{n_e}^\perp, -\pi)\right\} \cup \bigcup_{i=2,\dots,n_e-1} \left\{(e_i, \pi), (e_i^\perp, -\pi)\right\}$;



(a) Input (b) Lines 2–6 (c) Self-intersections (d) Line 7 (e) Folded State

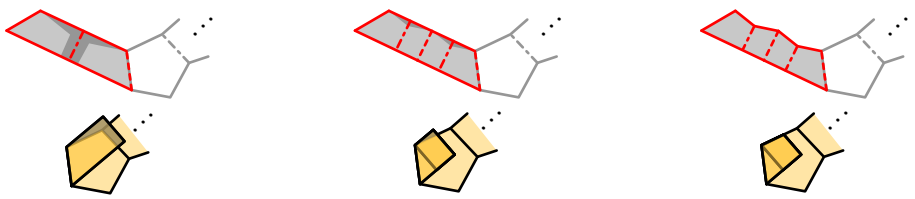
Figure 5-3: Algorithm 3: Constructing pleats to follow a path. (a) The edge e^P and the path $\mathbf{p}$ to follow. (b) Reflected edges e_i (solid) and perpendicular bisectors $e_i^\perp$ (dotted). (c) Resulting pleats. Types 1, 2, and 3 self-intersections are shaded. (d) Pleats with self-intersections corrected. (e) Folded state of pleats. All e_i coincide.

hull and that the new pleat is still an isosceles trapezoid. As in Algorithm 3, the edge e_{n_e} is then assigned a fold angle of π, and a fold line is added on the new pleat's axis of symmetry with a fold angle of $-\pi$. The last edge's exact location on the convex hull boundary is not prespecified. Since, however, the point v_{n_e-1} is a vertex on the straight skeleton bordering the region containing the convex hull edge, the median of this pleat will lie entirely inside the free space. The result of this step is a bridge that collapses flat onto the face adjacent to e^P.

Line 4: Remove Intersections with the Input Fold Pattern Although the path $\mathbf{p}$ lies entirely in the free space, the pleats following $\mathbf{p}$ have a width and may intersect with the input fold pattern (Figure 5-4). In this case, the overlapping regions can be cut out of the bridge. When this operation causes the bridge to become disconnected, then pieces that are not connected to e^P should also be removed. The bridge will still extend from e^P to the convex hull boundary since p lies entirely in the free space.

(a) Before (b) After

Figure 5-4: Intersection removal. (a) A bridge (red) that intersects with $(P, \mathcal{F})$ (gray). (b) The offending region is removed.



(a) Before (b) Crimped (c) Trimmed

Figure 5-5: A long pleat. *Top*: The fold pattern with offending pleat in red. *Bottom*: Folded state. (a) In the folded state, the pleat protrudes outside the adjacent face. (b) It can be crimped to not interfere with other folds and (c) trimmed to avoid protrusions.

Finally, if a pleat is so long that it interferes with the folding of P, it can be trimmed or fold lines can be added to crimp the pleat arbitrarily small (Figure 5-5).

Case 2: Interior Edges

When e^P is not on the outer boundary of P, then Algorithm 2 alone cannot be used. Instead, we must first construct a new boundary edge e^P_{ref} that in the folded state will coincide with e^P. This can be achieved by taking a path through the interior of P to an edge e_b on the boundary and reflecting the path out of P. Algorithm 4, illustrated in Figure 5-6, gives the procedure for constructing a bridge for this case.

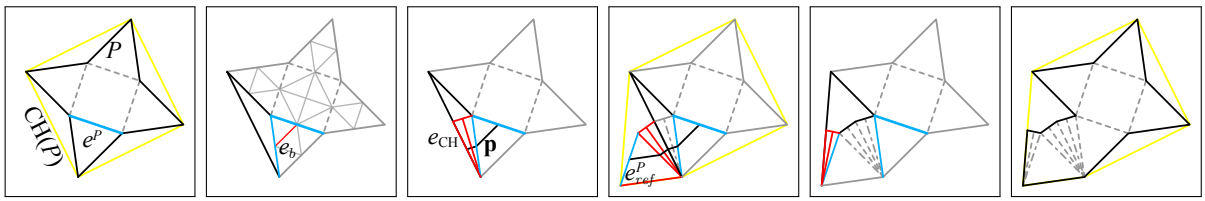
Line 1: Choose A Boundary Edge Since e^P is not on the outer boundary of P, connecting a bridge at e^P may cause self-intersection of the final fold pattern. Therefore, before constructing the bridge, it is necessary to choose where to attach it to the fold pattern. Any edge e_b on the outer boundary can be used here. For our implementation, we chose the e_b that minimizes the length of the final constructed bridge. We estimated the length of the bridge by using the fold pattern's edge-adjacency graph, which is the dual of the fold pattern where the vertices are located at the midpoints of the edges in P and F, and edges connect vertices that lie on the boundaries of the same face in $(P, \mathcal{F})$. The edge-adjancency graph has the following properties:

- Every edge corresponds to a single face in $(P, \mathcal{F})$.

Algorithm 4: BRIDGEFROMFOLDLINE$((P, \mathcal{F}), e^P)$

Data: $(P, \mathcal{F})$ = input fold pattern
e^P = fold line edge to bridge
Result: $(P_{new}, \mathcal{F}_{new})$ = fold pattern containing $(P, \mathcal{F})$
e^P_{new} = edge on boundary of convex hull of P_{new} that folds onto e^P

1 $e_b \leftarrow$ a boundary edge on P;
2 $\{(P_2, \mathcal{F}_2), e_{\text{CH}}\} \leftarrow$ BRIDGEFROMBOUNDARY$((P, \mathcal{F}), e_b)$;
3 $\{(P_3, \mathcal{F}_3), e^P_{ref}\} \leftarrow$REFLECTFACES$((P_2, \mathcal{F}_2), e^P, e_{\text{CH}})$;
4 $\{(P_{new}, \mathcal{F}_{new}), e^P_{new}\} \leftarrow$ BRIDGEFROMBOUNDARY$((P_3, \mathcal{F}_3), e^P_{ref})$;



(a) Input (b) Line 1 (c) Line 2 (d) Line 3 (e) Line 4 (f) Output

Figure 5-6: Algorithm 4: Bridging an edge on the interior of the fold pattern to the boundary of the convex hull. (a) Original fold pattern. The edge to join e^P is highlighted in blue, and the convex hull is outlined in yellow. (b) The edge-adjacency graph with the path from e^P to e_b highlighted in red. (c) Pleats attached to e_b using Algorithm 2. (d) The accordion path and interior faces reflected over the boundary of the convex hull. The convex hull is also updated. (e) Pleats attached to e^P_{ref} using Algorithm 2. (f) Output fold pattern with the bridge added.

- A path in the graph corresponds to a connected set of faces in $(P, \mathcal{F})$.
- The graph has M vertices and $\mathcal{O}(M^2)$ edges.
- The graph can be constructed in $\mathcal{O}(M^2)$ time [38].

The length of a bridge can then be approximated as the sum of the lengths of 1) the path from e^P to e_b in the edge-adjacency graph and 2) the path from e_b to the boundary of the convex hull in the straight skeleton produced when calling Algorithm 2. Note that to calculate this cost, we generate the straight skeleton of every region of free space P_e^{CH} in the fold pattern's convex hull. There are $\mathcal{O}(N)$ such regions.

Line 2: Bridge e_b to the Convex Hull Boundary Unless e_b is on the boundary of P's convex hull, the faces between e^P and e_b cannot simply be reflected over e_b

(a) Fold Pattern (b) Dual: Edge-Adjacency Graph

Figure 5-7: Example edge-adjacency graph

to make e^P a boundary edge, since this operation may result in intersection with P. Instead, the edge e_b must first be bridged to the boundary of the convex hull using Algorithm 2. Let e_{CH} be the resulting edge on the convex hull boundary.

Line 3: Reflect Faces between e^P and e_{CH} In order to construct an e^P_{ref} on the boundary of the fold pattern, reflect all faces between e^P and e_{CH} over e_{CH}. These faces can be found by consulting the edge-adjacency graph. A path in the graph from the vertex corresponding to e^P to the vertex corresponding to e_{CH} yields a set of faces that connect e^P and e_{CH}. Since e_{CH} is on the convex hull boundary, the reflected faces will not intersect with the rest of the fold pattern.

Note that when e^P is a boundary edge but not an exterior edge, then it must be the boundary of a hole. Reflecting the entire face that contains e^P may not help the situation if e^P will remain on the boundary of a hole. Therefore, in this case, only part of the face containing e^P is reflected. This part is computed by taking the shortest path within the face from the edge e^P to the edge connecting the face to the bridge being built and then maximizing the width of this path within the face.

Line 4: Bridge e^P_{ref} to the Convex Hull Boundary The result of line 3 is that e^P_{ref} is now on the boundary of P but not necessarily of the convex hull, reducing the situation to Case 1. In the folded state, the reflected path will fold flat along the surface of the input folded structure Q so that the reflected edge is coincident e^P. Thus any material attached to e^P_{ref} is as if it were added at e^P.

5.2.3 Optimizations

It is not always necessary to perform the full bridging procedure to compose two fold patterns. For simple fold patterns, it is often the case that two edges can be connected together to create a valid fold pattern without any extra material. Even if this is not the case, every other edge of the pleated bridge folds onto the edge e^P and can be used as the connecting edge. Using any one of these edges that allows a valid composite fold pattern to be produced will reduce the amount of waste material used.

5.2.4 Face-Composition

This chapter focuses on composing fold patterns along an edge, which is the minimum attachment necessary for two surfaces to be joined. This leads to a folded state where the two input surfaces can move relative to each other. Another type of joining is a face-composition, where the two input surfaces would be attached along one or multiple faces and would be fixed relative to each other. An extension of the proposed algorithm to face-compositions is straightforward: remove the faces to join from the fold patterns if desired, possibly breaking the fold pattern into multiple parts, and repeat the algorithm for every pair of edges along the boundaries of those faces, ensuring that the fold angles at each of the edges joining the two fold patterns are consistent. If the resulting pattern is multiple pieces, information about the gluings (i.e., how cuts on the boundary of the fold pattern are joined in the folded state) can be used to further merge the fold patterns into a single piece.

5.3 Correctness and Material Usage Guarantees

To prove that these algorithms are correct, we must show that they will always produce a valid non-self-intersecting fold pattern. We start with the main insight.

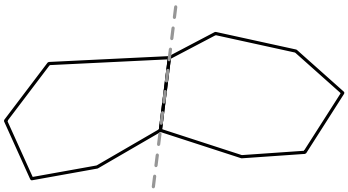


Figure 5-8: Two convex polygons placed next to each other are guaranteed not to intersect

Lemma 5.2.1. *If e^{P_1} and e^{P_2} are edges on the boundaries of the convex hulls of fold patterns $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$, respectively, then the fold patterns may be rigidly transformed such that the edges are coincident in the plane and they will not intersect.*

Proof: Let $\mathrm{CH}(P_1)$ be the convex hull of P_1. By definition, $P_1 \subseteq \mathrm{CH}(P_1)$. Because $\mathrm{CH}(P_1)$ is convex and e^{P_1} is an edge on its boundary, $\mathrm{CH}(P_1)$ must lie entirely on one side of the line common to e^{P_1} (see Figure 5-8). Similarly, $\mathrm{CH}(P_2)$ must lie entirely on one side of the line common to e^{P_2}.

Rotate, translate, and reflect $(P_2, \mathcal{F}_2)$ so that e^{P_2} is coincident to e^{P_1} and $\mathrm{CH}(P_2)$ is on the opposite side of e^{P_2} as $\mathrm{CH}(P_1)$. Since $\mathrm{CH}(P_1)$ and $\mathrm{CH}(P_2)$ are on opposite sides of the line now collinear to both e^{P_1} and e^{P_2}, they cannot intersect. Likewise, P_1 and P_2 cannot intersect. ■

5.3.1 Pleat Creation

In order to bridge e^P to the boundary of the convex hull, our algorithm computes a path and overlays it with accordion-style pleats. This is always possible by virtue of the following lemma.

Lemma 5.3.1. *Given a starting edge e^P and any simple path $\mathbf{p} = v_1 v_2 \ldots v_{n_e}$ such that the first vertex v_1 is on e^P, Algorithm 3 constructs a series of pleats such that every v_i lies on a fold f_i, and in the folded state, all f_i are coincident to e^P.*

Proof: The pleat structure produced by Algorithm 3 is based on an isosceles trapezoid. If an isosceles trapezoid is folded with a fold angle of $\pm\pi$ down its axis of symmetry, then its two legs will coincide in the folded state. In a chain of isosceles trapezoids, where every trapezoid shares only its legs with its neighbors, folding every trapezoid

down its axis symmetry will cause the legs of all the trapezoids to coincide. Therefore, a chain of folded isosceles trapezoids where every path vertex v_i lies on a leg of a trapezoid and e^P is also the leg of a trapezoid is a pleat structure that satisfies the conditions of this lemma.

Line 3 of the algorithm uses the perpendicular bisectors of the segments in $\mathbf{p}$ to ensure that the median of every trapezoidal pleat is exactly one segment of $\mathbf{p}$, and that each newly created edge has as its midpoint the next vertex of $\mathbf{p}$. The resulting pleats have a width of at most $\|e^P\|$.

Line 7 makes the pleats non-self-intersecting. During the pleat construction in lines 2-8, three types of self-intersection can occur (see Figure 5-3(c)):

1. When an edge to reflect e_i intersects with the perpendicular bisector, the resulting pleat must be trimmed to a triangle. Note that the triangle will still contain the path vertex.

2. When a pleat overlaps with the subsequent segment of $\mathbf{p}$, it results in an intersection between adjacent pleats. Let e_i be the edge shared between the two intersecting pleats, and v_i be its midpoint. The two pleats are both trimmed to non-isosceles trapezoids that meet at v_i. This operation alone would cut the pleats into two pieces. Therefore, a second set of right triangular pleats must be added in the free space next to e_i to maintain connectivity. The addition of the triangular pleats causes a change in orientation of the pleats. If such a change in undesirable, the fold down the middle of the triangles can also be omitted.

3. When nonadjacent segments of $\mathbf{p}$ are close together, their corresponding pleats may overlap. The overlap may be resolved by dividing the overlapping region along the bisectors of the two conflicting segments of $\mathbf{p}$. Since the path $\mathbf{p}$ is simple, the pleats will remain connected and the fold lines will still contain the path vertices.

In all cases, the modifications do not disconnect the pleats, so the pleats will be a valid fold pattern. In addition, the fold lines remain in the same locations, so the pleat structure will satisfy the conditions of the lemma. ■

It is also of interest to check the storage size required by the resulting pleat structure and the complexity of Algorithm 3.

Lemma 5.3.2. *Algorithm 3 outputs a fold pattern with $\mathcal{O}(n_e)$ vertices and $\mathcal{O}(n_e)$ edges.*

Proof: Each of the n_e iterations of the for loop in lines 2–6 adds a constant number of vertices and edges to the pleat structure. In line 7, type 1 intersections cause removal of two vertices and one edge for each intersection, of which there can be at most n_e. Type 2 intersections cause addition of two vertices and four edges each, and again there can be at most n_e. In type 3 intersections, the number of vertices added to the pleats is equal to twice the number of vertices involved in the intersection, plus four. We say that a vertex is involved in an intersection if it lies on the boundary of the intersecting region. It is clear that a vertex can only ever participate in at most one type 3 intersection. Therefore, the total number of vertices added to correct type 3 intersections is at most $\mathcal{O}(n_e)$. Since the edges only form chains to connect the new vertices, the number of edges added is also $\mathcal{O}(n_e)$. Thus the output fold pattern of Algorithm 3 has $\mathcal{O}(n_e)$ vertices and $\mathcal{O}(n_e)$ edges. ■

Lemma 5.3.3. *Algorithm 3 takes $\mathcal{O}\left((n_e + n_i)\log n_e\right)$ time, where n_i is the number of self-intersections detected in line 7.*

Proof: The for loop in lines 2–6 is executed $n_e - 1$ times and contains a body that is $\mathcal{O}(1)$, so it takes $\mathcal{O}(n_e)$ time total. In line 7, the pleats must be checked for each type of intersection. Type 1 and 2 intersections can only occur between neighboring pleats. They can be found and corrected simply by iterating through the pleats in order, an operation that takes $\mathcal{O}(n_e)$ time. Detecting type 3 intersections, on the other hand, requires a check for self-intersection of P^b. Naïvely checking every pair of edges on the boundary of P^b for intersection would take $\mathcal{O}(n_e^2)$. It is usually faster to use the Bentley-Ottman sweep line algorithm [14], which is $\mathcal{O}\left((n_e + n_i)\log n_e\right)$ time, where n_i is the number of intersections.

Summing over all the lines yields an overall complexity of $\mathcal{O}\left((n_e + n_i)\log n_e\right)$. Note that in the worst case, when n_i is $\mathcal{O}(n_e^2)$, the complexity can be reduced to $\mathcal{O}(n_e^2)$ if the naïve collision checking method is used. ■

5.3.2 Bridging Exterior Edges

Our bridging algorithm for exterior edges is based on the following lemma:

Lemma 5.3.4. *If e^P is on the outer boundary, then there exists a path through the free space beginning at a point on e^P and ending on the boundary of the convex hull of P.*

Proof: Because P is a polygon, its convex hull $\mathrm{CH}(P)$ is also a polygon. The vertices of $\mathrm{CH}(P)$ are a subset of the vertices of P. If the vertices on the boundary $V(P) = \{v_1^P, v_2^P, \ldots, v_n^P\}$ are numbered in clockwise direction, $\mathrm{CH}(P)$ can be represented as an increasing sequence $(i_1, i_2, \ldots, i_m)$ such that $\{v_{i_1}^P, v_{i_2}^P, \ldots, v_{i_m}^P\}$ are the m vertices of $\mathrm{CH}(P)$ in clockwise order. If $e^P = (v_j^P, v_{j+1}^P)$, let P_e^{CH} be a polygon bounded by the path $\mathbf{p}_k = v_{i_k}^P v_{i_k+1}^P \ldots v_{i_{k+1}}^P v_{i_k}^P$ where $i_k \leq j < i_{k+1}$. Since e^P is an edge of P, it is a boundary edge of P_e^{CH}. The convex hull edge $(v_{i_k}^P, v_{i_{k+1}}^P)$ is also on the boundary of P_e^{CH}. Finally, P_e^{CH} is not self-intersecting, or else P would be self-intersecting or the convex hull would not completely contain P. Given these characteristics, a path from e^P to $\mathrm{CH}(P)$ that does not intersect with P or $\mathrm{CH}(P)$ except at the terminal vertices must exist inside P_e^{CH}. ■

Combining this result with Lemma 5.3.1 yields the following result:

Lemma 5.3.5. *For a given fold pattern $(P, \mathcal{F})$ and exterior edge e^P in $(P, \mathcal{F})$, Algorithm 2 outputs a valid fold pattern that bridges an edge e^P to the boundary of the convex hull.*

Proof: Lemma 5.3.1 indicates that a set of pleats can always be created along a path such that the edges at the beginning and end of the path coincide in the folded state. Inputting a path where the beginning of the path lies on e^P and the end is on the boundary of the convex hull of $(P, \mathcal{F})$, a path that must exist according to

Lemma 5.3.4, therefore bridges e^P to the boundary of the convex hull. Since this path lies entirely in the free space, the midline of the pleat structure will never intersect the original fold pattern, so removing portions of the bridge that collide with the the fold pattern will never disconnect the bridge. ■

Lemma 5.3.6. *Algorithm 2 outputs a fold pattern with $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges.*

Proof: We use the straight skeleton to generate the path $\mathbf{p}$ along which pleats are constructed. When P_e^{CH} is an n_{CH}-gon, the straight skeleton is a tree with $(n_{\text{CH}} - 2)$ vertices and $(2n_{\text{CH}} - 3)$ edges that partitions the polygon into n_{CH} regions [3]. Thus, the length of the path $\mathbf{p}$ is at most $n_{\text{CH}} \sim \mathcal{O}(N)$. According to Lemma 5.3.2, this means that the bridge will have $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges. Merging the bridge into the fold pattern in line 3 decreases the number of total vertices by two and the number of edges by one. Finally, removing intersections and trimming pleats in line 4 can add at most N vertices and M edges to the fold pattern. Taking into account that $M \sim \mathcal{O}(N)$, the output fold pattern must have $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges.

■

Lemma 5.3.7. *Algorithm 2 computes a bridge in $\mathcal{O}\left(N^2 \log(N)\right)$ time.*

Proof: Line 1 of the algorithm finds a path to the boundary of the convex hull. This requires first computing the convex hull $\text{CH}(P)$, which can be done in $\mathcal{O}(N)$ time [108,115]. The region P_e^{CH} can be found by following the boundary of P starting at e^P until a vertex that is also a vertex of the convex hull is reached. Tracing this path will also take $\mathcal{O}(N)$ time and will result in a region P_e^{CH} bordered by a path of length $\mathcal{O}(N)$. The straight skeleton of P_e^{CH} will thus contain $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges [3], and it can be computed in $\mathcal{O}(N^{17/11+\epsilon})$ time [47]. Finally, the path $\mathbf{p}$ is found via a search over the straight skeleton tree that will, in the worst case, require traversal over all the edges in the tree, taking $\mathcal{O}(N)$ time. The length of path $\mathbf{p}$ is also $\mathcal{O}(N)$.

Line 2 calls Algorithm 3, which takes $\mathcal{O}(N^2)$ time for a path of length $\mathcal{O}(N)$ and produces a pleat structure with $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges. Line 4 requires

another check for intersections, this time between the boundary of the pleats and the the boundary of P. Since the only intersections will occur between the pleats and the fold pattern $(P, \mathcal{F})$, the number of intersections is $\mathcal{O}(NM)$, and line 4 will take at worst $\mathcal{O}((N + M + NM)\log(N + M)) \sim \mathcal{O}(NM\log(N + M))$ time [14], which is $\mathcal{O}(N^2\log(N))$ since $M \sim \mathcal{O}(N)$. Lines 3 and 5 are $\mathcal{O}(1)$ operations.

Summing over the entire algorithm yields an overall complexity of $\mathcal{O}(N^2\log(N))$ for Algorithm 2. ■

5.3.3 Bridging Interior Edges

Similarly, we provide correctness guarantees for interior edges.

Lemma 5.3.8. *For a given fold pattern $(P, \mathcal{F})$ and interior edge e^P in $(P, \mathcal{F})$, Algorithm 4 outputs a valid fold pattern that bridges an edge e^P to the boundary of the convex hull.*

Proof: Bridging any boundary edge e_b to the boundary of the convex hull as in line 1 of the algorithm is valid as shown in Lemma 5.3.5. Moreover, since the new edge e_{CH} created through this process is on the boundary of the convex hull of the resulting fold pattern, reflecting edges of the pattern over e_{CH} will not result in any self-intersection for the same reasons as Lemma 5.2.1, and it creates an exterior edge e^P_{ref} that in the folded state coincides with e^P. Bridging e^P_{ref} to the boundary of the convex hull results in a valid fold pattern (ref. Lemma 5.3.5) with an edge e^P_{new} that folds onto e^P_{ref} and therefore onto e^P. ■

Lemma 5.3.9. *Algorithm 4 outputs a fold pattern with $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges.*

Proof: According to Lemma 5.3.6, line 2 will increase the size of the fold pattern to have $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges. Since line 3 reflects only faces that already exist, it can at most double the size of the fold pattern. A final application of Algorithm 2 in line 4 yields an output fold pattern with $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges. ■

Lemma 5.3.10. *Algorithm 4 computes a bridge in $\mathcal{O}\left(N^2\log(N)\right)$ time.*

Proof: Line 1 of the algorithm chooses a boundary edge e_b. This requires constructing the edge-adjacency graph of the fold pattern in $\mathcal{O}(M^2)$ time, as well as straight skeletons in $\mathcal{O}(N^{17/11+\epsilon})$ time. The closest boundary edge to e^P can be found on the resulting $\mathcal{O}(M+N)$-vertex, $\mathcal{O}(M^2+N)$-edge graph in $\mathcal{O}((M+N)^2)$ time using Dijkstra's algorithm [43]. Since $M \sim \mathcal{O}(N)$ for a non-self-intersecting fold pattern, this whole procedure takes $\mathcal{O}(N^2)$ time.

Line 2 calls Algorithm 2, which by Lemmas 5.3.6 and 5.3.7 produces a fold pattern with $\mathcal{O}(N)$ vertices and $\mathcal{O}(N)$ edges in $\mathcal{O}\left(N^2 \log(N)\right)$ time. Since the path from e^P to e_b was already calculated in line 1, line 3 incurs only the cost of reflecting $\mathcal{O}(N)$ faces in $\mathcal{O}(N)$ time. Finally, line 4 takes $\mathcal{O}\left(N^2 \log(N)\right)$, according to Lemma 5.3.7.

Summing over all the lines yields an overall complexity of $\mathcal{O}\left(N^2 \log(N)\right)$ for Algorithm 4. ■

5.3.4 Full Composition Algorithm

We have now provided enough detail to support Theorem 5.2.3:

Lemma 5.2.2. *Given a fold pattern $(P, \mathcal{F})$ and an edge e^P, it is always possible to bridge e^P to the boundary of the convex hull. The bridge can be constructed in $\mathcal{O}\left(N^2 \log(N)\right)$ time.*

Proof: The previous sections detail how to construct a bridge when e^P is a an exterior or interior edge. The edge to join e^P must fall into one of these cases. It is therefore always possible to bridge e^P to the convex hull.

Complexity: The algorithm for interior edges (Algorithm 4) subsumes that for exterior edges (Algorithm 2). Since the complexity of the two algorithms is the same, it is not unreasonable to use Algorithm 4 all the time. The overall complexity of constructing a bridge for arbitrary e^P is $\mathcal{O}\left(N^2 \log(N)\right)$. ■

Theorem 5.2.3. *For any fold patterns $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$, and edges e^{P_1} in $(P_1, \mathcal{F}_1)$ and e^{P_2} in $(P_2, \mathcal{F}_2)$, Algorithm 1 produces a fold pattern $(P_3, \mathcal{F}_3)$ that satisfies Problem 5.1.2. Furthermore, this fold pattern has $\mathcal{O}(N_1 + N_2)$ vertices and $\mathcal{O}(N_1 + N_2)$ edges, and it can be computed in $\mathcal{O}(N_1^2 \log(N_1) + N_2^2 \log(N_2))$ time.*

Proof: According to Lemma 5.2.2, edges e^{P_1} and e^{P_2} can always be bridged to the boundaries of the convex hulls of P_1 and P_2 respectively. Let $e^{P_1}_{new}$ be the bridge edge on the boundary of the convex hull that coincides with e^{P_1} in the folded state, and similarly with $e^{P_2}_{new}$. Applying Lemma 5.2.1 to the modified $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$ using $e^{P_1}_{new}$ and $e^{P_2}_{new}$ as the edges to join yields a fold pattern $(P_3, \mathcal{F}_3)$ that satisfies the conditions of Problem 5.1.2.

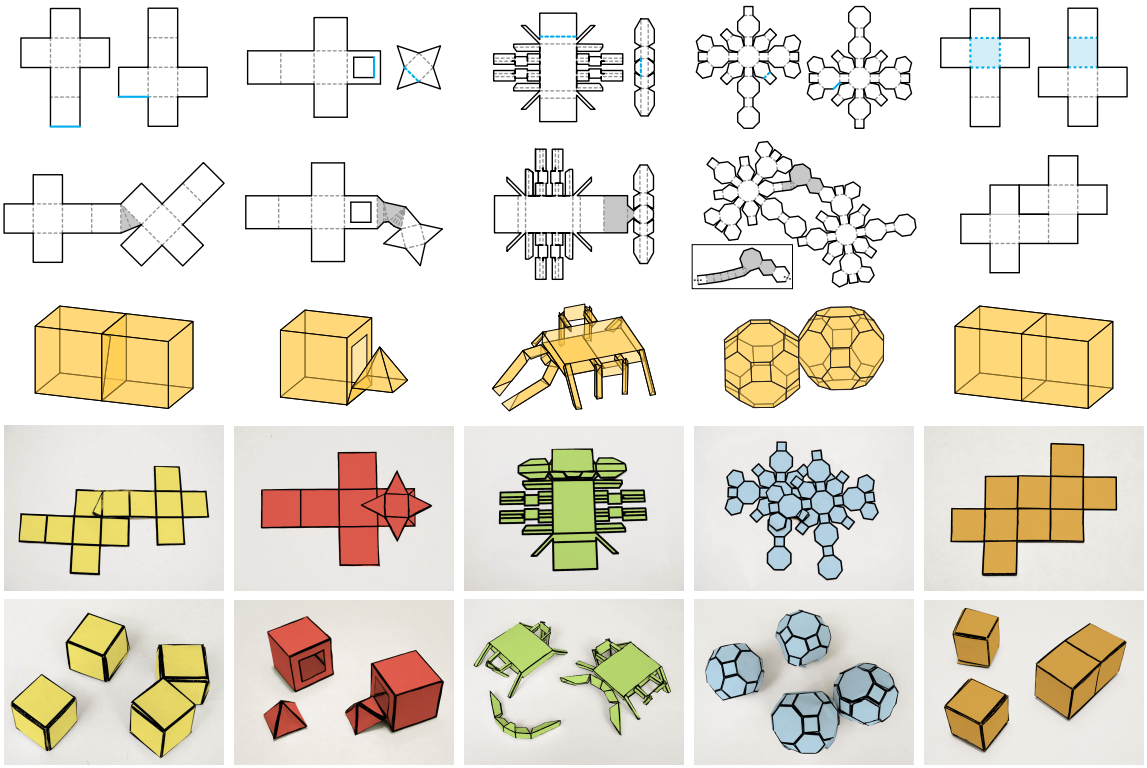
Complexity: According to Lemma 5.2.2, bridging takes $\mathcal{O}\left(N_i^2 \log(N_i)\right)$ time for each fold pattern. Rotating and translating the modified fold patterns so that the new edges to join are coincident takes $\mathcal{O}(\min(N_1, N_2))$ time (we only have to transform one fold pattern). Thus the entire composition algorithm has complexity $\mathcal{O}\left(N_1^2 \log(N_1) + N_2^2 \log(N_2)\right)$. ■

5.4 Experimental Results

We have implemented a system in MATLAB in which users can explore compositions of folded structures. In addition to generating joints, users can input custom patterns for folded structures as a vector file. They can then specify the edges or faces on individual folded structures that they wish to connect, and the system will combine the fold patterns of both into a single-sheet pattern for the composed structure. The system provides views of both the flat fold pattern and its folded state in 3-D so that users can visually verify that the composition is correct. We have tested this system for various modules, joint combinations, and linkage mechanisms.

5.4.1 Compositions of Rigid Bodies

We have tested the composition algorithm on a variety of rigid bodies. Figure 5-9 shows the input fold patterns and the composition for each test. Cut lines on the boundary of the fold patterns are shown as solid, and folds are shown as dotted lines. The edges to join are highlighted in bold on the input fold patterns. The expected folded states of each generated fold pattern are simulated and verified via physical models folded from poster board.



(a) Two cubes (b) Cube and square pyramid (c) Walking and gripping robots (d) Two truncated cuboctohedra (e) Face joining, two cubes

Figure 5-9: Fold patterns generated by this algorithm. *Top*: Input fold patterns for the rigid bodies to join. The bold edges are the edges to join. *Second row*: The generated composite fold patterns. Bridges constructed by our algorithm are shaded. *Third row*: The folded states of the composite fold patterns. *Fourth row*: Physical model of the composition with just the bridge folded. *Bottom*: Physical models of the input surfaces and the composition folded from poster board.

All final folded structures are, as expected, the two inputted surfaces connected along the specified edge. The constructed bridges (second row, shaded) are a series of pleats connecting the edges to join to the boundaries of the convex hulls of their respective fold patterns. The fourth row of Figure 5-9 shows the compositions with only the bridges folded. The edges to join coincide in the folded state.

Figure 5-9(a) joining two cubes is an example of Algorithm 2. Both edges to join are on the boundaries of their fold patterns. Pleats are added to the fold pattern on the right since the edge to join is not already on the boundary of the convex hull. In the folded state, these pleats are flattened between the two cubes. Figures 5-9(b)–(c) demonstrate Algorithm 4, when an edge to join is on the interior of the fold pattern.

The square pyramid in Figure 5-9(b) is the same one considered in Figure 5-6. Not only are pleats added but a face of the fold pattern is reflected in the bridge construction. In the folded state, this face lies flat against the surface of the square pyramid so that one side is doubly covered. The cube in this composition is an example of an interior boundary edge. This time, when Algorithm 4 is used, only part of the face containing the hole is reflected to place the new edge on the boundary of the fold pattern. The result is that a trapezoidal section of the original face is doubly covered.

Faces are also reflected in the bridge for the insect and gripper in Figure 5-9(c). Compared to Figure 5-1, the outputted pattern is similar. Human designers creating the composed pattern made optimizations based on rearranging faces on the original fold patterns, which our algorithm did not consider.

Practically, additional factors other than connectivity between the two input fold patterns must be taken into account when generating bridges. Sometimes, even if e^P is on the boundary, taking a longer path through the interior of the fold pattern to an area with a greater amount of free space may yield wider pleats or fewer added layers, so we may want to use Algorithm 4 even if Algorithm 2 is applicable. This was the approach used for the right fold pattern in Figure 5-9(d).

Figure 5-9(d) also demonstrates how pleats in the bridge can be removed to minimize added material. The box in the lower left of the composite fold pattern shows the bridge as would be constructed by the full algorithm. Some of these pleats can be removed without causing self intersection of the fold pattern while still producing a valid 3-D structure. Note that reflected faces can never be removed unless the bridge is connected to a different boundary edge.

Since the joined edges act as a hinge joint, the angles of the input structures relative to each other in the folded state are not fixed. This effect can be clearly seen in the physical models, where the stiffness of the material used prevented faces from resting coincident as they do in the simulated folded states. Composing fold patterns at faces instead of edges can be used to restrict the positions of the two structures. Figure 5-9(e) shows the result of composing the two cubes from Figure 5-9(a) along a face to create a rectangular prism. The composition was optimized to minimize

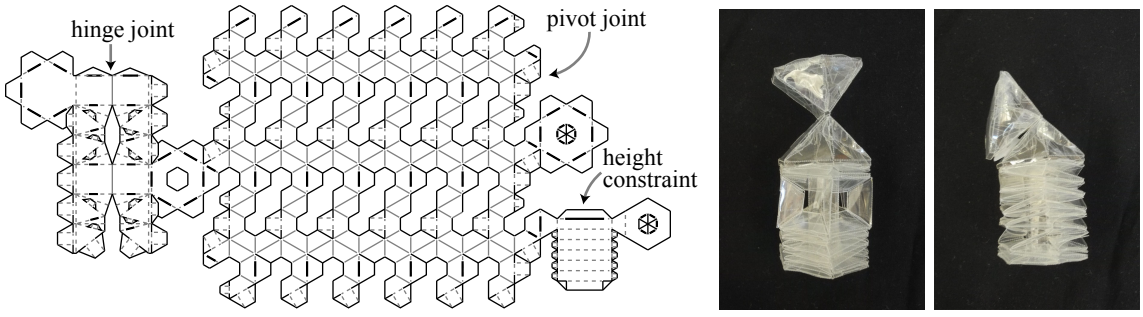
bridge length. In this case, the two fold patterns were able to be connected without adding any extra material.

5.4.2 Joints with Multiple Degrees of Freedom

Complex joints with higher degrees of freedom can be created by composing our basic joints from Chapter 4. In some cases, extra degrees of freedom come for free. For example, the designs for the prismatic and pivot joints each allow vertical translation in addition to the intended motion when the height of the joint is not constrained. Thus we can produce a cylindrical joint simply by not including a distance constraint between the outer faces of the pivot joint. When joints with higher degrees of freedom are created in this way, the joint limits for one degree of freedom will often depend on the position along the others.

For more independent ranges of motion, joints can be combined. For example, a universal joint is two hinge joints with orthogonal axes of rotation connected in series. A spherical joint is a pivot joint combined with a hinge. Since our designs are parameterized, basic joints can be adjusted for simpler joining (e.g., by having the same base) without restricting the joint limits.

We tested joint composition by generating a spherical joint, shown in Figure 5-10. The composed 6-sided joint consists of a pivot joint with a $\frac{10}{3}\pi$ radian (600°) range of motion attached to a 6-sided hinge joint with a π radian (180°) range of motion. Since the joint designs are already quite complex, we used the optimization that if the



(a) Composed fold pattern (with tabs and slots)

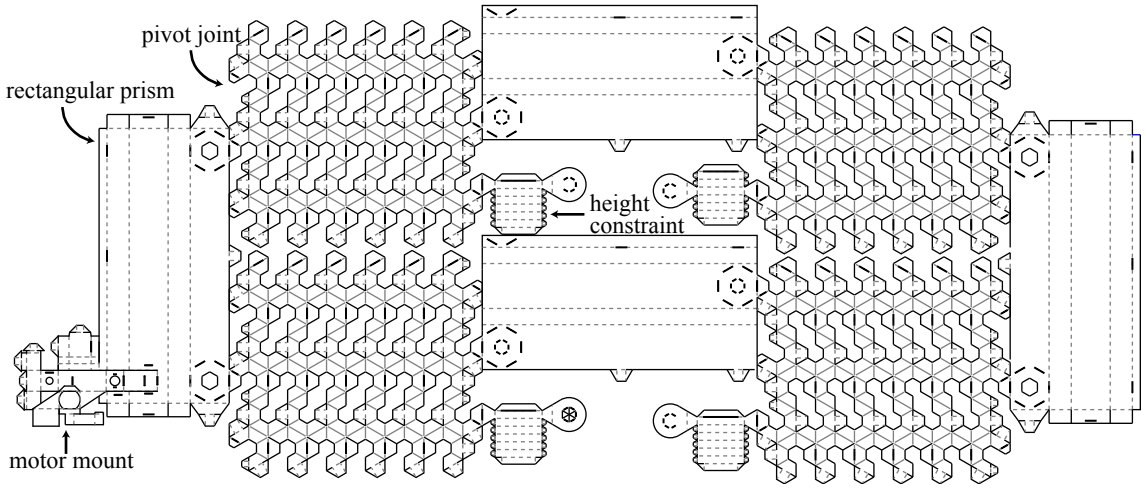
(b) Folded joint in two positions

Figure 5-10: Spherical joint composed from 6-sided pivot and hinge joints

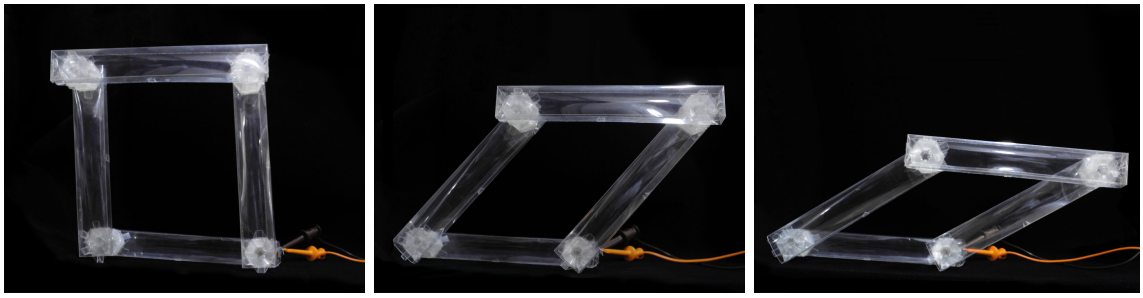
fold patterns do not intersect without the addition of bridges, then they are simply attached together at the chosen edges. Tabs were added manually after composition to simplify assembly. The fold patterns for the individual subcomponents are readily visible in the combined fold pattern for the entire joint, shown in Figure 5-10(a). Since the axes of rotation of the pivot and hinge joints intersect at the center of the hinge joint, the resulting joint approximates well the behavior of a spherical joint, despite the pivot and hinge joint being two separate entities.

5.4.3 Mechanisms

Rigid bodies can be composed with joints to produce entire foldable linkages. We used our system to compose a four-bar linkage (Figure 5-11(a)). This linkage consists of four rectangular prisms connected in a cycle using four pivot joints. A tubular

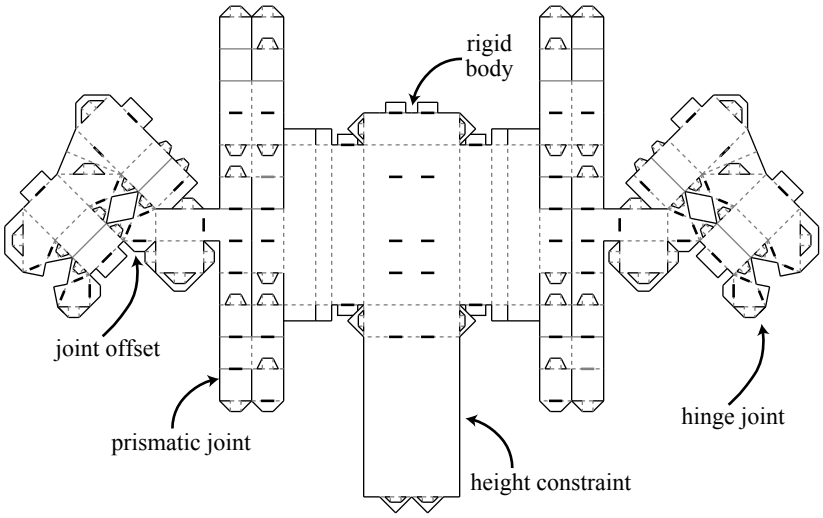


(a) Composed fold pattern

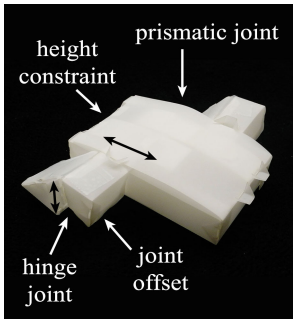


(b) Movement of four-bar

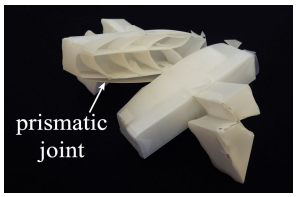
Figure 5-11: Foldable four-bar linkage



(a) Composed fold pattern



(b) Folded rowboat



(c) Section cut

Figure 5-12: Foldable rowboat

structure similar to that in Figures 4-10(b) and 5-10 was used to constrain the height of the pivot joints. We also manually designed a motor mount (also used in the hinge in Figure 4-11) to actuate one joint and the linkage. The linkage was folded from 0.127 mm thick polyester film perforated at the fold lines. Frames of the resulting motion are shown in Figure 5-11(b). Since the joint angles are limited, the joint positions must be initialized carefully when folding to allow movement for the entire linkage.

In a second test, we composed two prismatic joints, two hinge joints, and a rectangular body to form a rowboat, shown in Figure 5-12. The prismatic joints lie inside the rectangular body (Figure 5-12(c)) and attach along the left and right edges. The top of the rectangular body forms the height constraint for the prismatic joint and enforces purely horizontal motion. On the other base of each prismatic joint is a paddle. Paddles are hinges joints with a $\frac{\pi}{2}$ radian (90°) range of motion mounted at a $\frac{\pi}{4}$ radian (45°) angle relative to the body of the boat to produce asymmetric joint limits. Thus, the paddles can lie horizontally over the water or can extend vertically down into the water to provide thrust.

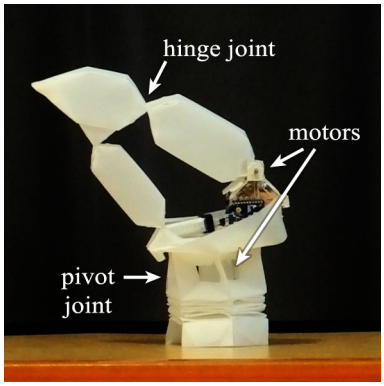


Figure 5-13: Actuated four-bar linkage atop actuated pivot mount

5.4.4 Foldable Robots

Crane When our system composes folded structures, the original fold patterns are preserved in their entirety. This feature enables us to reuse circuit layouts. Starting from the actuated hinge from Section 4.3.1, we added square bars and three more 4-sided π radian (180°) hinge joints to produce a four-bar linkage, and we composed the entire linkage with a 6-sided 2π radian (360°) pivot mount to produce a machine with kinematics similar to those of a manufacturing crane. To the hinge circuitry, we added the additional control circuitry for a DC motor mounted at the center of the pivot joint. Finally, we added circuitry for serial communication via a Digi XBee radio module so that the robot could be controlled wirelessly. Commands were sent from a laptop, which controlled the direction of rotation of the pivot mount and the position of the hinge joint.

The robot was constructed from 0.127 mm thick polyester film. It had a base radius of 36 mm and a height of 213 mm. During testing, the hinge joint on the robot performed exactly as the original hinge joint did, although its range of motion was constrained due to the additional hinges and links attached to it. The pivot joint was able to achieve its full 2π radian (360°) range of motion.

Camera Mount As a final test, we composed a mount for a smartphone using the spherical joint shown in Figure 5-10, yielding a camera with pan-tilt capabilities, and we actuated both degrees of freedom independently using off-the shelf servos. Again

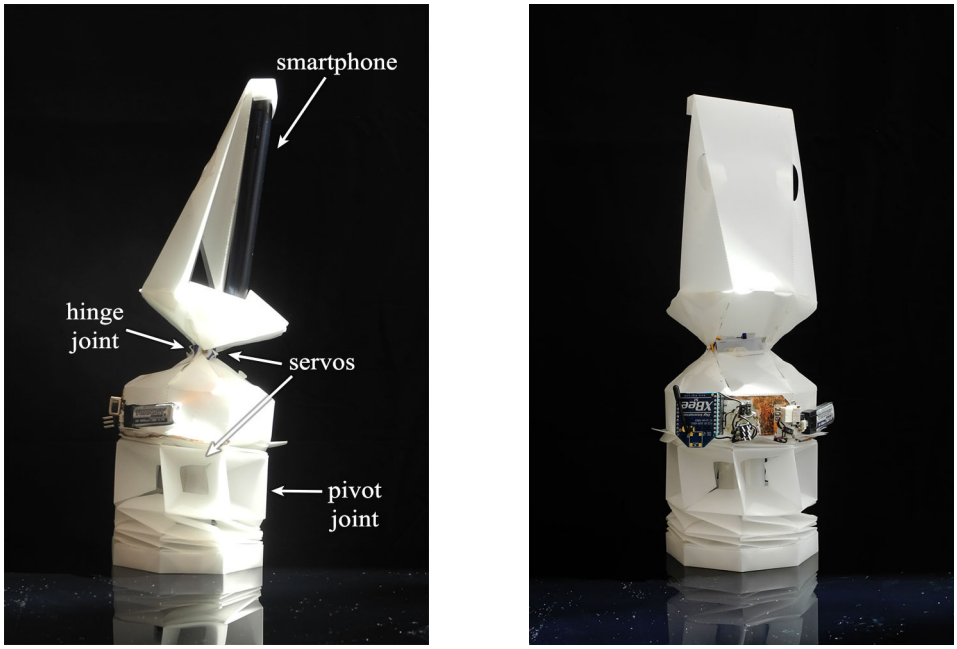


Figure 5-14: Smartphone mount attached to actuated spherical joint for pan and tilt

circuitry was designed by hand and etched directly onto the robot body, except for the servos, which were plugged into headers in the circuit. A laptop sent commands to tilt forward or backward or to pan left or right via serial communication through a Digi XBee radio module. The camera mount had a base radius of 43 mm and a total height of 253 mm.

The spherical joint in the camera mount was designed for $\frac{3}{2}\pi$ radians (270°) of pan and $\frac{2}{3}\pi$ radians (120°) of tilt. During testing, the camera was able to achieve the full $\frac{3}{2}\pi$ radians (270°) of pan. However, since small servos are typically not designed to sustain large loads such as the weight of a smartphone, tilt had to be limited to $\pm\frac{\pi}{4}$ radians (±45°) in order to maintain controllability when a smartphone was in place. In addition, since over half of the weight of the device lay above the hinge joint (camera mount: 91 g, smartphone: 116 g), the mount would bend or topple when large tilt angles were attempted.

5.5 Summary

We have described an algorithm for automatic composition of fold patterns for 3-D surfaces. We show that given the fold patterns of two 3-D surfaces, it is always possible to construct a bridge between them such that the resulting structure is the two originals connected along a fold, and we provide a polynomial-time algorithm to generate the composite fold pattern. The algorithm was tested on a variety of geometries, demonstrating that it can indeed be used to generate complex one-piece fold patterns for robot designs.

Although we have shown that composite fold patterns can be constructed, our algorithm does not address the folding sequence for these structures. While the two original fold patterns can be assumed to be foldable, it is entirely possible that the addition of a bridge renders the result unfoldable. Practically, we have never encountered this difficulty with folding sequences. Future work includes further investigation on this issue.

Finally, the results of this algorithm are restricted in that the generated fold pattern must contain $(P_1, \mathcal{F}_1)$ and $(P_2, \mathcal{F}_2)$ in their entirety. When humans compose origami designs, they often rearrange the fold patterns and change the shape of the free space to achieve more efficient composite fold patterns (ref. Figure 5-1). However, with robots folded from thin materials, strength, stiffness, and an actual ability to transfer and withstand high forces and torques becomes a concern, so depending on the application, certain folds in a pattern should not be cut. Still, a 3-D surface often has several equivalent fold patterns that yield the same mechanical strength. We would like to incorporate models such as those in Section 4.2 to explore equivalent fold patterns to be used in composition.

Chapter 6

Composition of Foldable Ground Robots

In order to address concurrent design of robot geometry and motion, our database must support motion information in addition to the mechanical modules presented in Chapter 4. We choose to focus on ground locomotion and the design of walking gaits. In this chapter, we describe our grammar-based approach to gait suggestion for ground robots. The grammar is based upon a classification of robot parts, and the gaits are parameterized to accommodate different geometries. Since the joint controllers used are modules, they enable us to automatically generate the electronics and software required for assembly. The system outputs a full fabrication plan, including the parts for assembling the robot's body, the required electronics, and the control software. Since the modules are parameterized, the designs can be optimized for desired performance metrics. We provide virtual optimization results, as well as full fabricated robots composed using these methods.[1]

[1] This work was done in collaboration with Adriana Schulz, Andrew Spielberg, Wei Zhao, Robin Cheng, Eitan Grinspun, Wojciech Matusik, and Daniela Rus.

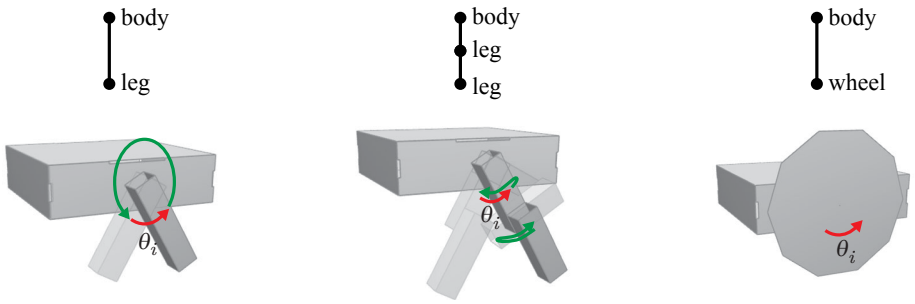
6.1 Gait and Trajectory Design

For ground locomotion, we use a grammar-based approach to define the motion at each joint. We have classified geometric modules into three categories: robot bodies, limbs (wheels and legs), and peripherals. Each of these modules is associated with a kinematic chain that defines the degrees of freedom of the module. Composing the modules yields a full kinematic chain equivalent to the robot design. Under our classification scheme, connections between limbs and other modules are dynamic and associated with a particular joint controller, while all other connections are static.

In order to reduce the complexity of gait design, we have chosen a gait that consists of two phases: a step phase and a reset phase. During the step phase, limbs take turns lifting off and re-situating. During the reset phase, all limbs move simultaneously to shift the robot's joint angles back to their pre-step configuration. When the robot is stable and does not rely on body contact with the ground to move, these phases correspond to the swing and stance gait phases commonly used in the literature [8,33], the main difference being that in our robots, the stance phases for all limbs occur simultaneously.

Our grammar differentiates between single-link legs that are attached to robot bodies, multi-link legs, and wheel connections. The joint motions that occur under each controller are shown in Figure 6-1, along with the equivalent kinematic chain. Green arrows indicate the joint trajectory during the step phase. Red arrows indicate the trajectory during the reset phase.

Unlike previous systems that allow users to specify full trajectories for limbs [10, 110], we have chosen to design joint controllers that each have only one free parameter. Each joint controller has a single parameter θ_i, which controls the magnitude of the angle swept out during the reset phase. Negative values for θ_i indicate that the joint moves in the opposite direction. For a stable robot, the value of θ_i controls the distance the robot moves forward during one gait cycle.



(a) Single-link leg (b) Multi-link leg (c) Wheel

Figure 6-1: Joint controllers for single-link leg, multi-link leg, and wheel joint types. The diagram on top shows the equivalent kinematic chain. Controllers for every joint are separated into a step phase (green) and a reset phase (red). Gaits are changed by modifying the θ_i values for each joint and defining the step sequence.

6.1.1 Joint Controllers

Single-link Leg Controller When single-link legs are connected directly to bodies, they follow a circular joint motion. The leg starts out at an angle of $\frac{\theta_i}{2}$ from vertical. It sweeps out an angle of $(2\pi - \theta_i)$ during the step phase and an angle of θ_i during the reset phase to complete one full rotation every gait cycle.

Multi-link Leg Controller In multi-link legs, the links attached to the robot body are differentiated from links attached to other links. We call the joints that attach leg links to the body "shoulder joints" and all other joints "elbow joints." The joint controllers for each of the joints on the leg are related and perform synchronized movements.

The leg starts out with the shoulder joint at an angle of $\frac{\theta_i}{2}$ from vertical and all other joints at an angle of 0 so that the leg points straight out from the body at an angle of $\frac{\theta_i}{2}$. During the step phase, the shoulder joint first rotates to an angle of 0.6π radians (108°) while the elbow joints rotate to an angle of $-\theta_i$ to lift the leg off the ground. Following, the shoulder joint swings forward to an angle of $-\frac{\theta_i}{2}$ while the elbow joints return to an angle of 0 to bring the leg back into contact with the ground in a new location. During the reset phase, the shoulder joint sweeps an angle of θ_i to bring the leg back to its starting position while the elbow joints remain still.

Wheel Controller The wheel controller is the simplest of all joint controllers. Since wheels cannot lift off the ground unless other parts of the robot move, they do not experience a step phase. Instead, during the reset phase, the wheels simply sweep out an angle of θ_i to shift the robot forward.

6.1.2 Gait Design

Gaits are compositions of joint controllers. In particular, gaits can be defined using two types of parameters. For each of the limbs i on the robot, modifying the θ_i value for the controllers corresponding to that limb will affect the robot's local motions. In addition, the step sequence of limbs can be changed to affect the robot's global motions.

The step sequence for a gait indicates the order of limb movement. During the step phase, limbs can move in groups or separately. The number of groups of limbs is equal to the total number of steps per gait cycle N_g. Each limb is given a step assignment g_i, an integer between 0 and N_g that indicates at what time during gait the joints should execute the part of the motion corresponding to the step phase. A step assignment of 0 indicates that the limb does not execute a step phase and is reserved for wheels. Furthermore, steps that are not associated with any limbs take zero time and are effectively ignored.

6.1.3 Trajectory Design

For a particular robot geometry, multiple gaits can be defined, each with different θ_i parameters or different step sequences. The gaits can be composed into a sequence to define entire trajectories. Since consecutive gaits in the sequence may have different θ_i parameters and therefore different starting configurations, short transitions are added automatically between gaits to take joints from the post-reset configuration of the previous gait to the pre-step configuration of the subsequent gait as fast as possible.

6.2 Performance Metrics

We provide simulations and metrics that enable users to evaluate their combined geometry and gait designs during composition.

6.2.1 Simulation

The robot models are evaluated using a sequence of rigid body simulations. In particular, we discretize time into time steps. The procedure is given in Algorithm 5. At each time t_i, the configuration of the robot is first computed by updating the joint angles at each of the robot's joints (line 5). Next, the robot's orientation is found by placing the lowest points of the robot on the ground and iteratively pivoting the robot about its contact points until it is statically stable (lines 9–13). The direction of rotation is determined by computing the effect of a downward gravitational force applied at the robot's center of mass. The robot is statically stable when the projection of its center of mass onto the ground plane lies within its support polygon (i.e., the convex hull of the contact points with the ground). Timing information for the gait is computed based on the parameter values and the servomotor's maximum angular speed (3.3π radians/s, or $600°$/s).

We approximate friction by assuming that the robot has slipped as little as possible. Let $\mathbf{c}_{t_i}$ be the set of contact points at time t_i and $\mathbf{c}_{t_{i-1}}$ be the contact points in the previous time step. Our system computes the new pose of the robot by finding a least-squares rigid transformation between the locations of the contact points common to both time steps and applies the transformation to the robot geometry (lines 14–15). Let $\mathbf{c}'_{t_i}$ and $\mathbf{c}'_{t_{i-1}}$ be the contact points at times t_i and t_{i-1}, respectively, that are common to both time steps. The applied rigid transformation T is the one that minimizes the objective $\|\mathbf{c}'_{t_i} - T\mathbf{c}'_{t_{i-1}}\|_2^2$. When multiple solutions exist with equivalent objective values, the system chooses the one that minimizes the magnitude of the applied rotation.

Since the robot may exhibit different behaviors over multiple gait cycles (e.g., toppling), we iterate over the process until the robot has reached a steady state

Algorithm 5: SIMULATE($\mathbf{D}, \Delta t$)

Data: $\mathbf{D}$ = robot geometric design
Δt = time step for simulation
Result: t_{i_0} = start time for steady state robot motion
t_{i_f} = end time for steady state robot motion
$\{\mathbf{D}_t\}$ = sequence of states of robot from $t = 0$ to $t = t_{i_f}$

$i = 0$; $t_0 = 0$;
$\mathbf{c}_0 = \{\}$; `// contact points`
while *NOT steady state* **do**
    $i = i + 1$; $t_i = t_{i-1} + \Delta t$;
    $\mathbf{D}_{t_i} \leftarrow$UPDATEJOINTANGLES($\mathbf{D}, t_i$);
    `// Stabilize`
    Place lowest points of $\mathbf{D}_{t_i}$ on the ground plane;
    $\mathbf{c}_{t_i} \leftarrow$ current contact points;
    $COM_{t_i} \leftarrow$ center of mass;
    while $\mathbf{D}_{t_i}$ *not statically stable* **do**
        Apply gravitational torque to $\mathbf{D}_{t_i}$ until there is a new contact point;
        $\mathbf{c}_{t_i} \leftarrow$ current contact points;
        $COM_{t_i} \leftarrow$ center of mass;
    end
    `// Minimize slip`
    $T \leftarrow$rigid transform that minimizes sum of squared distances between points repeated in $\mathbf{c}_{t_i}$ and $\mathbf{c}_{t_{i-1}}$;
    Apply T to $\mathbf{D}_{t_i}$;
end
$t_{i_f} \leftarrow t_i$;
$t_{i_0} \leftarrow$ maximum t_i such that $\mathbf{D}_{t_i}$ is equivalent to $\mathbf{D}_{t_{i_f}}$ up to translation and rotation about $\hat{\mathbf{z}}$

behavior. This can be detected by comparing the pose of the robot at the beginning of a gait cycle to its pose at the beginning of every previous gait cycle. If its pose differs by only a translation and a rotation about the vertical axis $\hat{\mathbf{z}}$, then the robot has reached a steady state.

Our system stores information about the start time t_{i_0} and end time t_{i_f} for one iteration of the robot's steady state behavior for metric evaluation. The pose of the robot at any time after t_{i_f} can be calculated as a rigid transformation of a pose during this time period.

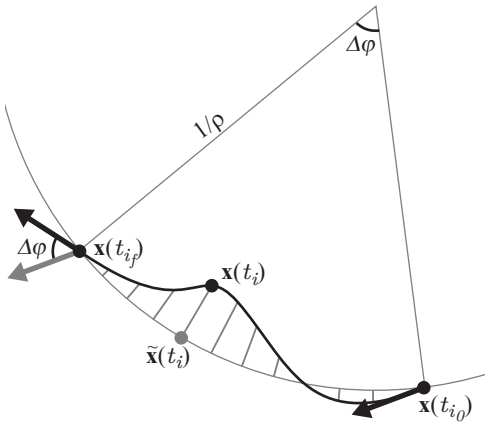


Figure 6-2: Metrics are calculated on the trajectory (black) of the robot over one steady state gait cycle using the indicated values. The point $\mathbf{x}(t_i)$ is the projection of the robot's center of mass onto the ground plane at time t_i. We can approximate the trajectory with a circular arc (gray).

6.2.2 Metrics

The simulations allow us to report relevant metrics about a design and compute information about the robot's expected performance for optimization. All metrics on the designed gaits are computed for the robot's steady state behavior for that gait. Figure 6-2 shows an example trajectory of a robot over one steady state gait cycle in black and indicates relevant values. The point $\mathbf{x}(t_i)$ is the position of the robot's center of mass projected onto the ground plane at time t_i. The robot starts at $\mathbf{x}(t_{i_0})$ with the heading indicated by the black arrow and ends at $\mathbf{x}(t_{i_f})$ with the indicated heading. We can approximate the robot's trajectory as a circular arc.

The system evaluates the following metrics about the robot's overall performance.

1. Stability: The stability S of the robot during its trajectory is evaluated as the minimum distance from the projection of the robot's center of mass onto the ground plane to the boundary of the robot's support polygon at any time step. This measure gives designers an idea of how much fabrication and assembly error their design can handle before the behavior changes drastically. Stability is reported in mm.

2. Speed: The average speed V of the robot in steady state is computed as the distance traveled during one iteration of steady state behavior, divided by the amount of time required to traverse that distance:

$$V = \frac{1}{t_{i_f} - t_{i_0}} \sum_{i=i_0+1}^{i_f} \|\mathbf{x}(t_i) - \mathbf{x}(t_{i-1})\|_2$$

Speed is reported in mm/s.

3. Wobbliness: The average wobbliness W of the robot is computed as the amount of orientation variation the robot experiences over one iteration of steady state behavior:

$$W = \frac{1}{i_f - i_0} \sum_{i=i_0+1}^{i_f} |\Delta\gamma\left(t_{i-1}, t_i\right)|$$

where $\Delta\gamma\left(t_{i-1}, t_i\right)$ is the combined pitch and roll angular change between time t_{i-1} and time t_i. Wobbliness is reported in radians.

4. Slip: The average slip of the robot is the root mean square of errors between contact points during the reorientation step of the simulation.

$$E = \frac{1}{i_f - i_0} \sum_{i=i_0+1}^{i_f} \sqrt{\frac{1}{n_i} \sum_j \|\mathbf{c}_i^j - \mathbf{c}_{i-i}^j\|_2^2}$$

where n_i is the total number of contact points that remain in contact between times t_{i-1} and time t_i, and $\mathbf{c}_i^j$ is the location of one of those contact points j at time t_i. Slip is reported in mm.

The following metrics are also computed for individual gaits.

1. Angle of Rotation: The robot's change in heading is computed over one gait cycle.

$$\Delta\varphi = \frac{1}{i_f - i_0} \left(\varphi(t_{i_f}) - \varphi(t_{i_0})\right)$$

where $\varphi(t_i)$ is the heading of the robot at time t_i. Angle of rotation is reported in radians.

2. Curvature: The average curvature of a gait is the reciprocal of the robot trajectory's radius of curvature and is computed as

$$\rho = \frac{2\sin(\Delta\varphi/2)}{\|\mathbf{x}(t_{i_f}) - \mathbf{x}(t_{i_0})\|_2}$$

Curvature is reported in 1/m.

3. Variance: The variance of a trajectory is calculated assuming that the robot ideally follows a circular path with curvature ρ and uses the perpendicular error from that path.

$$\sigma^2 = \frac{1}{i_f - i_0}\sum_{i=i_0}^{i_f} \|\mathbf{x}(t_i) - \tilde{\mathbf{x}}(t_i)\|_2^2$$

where $\tilde{\mathbf{x}}(t_i)$ is the closest point to $\mathbf{x}(t_i)$ on the circular trajectory. Variance is reported in mm^2.

6.3 Fabrication and Assembly

Because robot designs are composed from existing modules, we can automatically generate fabrication plans for physical prototyping. These plans include a list of electronics and port connections, software to load onto the microcontroller, and fabrication instructions for the robot's body. Appendix A shows an example of an outputted fabrication plan. The designer is responsible for assembling the robot model.

6.3.1 Electronics

In order to implement a designed trajectory, a robot requires actuators and control circuitry. The electronics plan is generated based on the kinematic tree of the robot design. Our control circuitry uses an Arduino Pro Mini and is based on a standardized electronics module described previously in [112]. First, one servomotor is assigned to each degree of freedom. For these robots, we use the Turnigy TGY-1370A servomotors, modified to also output a position signal. The robots are powered by a 3.7 V lithium ion battery. Connections between the servos, batteries, and the Arduino are

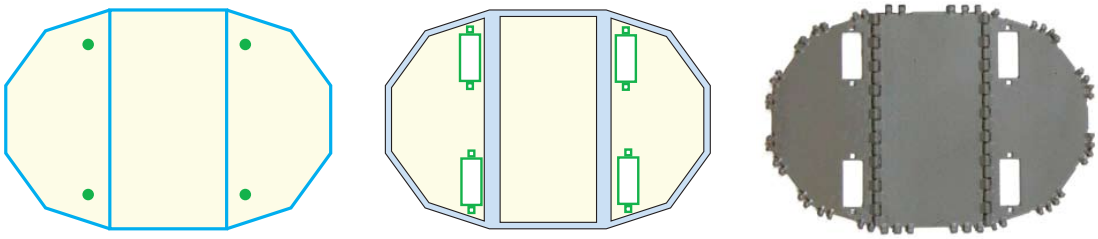
assigned using a greedy approach that matches pin types. Connections such as those for power and ground can connect multiple components, while connections sending control signals to servomotors must be one-to-one. Depending on the number of servos and the available ports, more microcontrollers may be added.

6.3.2 Software

Because our gaits are based on joint controller modules, the software for the robot can be generated easily using a software template. The parameters θ_i, the step sequences that define the gaits, and the gait sequence that forms the desired trajectory are written to a gait definition file. The rest of the software is general code that implements the trajectory given the parameters.

6.3.3 Robot Body

Robot bodies could be cut out of thin plastic film as in Chapters 4 and 5. In this chapter, to make sturdier robots, we leverage the versatility and rigidity of 3-D printing to fabricate fold patterns. All the modules in our database are foldable and are associated with fold patterns, which are combined during geometry composition. We convert this pattern into a 3-D printable mesh (Figure 6-3). Faces are first shrunk along the normals of folds and cut lines to make room for printed connections. Holes



(a) Input design (b) Process connections (c) Final print

Figure 6-3: We automatically generate a 3-D mesh for a foldable design. (a) The original design, with blue edges indicating folds and gluings and green dots indicating joint connections. (b) Connections are processed by shrinking faces to make room for hinges and adding holes for servo mounts. (c) Hinges are added and the result is a complete mesh that is 3D printed.

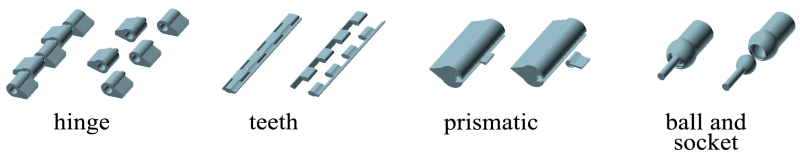


Figure 6-4: Printable, snappable connections used in 3-D mesh generation

for mounting servomotors and servo horns are also added to each of the part pairs forming a joint connection. The faces are then extruded to 1 mm, which we chose as the thickness that produces rigid faces while still allowing almost π radian (180°) fold angles.

Using 3-D printing instead of traditional planar manufacturing techniques gives us greater flexibility in the types of articulated structures that can be fabricated. Our folds are implemented as printed hinges. The hinge design is parameterized according to its location in the overall model, its length, and its angle. It is also snappable so that faces that are not adjacent in the 2-D pattern can still be connected in the folded state without having to add extra tabs and slots as we did in Section 5.4. The hinges snap to the correct fold angle during assembly. If the desired fold angle between two adjacent faces is zero, the two faces are simply merged without hinge.

In addition to hinges, we have designed three other printable, snappable joints, shown in Figure 6-4. Meshing teeth provide an alternative to hinges when faces that are not adjacent in the fold pattern must be connected in 3-D. Teeth generally provide a more rigid connection and take less volume than hinges. We have also provided designs for prismatic joints and ball and socket joints for other types of motion. The geometry of all joints are parameterized to fit in the fold pattern and provide the necessary range of motion.

Finally, 3-D printing enables certain parts to be printed as solid objects instead of folded ones. Elements such as feet and wheels are printed as solid objects with larger thickness in order to provide more rigidity. This information is incorporated into the description of the module design.

6.3.4 Assembly

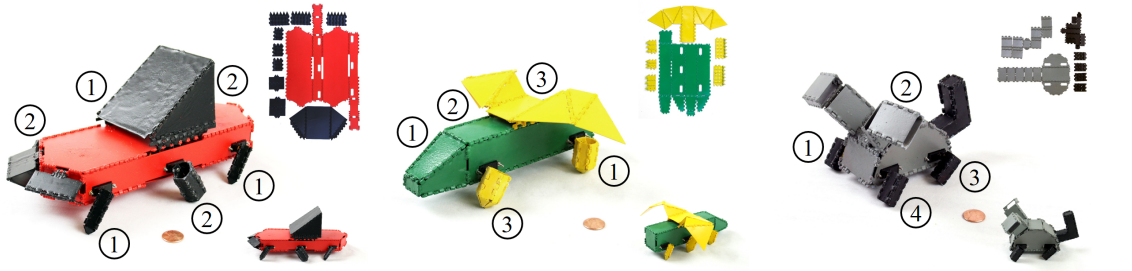
During assembly, the servomotors and servo horns are first screwed to their mounting holes. The electronic components are connected together, and the software is loaded onto the microcontroller. Each of the individual pieces of the robot body is then folded and snapped together. The final step is to snap the servo horns onto the output shafts of the servomotors to complete the design.

6.4 Experimental Results

6.4.1 Fabricated Robots

We tested the full design pipeline by composing and fabricating 6 robot models. The robots demonstrate a range of geometries and gaits, as shown in Figure 6-5. They each took 10–15 min. to design, 3–7 hr. to print, and 30–90 min. to assemble (ref. Table 6.2). The circled numbers in the figure indicate the step sequence. The mouse, dragon, and ant are each robots that use multiple single legs to walk. The mouse uses a gait that steps with one leg at a time, while the dragon and ant move their legs in groups. The ant uses a standard tripod gait commonly seen in robotic hexapods. The wheeled car robot is an example of locomotion with a rigid body and wheels that rotate continuously to move it forward. Two of the robots demonstrate combinations of limb types. The house has two front legs and two back wheels. It moves each of its two legs forward individually but rotates its wheels simultaneously to shift forward. The monkey has double-link front legs and single back legs, and it uses a similar step sequence to the mouse but with a different ordering of legs.

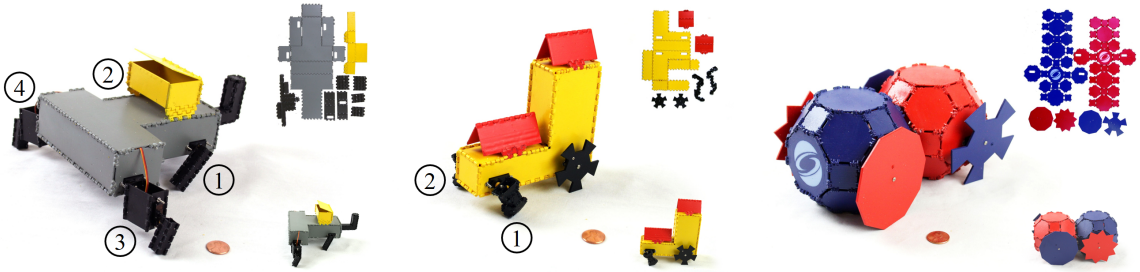
The motions performed by the robots were similar to those simulated by the system. Figure 6-6 shows frames from the simulation of the monkey compared with the physical robot. Each of the legs moves at the expected times and in the correct order. In addition, the robot dips forward when the third leg moves ($t = 3.0$ s) in both the simulation and in the physical model. Both models have turned slightly to the right by the end of one gait cycle.



(a) Ant
281 mm × 138 mm × 124 mm

(b) Dragon
210 mm × 162 mm × 88 mm

(c) Mouse
146 mm × 147 mm × 94 mm

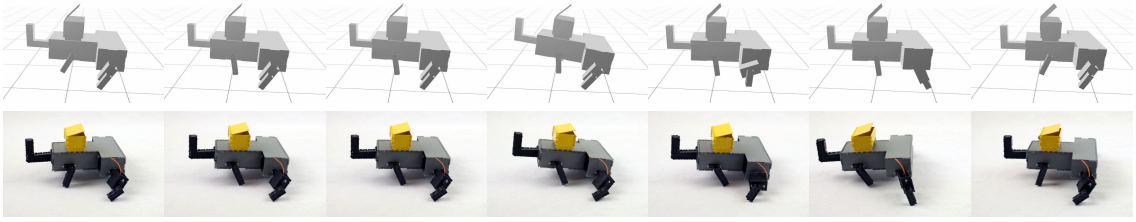


(d) Monkey
207 mm × 250 mm × 112 mm

(e) House
127 mm × 82 mm × 140 mm

(f) Car
200 mm × 117 mm × 98 mm

Figure 6-5: Six robots designed and fabricated using composition. Numbers over the limbs indicate the step sequence.



$t = 0.0$ s $t = 1.0$ s $t = 2.0$ s $t = 3.0$ s $t = 4.0$ s $t = 5.0$ s $t = 6.0$ s

Figure 6-6: Comparison of monkey gait in simulation and in physical robot. The robot motion matches those from the simulations.

To test the degree of similarity, we tracked the movement of each of the robots in a VICON motion capture system. The measured trajectories of the robot's geometric center over 10 gait cycles for 5 trial runs are shown in Figure 6-7, along with the circle of best fit. We calculated the speed, mean curvature, and turning angle for each of these trajectories. The mean metric values for each robot are given in Table 6.1. The car is the fastest robot, with a speed of 357.02 mm/s, since it does not require any step phase, while the monkey is the slowest of the robots, moving at a speed of 6.99 mm/s. Most of the measured metrics match well with the simulated values.

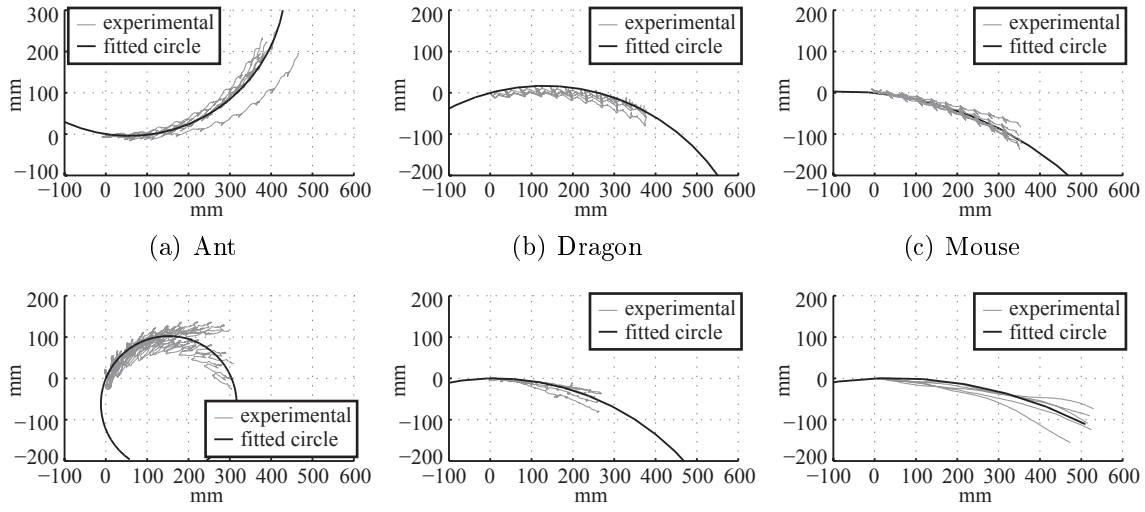
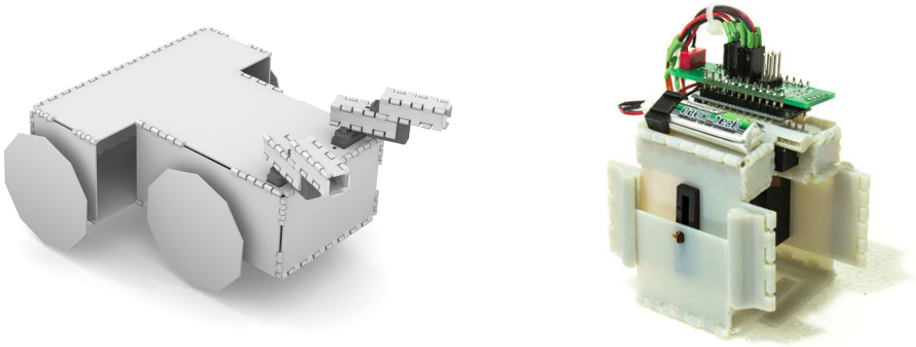


Figure 6-7: Trajectories measured for fabricated robots (gray) overlaid with circle of best fit (black). The robots start at the point (0 mm, 0 mm).

Table 6.1: Comparison of measured vs. simulated metric values

	Speed (mm/s)		**Curvature (1/m)**		**Turning Angle (rad)**	
	measured	simulated	measured	simulated	measured	simulated
Ant	20.60	27.63	2.63	0.02	0.129	0.001
Dragon	14.39	14.31	1.95	1.79	−0.073	−0.047
Mouse	10.63	12.40	1.19	3.97	−0.045	−0.115
Monkey	6.99	9.17	6.05	4.43	−0.182	−0.106
House	14.56	14.01	1.60	0.21	−0.042	−0.004
Car	357.02	419.40	0.97	0.00	−0.051	0.000

All the experimental speed values are within 26% of the expected value, and turning angles are within 0.13 radians (7.35°) of the expected turning angle. Larger errors in speed were correlated with larger errors in turning angle. During experiments, those robots experienced greater amounts of slip because their limbs were not moving completely synchronously and so dragged on the floor. Alignment errors that occurred during assembly also created some differences between the fabricated and simulated designs and accounted for much of the error.



(a) Gripper (b) Biped

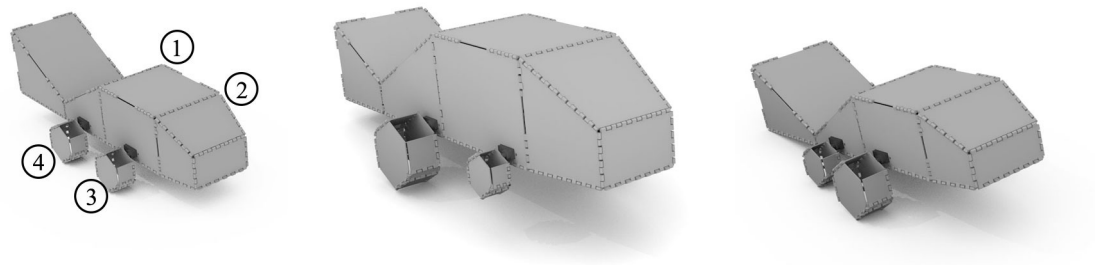
Figure 6-8: Other robot designs achievable using our design system. (a) Robot with gripper composed by attaching 2-link legs to the top of the robot body. (b) Biped robot that uses prismatic joints instead of rotational joints on the legs.

6.4.2 Extensions Beyond the Database

Although our database was designed for the purpose of creating ground robots, many of the modules can also be used for other tasks. For example, legs and fingers on robots are essentially equivalent in all but intended functionality. Our system is flexible enough to accommodate manipulation tasks by users connecting multi-link legs to the tops or sides of robot bodies (ref. Figure 6-8(a)). In addition, because of the joint library, other types of locomotion can be explored as well. Figure 6-8(b) shows an example of a walking robot that uses prismatic joints and linear servomotors instead of the rotational joints and servomotors used on the other robot examples. This robot was programmed manually, but its joint controllers follow the same step and reset phase structure.

6.4.3 Optimization

The design parameterization and performance metrics allow designs to be automatically optimized for particular metric improvements. We use NLopt's COBLYA implementation [83, 139] to search for parameter changes that achieve a local optimum of a metric. Different metrics produce different results. We optimized the geometry of the four-legged fish design in Figure 6-9. The model has 460 parameters. It moves



(a) Original
$V = 25.59$ mm/s
$W = 0.46$ rad

(b) Maximize speed
$V = 52.28$ mm/s
$W = 0.75$ rad

(c) Minimize wobbliness
$V = 15.25$ mm/s
$W = 0.06$ rad

Figure 6-9: Optimization results for a four-legged fish robot when maximizing speed or minimizing wobbliness. Numbers indicate the step sequence. Maximizing speed increases the overall size of the robot. Minimizing wobbliness makes the robot shorter.

one leg at a time in the order indicated by the numbers to walk forward. When the design is optimized for speed, the overall size of the robot increases and two of the legs are lengthened (Figure 6-9(b)), yielding a robot that is just over twice as fast. For physical validity, legs on the same side of the robot must be far enough apart that they will not collide during walking. Since robot speed is most affected by the maximum leg length, having one long leg and one shorter leg on each side yields the highest speed. The same model can also be optimized for minimum wobbliness, in which case the model becomes flatter and the back legs become shorter than the front ones to account for the heavy tail (Figure 6-9(c)). In both cases, the optimization took approximately 2 min.

6.4.4 Fabrication Comparison

Finally, we compare our print-and-fold method to directly 3-D printing the robot bodies. Table 6.2 compares the printing time and material usage for each of the six robots in Figure 6-5 for each process. Our print-and-fold technique yields a 73.2% reduction in 3-D printing time and a 69.9% reduction in material usage on average over all the robot geometries. The difference in material usage is due to the support material that is necessary to print the robots in their 3-D form. In addition to the fabrication efficiency, printing fold patterns enables fabrication of closed shapes that

Table 6.2: Fabrication time and material usage for 3-D printing (3-D) vs. print-and-fold (2-D)

	Material Usage		**Print Time**		**Assembly**
	3-D	2-D	3-D	2-D	
Ant	2176 g	614 g	24.4 hr.	5.9 hr.	55 min.
Dragon	1795 g	517 g	24.2 hr.	6.9 hr.	45 min.
Mouse	1154 g	358 g	21.2 hr.	3.4 hr.	35 min.
Monkey	2779 g	499 g	33.7 hr.	5.2 hr.	45 min.
House	841 g	342 g	10.8 hr.	4.4 hr.	40 min.
Car	1955 g	661 g	18.1 hr.	6.5 hr.	90 min.

are empty on the inside. Most of our printed models would have been impossible to assemble as 3-D models since there is no way to remove the support material from inside the body and to add the electronics.

6.5 Summary

We have described a method for designing ground robots and demonstrated it by designing and fabricating robots with a variety of geometries and gaits. We have presented joint controller modules that can be composed into gaits and trajectories, and we have demonstrated how such a composition approach can be used to automatically generate full fabrication plans that include the software, electronics, and mechanical body of the robot. Further, we have outlined a variety of metrics to evaluate composed designs, and we have shown that parameterizing modules in the database allows us to automatically optimize parameters for those metrics.

Future work includes expanding these techniques to address real world robotic applications. In order for a robot to interact with a physical environment, it would be useful to also be able to estimate metrics such as robustness, agility, and battery life. The database and grammar can be expanded to address a larger variety of tasks and environments. We have already shown how simple changes to our grammar allow us to incorporate other modes of locomotion and possibly manipulation tasks. However, the simulation and metrics described here target the geometry and static stability of the robot design. As a result, the dynamics or required actuation of the

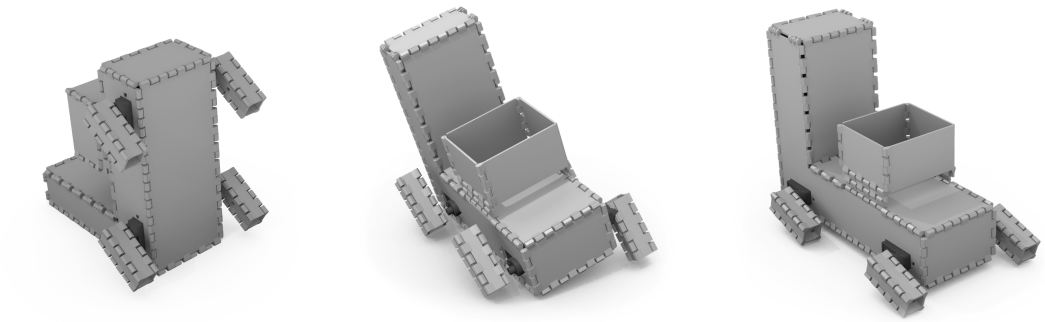
robot is not considered. Generating the electronics plan is therefore straightforward: all joints are assigned the same servomotor. We would like to incorporate information about material properties, load requirements, and other practical considerations to enhance design optimization.

Chapter 7

Interactive Design

Designers who want to create a robot must simultaneously consider multiple aspects of the robot design. In previous chapters, we have shown how design choices can be simplified by using a design-by-composition approach. However, the exact parameter values for the modules that are composed can have a large effect on the robot's performance, even parameters from different aspects of the design. Take for example the robot in Figure 7-1. The designer in this case had the goal of building a fast robot. With the initial design (Figure 7-1(a)), this robot attempts to propel itself forward by moving all legs simultaneously. Because of the tall back section, it topples backwards and then scoots forward slowly (as evidenced by the different wobbliness values W before and during steady state). Changing the gait so that the left and right pairs of legs step sequentially speeds up the robot by preventing it from toppling (Figure 7-1(b)). On the other hand, changing the geometry slightly by increasing the width of the back portion allows the robot to walk with the original gait without falling over (Figure 7-1(c)) and results in a robot that is almost 50% faster.

It is not always obvious what design changes will lead to the better performance. It is possible to use automatic optimization approaches to determine the best parameters for particular metrics (ref. Section 6.4.3), but often what improves one metric will worsen another. When designers would like to satisfy multiple objectives, they require the ability to explore these trade offs in an intuitive manner. Therefore, in this chapter, we present an interactive tool that allows designers to explore how ge-



(a) Original
$V = 34.66$ mm/s
$W = 0.12$ rad
$W_0 = 2.37$ rad

(b) Modified gait
$V = 66.62$ mm/s
$W = 1.37$ rad
$W_0 = 1.37$ rad

(c) Modified geometry
$V = 93.47$ mm/s
$W = 0.00$ rad
$W_0 = 0.00$ rad

Figure 7-1: A robot design that topples while walking (a). Its gait can be modified so that it only wobbles slightly (b), but changing the geometry (c) allows the robot to move faster and more steadily. Reported V and W values are for steady state behavior. W_0 is the robot's average wobble before t_{i_0}.

ometry and motion changes affect robot performance, enabling them to make relevant design choices. The tool uses our previously described design-by-composition framework, as well as our geometry and gait modules and evaluation metrics. We test the tool through a small user study and demonstrate that users with no previous design experience are able to use the tool to create functional robots after about 20 min. of training.[1]

7.1 User Interface and Workflow Overview

Our tool contains methods for geometry and gait design, as well simulations for evaluating the design. Figure 7-2 shows a system diagram for our tool. Users interact with the tool through a graphical user interface (Figure 7-3) in which they can visualize the design they create and receive real time feedback as to how changes to the design affect the robot's real-world performance. In the typical workflow, users first specify the robot's geometry by dragging modules into the workspace from the database shown in the left panel and connecting them together. As they compose the design, the tool automatically adds constraints and determines valid parameter values. Once

[1]Parts of this chapter were published in [153].

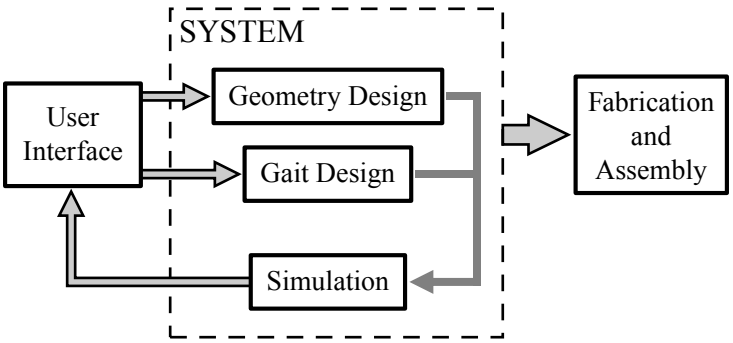


Figure 7-2: Interactive design tool system diagram. Users interact with geometry and gait design tools. The designs are simulated to provide feedback to the user. The user may iterate over the design before fabricating and assembling the robot.

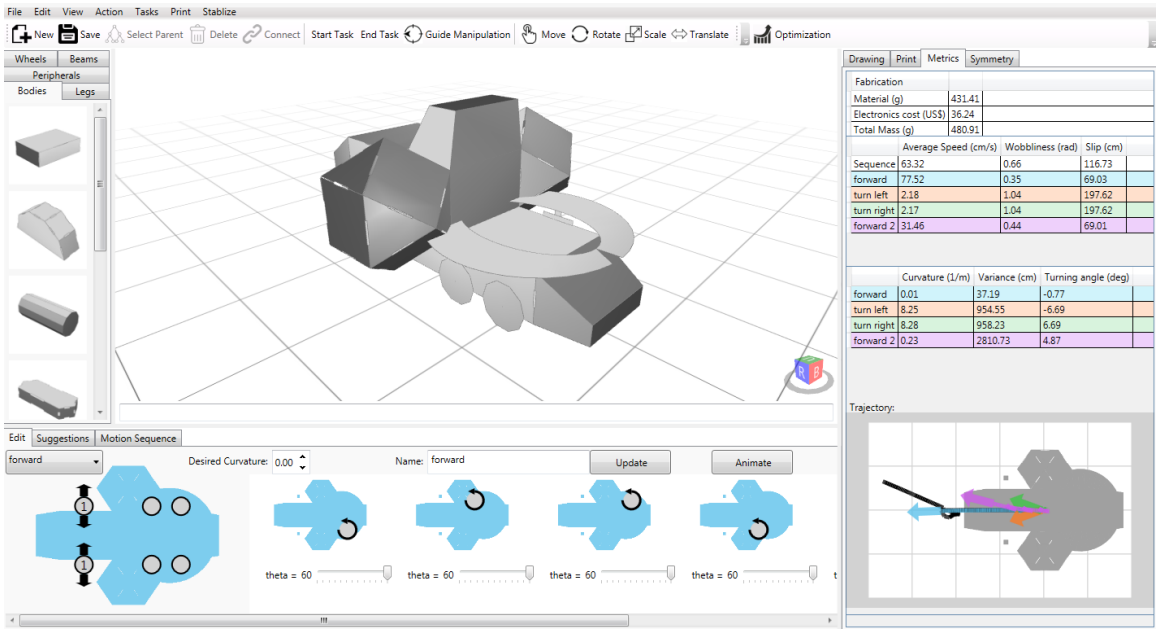


Figure 7-3: User interface. The database of geometry modules is displayed on the left and the gait design tool is on the bottom. Users design robots by dragging modules into the center canvas and editing them. Performance metrics for the design are shown on the right.

the users are satisfied with the geometry, they can design a variety of gaits using a gait design tab at the bottom of the screen. The gaits they design and their corresponding metrics appear in the panel to the right. Once users have a general design, optimization tools guide users to realize their desired robot. Users can explore the geometry and gait design parameters and track the metric changes. The tool can also provide guidance on which parameters have the greatest effect on particular metrics.

The tool keeps track of the geometry and motion in order to output a full fabrication plan once the user is ready to build the design.

7.1.1 Geometry Composition

Our tool allows users to create new designs by composing modules from the database and manipulating their shape parameters.

Geometry Modules

Our database of rigid body modules is shown in Figure 7-4. The database contains a total of 45 modules, including 12 bodies, 23 limbs, and 10 peripherals. All the modules in the database are parameterized to allow structure-preserving variations, and they were created using a system similar to that in [113]. Each module contains a set of shape parameters $\mathbf{q}$ and a corresponding feasible set Q, which together describe the viable module geometries (see Figure 7-5). Each vertex on the geometry is stored as a linear function of the shape parameters $\mathbf{q}$. The module representation includes information about both the 3-D geometry and the 2-D fold pattern. It also includes connection information between the edges where folds and joints are located in the

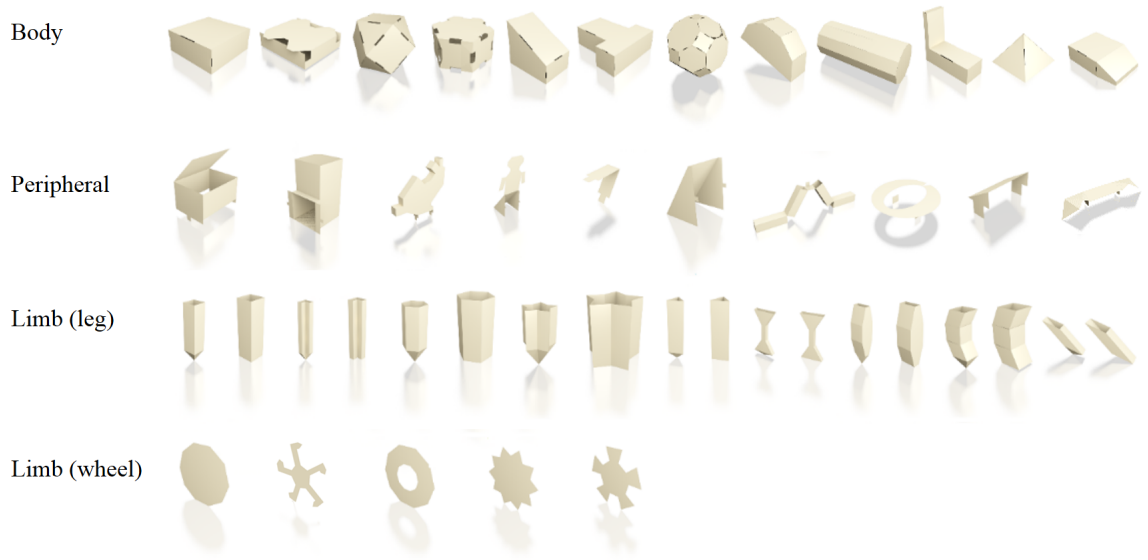


Figure 7-4: Geometry modules by category. Our database contains 45 modules: 12 bodies, 23 limbs, and 10 peripherals.

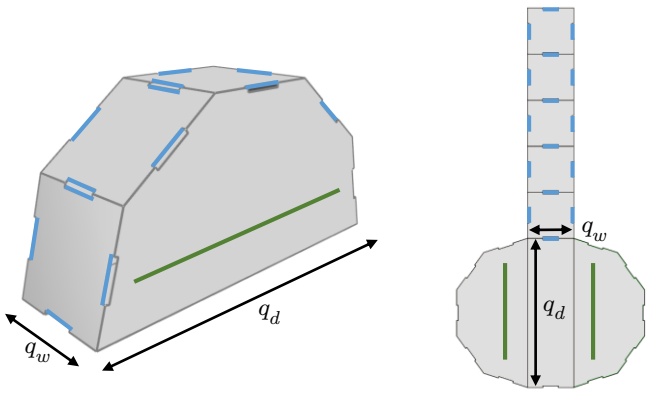


Figure 7-5: One of the geometry modules in the database, with both its 3-D shape and 2-D fold pattern. The module has parameter values of diameter q_d and width q_w. The green line indicates a functional patch, and the blue lines are static patches.

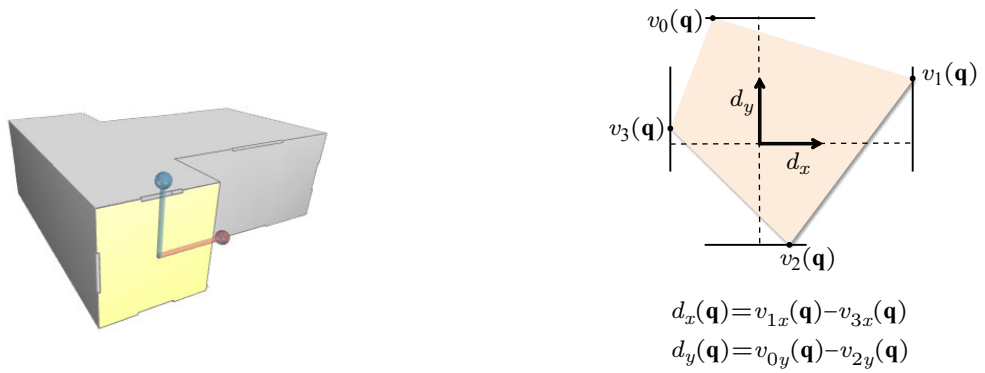
design. The 3-D and 2-D representations are coupled so that they are simultaneously updated as parameters are changed.

For more intuitive composition, modules are annotated with connecting patches that indicate where they are allowed to be connected together. Patch representations are also a function of the shape parameters $\mathbf{q}$. We define two types of patches: static patches for rigid attachments and functional patches for articulated connections. Patches can be points or lines.

Geometry Manipulation

Modules can be modified using the parametric representation. Allowable manipulations include translation, rotation, and dimension scaling. To manipulate a module, users click on the rotate, translate, or scale buttons at the top of the screen. When they next click on a face on the design, orthogonal control axes appear to indicate the possible manipulations (Figure 7-6(a)). Users click and drag on the axes to change the design. The current state and magnitude of the change are displayed to the user in the top left corner of the screen.

In order to maintain a linear representation for the geometric modules, rotation is restricted to a global rotation of the model. When users rotate a part, the linear functions that represent the vertex locations and connections are all updated in the part's representation.



(a) UI control axes

(b) Relevant face dimensions

Figure 7-6: Dimension scaling in the UI. (a) When users click on a face, orthogonal control axes appear that they can use to manipulate the part. (b) The dimension of the part in each direction is written as a function of $\mathbf{q}$ by using the parametric representation of the extreme vertices. The distance $d(\mathbf{q})$ between these vertices is used to update the parameter values.

Translation and scaling correspond to an optimization of the parameter values. When users select a face and drag one of the control axes, the tool solves an optimization problem to perform the indicated change. For scaling, the dimension in the selected direction is expressed as a function $d(\mathbf{q})$ of the design parameters $\mathbf{q}$, which is the distance between the extreme vertices in this direction (see Figure 7-6(b)). Resizing involves finding a feasible solution, $\mathbf{q} \in Q$, such that $\|d(\mathbf{q}) - (d(\mathbf{q}_{current}) + K)\|_2$ is minimized, where $\mathbf{q}_{current}$ are the parameter values at the current configuration and K is the resizing amount defined by the user's dragging. Since there might be many feasible solutions to this problem, the system finds the one that that keeps the design as close as possible to its original state. We express the distance to the current configuration as $\|D(\mathbf{q}) - D(\mathbf{q}_{current})\|_2$, where D is a matrix created by stacking the directional dimensions of all faces and represents the current dimensions of the design. The closest design satisfying the user's input can be found by solving for

$$\begin{aligned} \mathbf{q}^* := \operatorname{argmin} & \|d(\mathbf{q}) - (d(\mathbf{q}_{current}) + K)\|_2 + \\ & \alpha\|D(\mathbf{q}) - D(\mathbf{q}_{current})\|_2 \quad s.t \;\; \mathbf{q} \in \mathcal{Q} \end{aligned} \tag{7.1}$$

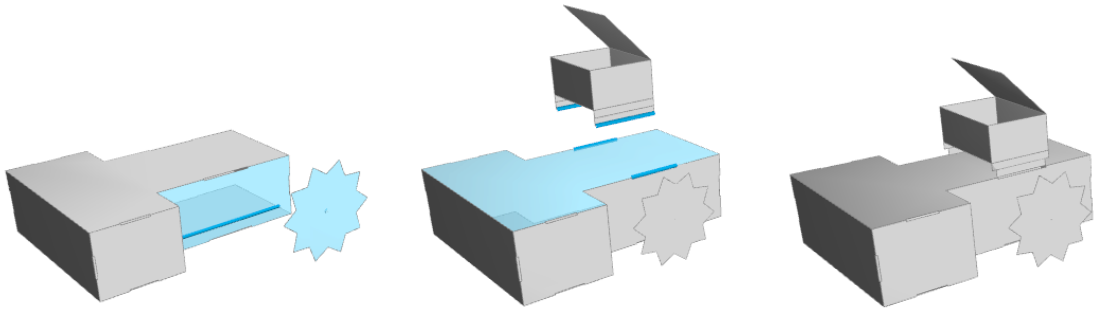


Figure 7-7: As the user drags in modules, the UI highlights the patch pairs that will be connected. Limbs are attached to other parts along functional patches. All other modules attach via static patches.

where α is a weighting parameter. Translation of parts is performed in an analogous manner.

Geometry Composition

As users drag new modules onto the screen, the tool searches for the closest pair of either static or functional patches and proposes to connect them by highlighting them in real time (ref. Figure 7-7). When the users are satisfied with a matching pair, they finalize the connection. This operation first rotates the added module so that the patches are properly aligned, then defines constraints on the parametric representation that snap the modules into place and adds connection information.

The information added depends on the patch type. For static patches between body parts or peripherals, constraints ensure that the pairs of patch edges remain coincident. The tool also adds folds to connect the 2-D fold pattern. Because folds are snappable hinges, if two fold patterns cannot be directly connected together without intersecting, they are printed as separate components. For functional patches connecting limbs to other parts, the constraints enforce that patch regions intersect. The fold patterns are not composed. Joint information indicating that a servomotor lies between these two modules is also added. Joint types are assigned based on the grammar described in Section 6.1. The connections define a kinematic chain that is used for gait design and simulation.

Constraints on point and edge locations can be defined as linear constraints on the shape parameters $\mathbf{q}$. When connecting a limb to a body, a new parameter $\mathbf{q}_c$ indicating the position of the leg along the side of the body is added. Linear constraints are then defined to enforce that the point patch on the limb is attached at this location. In addition, when multiple limbs are attached to the same functional patch, inequality constraints on $\mathbf{q}_c$ are added to enforce that limbs do not collide during locomotion. We use a conservative approximation that ensures limbs will not collide in any gait.

Once constraints are defined, the tool finds the valid parameters that keep the design as close as possible to the current configuration by solving for

$$\mathbf{q}^* := \text{argmin} \, \|D(\mathbf{q}) - D(\mathbf{q}_{current})\|_2 \quad s.t \;\; \mathbf{q} \in \mathcal{Q} \tag{7.2}$$

The solution of this optimization snaps the modules into place.

Since composed designs preserve the parametric representation, users can continue to manipulate the design after composition. When one part is manipulated, the different parts of the composed design will update to satisfy the constraints imposed by the composition algorithm.

Finally, our tool also allows users to define additional constraints on composed designs. The user interface exposes a list of symmetry constraints that can be enforced or relaxed at any design stage. These include equalizing the length of all limbs, uniformly spacing all limbs along the sides of the body, equalizing thickness of all legs, and equalizing the lengths of links on multi-linked legs.

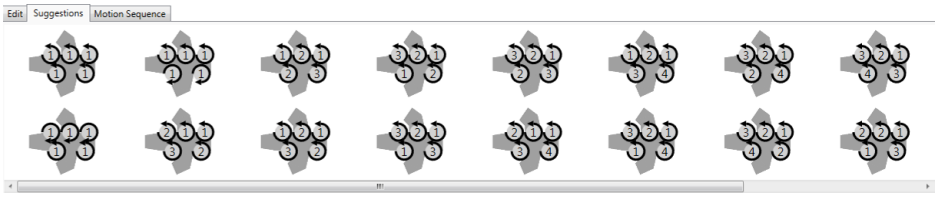
7.1.2 Motion Composition

Similar to the geometry design process, gait design is done by manipulating parameters of gait modules and composing them into a gait sequence.

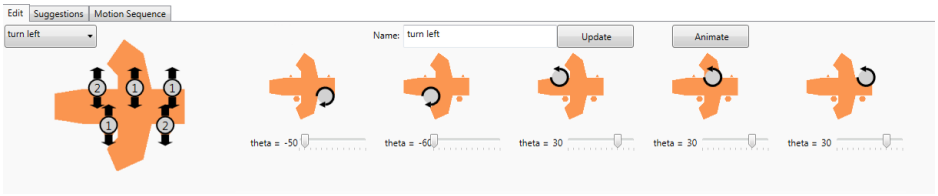
Gait Modules

Our gait modules are as described in Section 6.1. Each joint is assigned a controller based on its joint type.

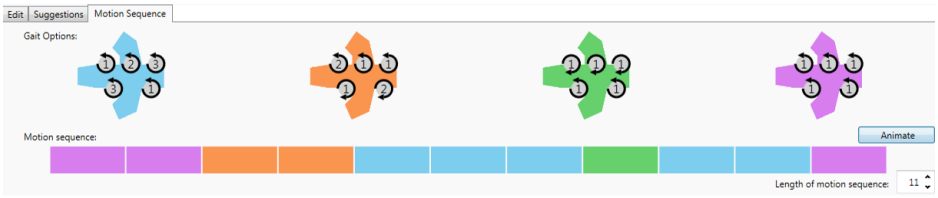
To simplify gait design, the tool provides users with a list of gait suggestions using a heuristic based on the stability and speed for a given topology. Stability is approximated by considering the distance between the projection of the robot's center of mass on the ground plane and its support polygon at its initial configuration, and speed is approximated by counting the number of steps N_g. The tool performs a combinatorial search of all step sequences that keep at least 3 limbs on the ground at all times and suggests step sequences that maximize stability, then speed. These suggestions are presented to the user as an ordered list (Figure 7-8(a)). The numbers indicate the step sequence and arrows show the rotation direction.



(a) Gait suggestions



(b) Gait manipulation



(c) Gait composition

Figure 7-8: Gait design interface. Users can (a) choose among gait suggestions, (b) edit the gait parameters, and (c) compose gaits into a gait sequence by dragging them into a timeline.

The tool also suggests three standard gaits in which all limbs step simultaneously ($N_g = 1$). The three variations consist of all positive values for θ_i's (forward gait), negative values for θ_i's along the left side of the body and positive values for θ_i's on the right (rotation to the left), and positive values for θ_i's on the left side and negative values for θ_i's on the right (rotation to the right). These suggestions are placed first in the list.

Gait Manipulation

After choosing a suggested gait, users can modify the gait in an Edit panel. Our user interface exposes the parameters of the gait, allowing users to vary both step ordering and the θ_i angle values (see Figure 7-8(b)). The panel shows an overhead view of the robot design and indicates the locations of the attached limbs. The limbs are numbered according to the step sequence. Users can change the step sequence by clicking on up and down arrows next to the displayed numbers. The tool ensures that there are no empty steps in the gait. Users can also use the slider bars to change the θ_i values for particular legs up to a magnitude of $\frac{\pi}{3}$ radians (60°). An Animate button on the tab allows users to visualize the gait after editing it.

Gait Composition

After designing several gaits, users can compose a gait sequence by dragging gaits onto a timeline (Figure 7-8(c)). Colored boxes on the timeline indicate the ordering of gaits in the sequence.

7.1.3 Feedback and Guidance

Our tool computes the performance metrics described in Section 6.2 at interactive rates and exposes them to the user as feedback during the design process. A tab on the right side of the UI displays each gait design's metric values, colored according to gait. It also shows an overhead view of the trajectory of the robot's center of mass for each gait. An arrow at the end of the trajectory indicates the robot's final heading.

As the user manipulates the design and composes parts, the metrics and trajectories update automatically to reflect the changes. Users can also visualize an animation of any of the designed gaits in order to have a more comprehensive understanding of the robot's motion.

In addition to these performance metrics, the tool computes fabrication metrics related to the robot geometry and kinematics. These include:

1. Fabrication Cost: The fabrication cost is computed as the mass in grams of nonsupport material required to print the robot body.

2. Electronics Cost: The cost of electronics is the combined cost of the servomotors, the microcontroller, and the battery.

3. Total Mass: The total mass of the robot in grams is calculated as the combined mass of the printed body and the electronics.

Our tool also provides guidance to the user on how to manipulate design dimensions to optimize these metrics. To activate this feature, users select a metric value to improve (i.e., increasing speed, decreasing wobbliness, decreasing slip, etc.) and turn guidance on. When they next choose a face on the design to scale or translate, the tool displays arrows on the control axes indicating which direction to change the dimensions to improve that value. The system uses finite differences to determine how much the metric improves for each control axis and draws an arrow if the improvement is above a given threshold. Figure 7-9 shows an example of these arrows

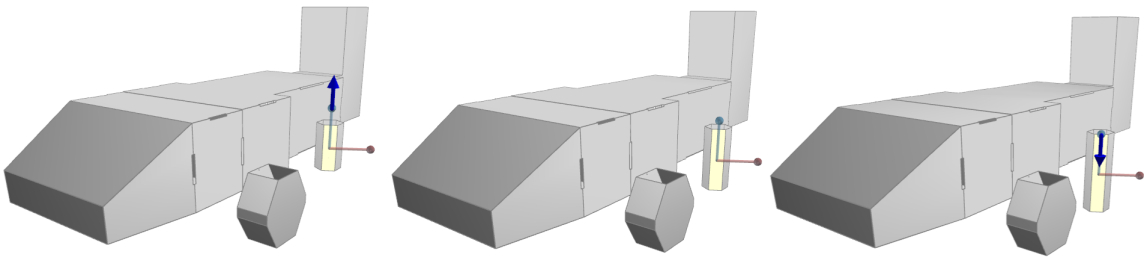


Figure 7-9: The tool provides guidance arrows for users who want to make local geometry optimizations. The up and down arrows indicate that the user should lengthen and shorten the leg respectively. No arrow indicates that the part dimension is already at a local optimum.

for reducing wobbliness on a four-legged robot. The size of an arrow indicates the size of the change in metric value and helps users to decide which geometry changes will have the greatest impact. The guidance can be performed for metrics on individual designed gaits or for the composed gait sequence, and it helps the user to find optimal part dimensions for that metric.

7.2 User Study

To test the usability of the tool and the effectiveness of the design-by-composition approach, we introduced our system to 8 users with no previous experience in robot design. The users were all engineering graduate students (3 female, 5 male) between the ages of 22 and 31. Four of the users had previous experience with CAD and modeling tools (SolidWorks, Blender, Maya, etc.). The users were given 20 min. of training in the tool's features and were then asked to perform 3 tasks designed to evaluate the expressiveness of the geometry and gait design frameworks. The users were asked to fill out questionnaires on their experiences at the end of each task. We present a summary of the results here. Full questionnaire responses can be found in Appendix B.

7.2.1 Geometry Design

In the first task, users were given 10 min. to interact with the geometry composition tool. They were asked to create a visually interesting car for a parade. Figure 7-10 shows the designs that the users created. All of the designs were functional vehicles that rolled forward without toppling. The results demonstrate a wide range of geometries. Interestingly, when body geometries that users wanted were not available as single modules in the database, they were able to create new shapes by composing and rescaling individual bodies together.

In the post-task questionnaire, users generally expressed satisfaction with the expressiveness of the system. An enthusiastic user designed an additional 18 robots with different levels of complexity (Figure 7-11). Each design took 3 to 25 min. to

create. Almost all of the users agreed that the scale and connect functionalities were the most useful in the system. Several users stated that the visualization of the proposed patch connections while the users were manipulating modules was helpful but also expressed a desire to be able to directly control how modules were connected. This could be implemented by displaying all connecting patches on the modules in the workspace and asking users to select the ones they want to connect. Some users were also disappointed with the small size of the database used in the study, and a few suggested that undo functionality would be very useful. These additions could be incorporated with minor changes to the tool.

7.2.2 Combined Geometry and Gait Design

In the second task, users were given a robot design and asked to design a trajectory to navigate the robot through an obstacle course in the least amount of travel time. The users were given 15 min. to complete the task. Half of the users were allowed both to design gaits and to change the geometry parameters of the design, while the other half were restricted to gait design. The robot geometry provided was similar to the one described in Figure 7-1, which topples during forward gaits where all legs

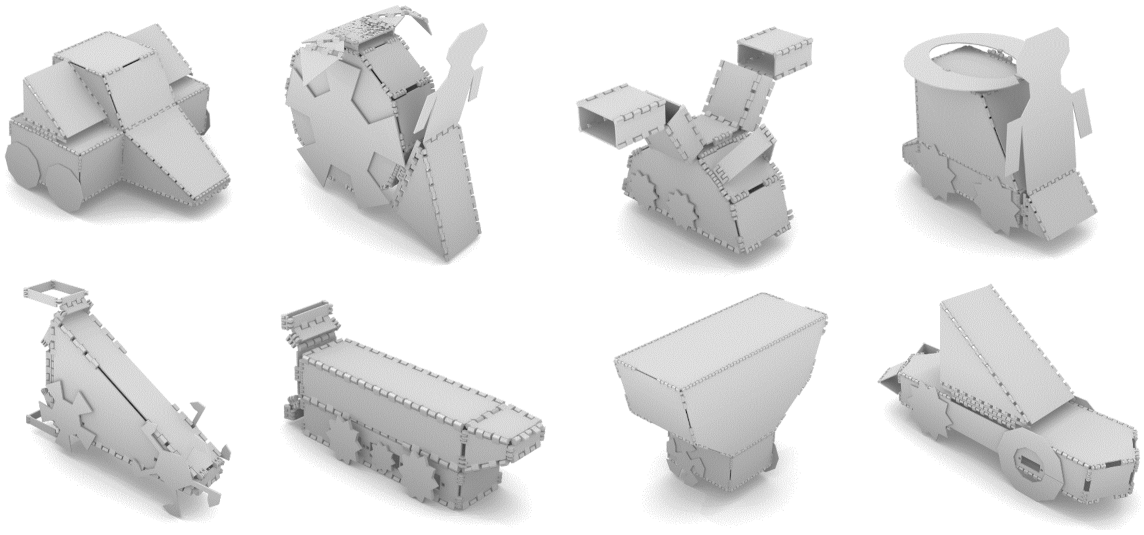


Figure 7-10: Cars designed by 8 different users after a 20 min. training session with the tool. Users were given 10 min. to design their car.

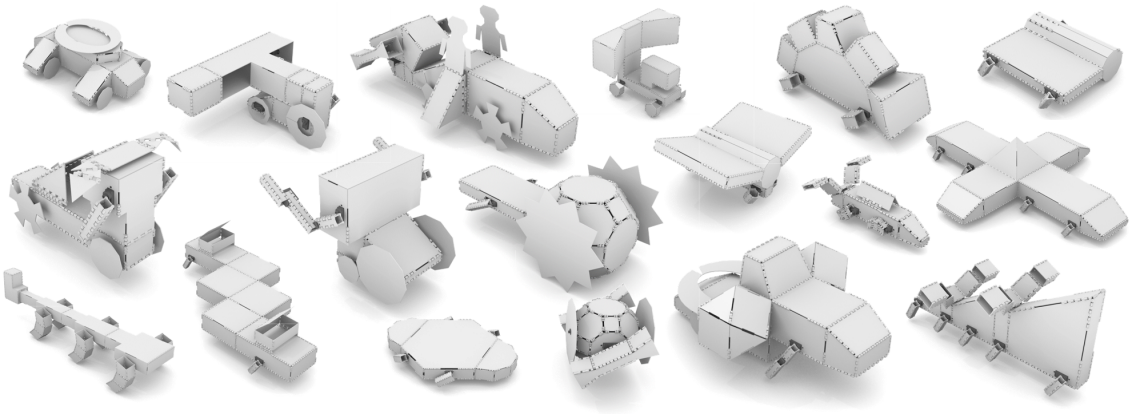


Figure 7-11: Gallery of designs created by one of the users in the user study after a 20 min. training session. Each of the designs took between 3 and 25 min. to design and contains multiple modules from the database.

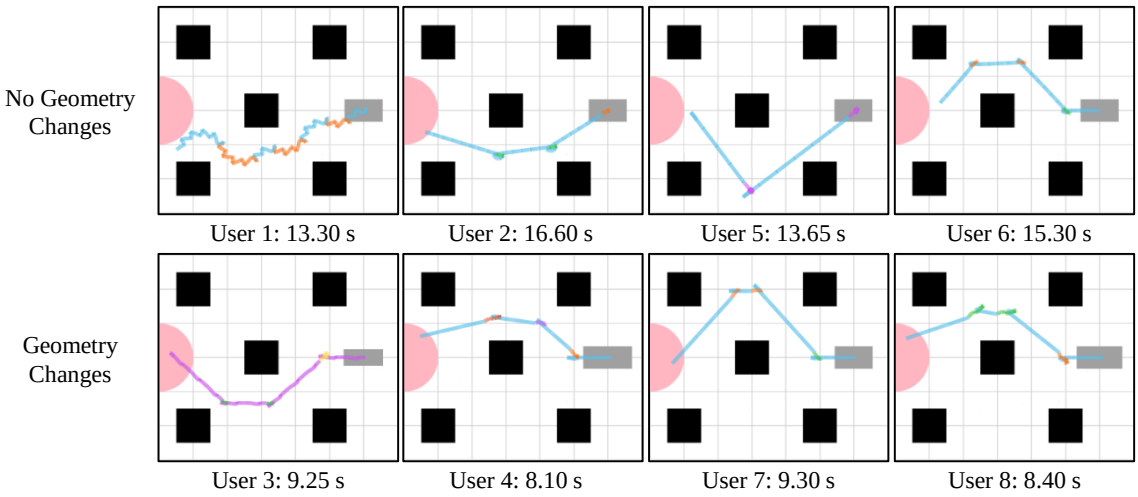


Figure 7-12: Trajectories designed by users during task 2 of the user study. Users who were allowed to make changes to the robot geometry were able to make the robot navigate the course about 40% faster on average.

have equal values for θ_i's. Figure 7-12 shows the trajectories designed by each user and the duration of each gait sequence.

Users who were not allowed to manipulate the geometry were able to explore the gait design tool creatively to reach the goal. Solutions included a combination of right and left rotation gaits (user 1), fixing the back legs and moving forward stably with only the front legs (users 2 and 6), and performing a 180° rotation before moving with a backwards gait towards the goal (user 5).

Users who were not restricted to gait design edited the robot dimensions so that it would be stable during a forward gait with all legs moving simultaneously. The resulting gait sequences were significantly faster (40% time reduction on average) than the ones designed without geometry modification. This result speaks to the effectiveness of concurrent design of multiple aspects of a robot.

To reach these solutions, users made extensive use of the interactive feedback during manipulation of gait parameters and gait sequence composition. Users who could modify the geometry referenced the metrics tab on the user interface when manipulating the shape to find more stable configurations. One user used the guide arrows to minimize the wobbliness metric. The users reported that the most useful features were the animations and the trajectory visualization. Several of them suggested that the ability to pause the animation and view the configuration at particular points in time would have been useful for the task since the designed trajectories were quite long. The users seemed to have no difficulty using the gait design tool to explore the gait parameters, although one user suggested that guidance on the θ_i parameters in a similar vein to the geometry guidance would have been helpful.

7.2.3 Interactive Feedback and Optimization

In the third task, users were presented with two four-legged robot designs with a given step sequence and asked to change the parameters of the design to bring wobbliness below 0.5 radians while maintaining speed above a threshold value (150 mm/s for design A, 190 mm/s for design B, ref. Figure B.4) set prior to the study to equalize task difficulty. The users were given 5 min. for each design, and the design order was randomized. They were asked to modify the robot using the feedback guidance arrows and metrics for one task and without them on the other. Users were able to animate the robot and visually check its performance in both cases.

Table 7.1 shows the results of this task. Most users were able to complete the task with the feedback and guidance arrows, while only half of the users were able to complete the task without the feedback. The two users who completed the task in less time without the feedback reported the highest familiarity with CAD tools and

Table 7.1: Timing data from task 3 of user study where users optimized a design with and without feedback

User ID	Time taken (s)		Which was easier?
	with feedback	without feedback	
1	192	*	with feedback
2	108	214	with feedback
3	*	260	with feedback
4	152	*	with feedback
5	+	*	with feedback
6	110	96	with feedback
7	112	70	without feedback
8	138	*	with feedback

* did not complete the task + system crashed

robot design out of all the users. Both users reported in the post-task questionnaire that they were able to use their intuition to optimize the design. Most users reported that the task was easier when feedback was turned on since it helped them to detect where the greatest gains could be made. However, some users were unhappy with how the guidance slowed down the system. Users 3 and 5, who were both unable to complete the task with the feedback on, experienced significant lags during the experiment, and the system crashed during the test with user 5. These lags were caused by the use of finite differences to compute the magnitude and direction of the guidance arrows. Some users suggested that instead of having the user click faces and check the guidance arrows, the system should compute all the potential guidance arrows and indicate which changes would cause the greatest change in the selected metric. This feature, although potentially helpful, would be even slower than the existing methods. Future work on making manipulation guidance faster is therefore required for the feedback to be useful.

7.3 Summary

We have presented an interactive tool that allows designers to create robots using composition. Users can explore the space of geometries and gaits in their designs in order to efficiently achieve desired performance metrics. The tool incorporates simu-

lation and optimization methods to provide the user with feedback and guidance on how to modify their designs. It also takes care of the implementation details required for fabrication, allowing users to focus on the conceptual high-level design. We have tested the tool in a preliminary user study and shown that novice designers respond well to the user interface, the flexibility given them to manipulate design parameters, and the metrics and manipulation guidance. The system is able to provide sufficient feedback for standard planning tasks.

Our user study provided us with valuable comments for future improvements. Users generally liked the composition tool but expressed a desire for greater variety in the database, as well as more feedback as to how modules would be composed. They were also dissatisfied with the slowness of the system when manipulation guidance was turned on, although they agreed for the most part that the guidance was useful. In addition, although users did not comment on this aspect, our user feedback is restricted to local guidance. The tool is currently unable to accommodate situations where more global changes are needed (e.g., changing the dimensions of multiple parts simultaneously) or where additional parts should be added. It is also unable to determine when a user-designed robot is unable to achieve the desired performance metrics given the parameter constraints. Future work includes enhancing the system with such features.

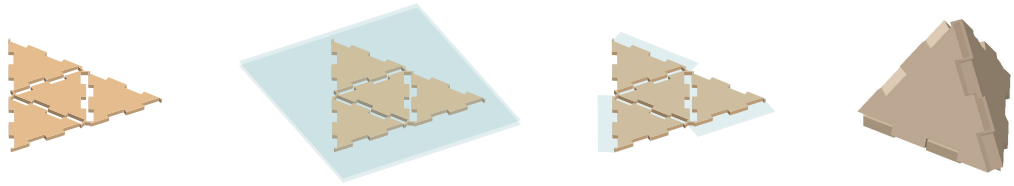
Chapter 8

Extensions to Other Fabrication Methods

The fabrication method we have demonstrated so far uses a combination of 3-D printing, laser cutting, and folding to rapidly fabricate lightweight robots. However, the resulting designs are limited in size to what can fit in the machine and are intended to have thin walls that do not interfere with the fold, so they are limited in the types of tasks that they can accomplish. Our system ensures the validity of a fold pattern and is not specific to a fabrication method, so the same fold pattern can be implemented using any method that can generate the necessary folds and joints for the pattern and that produces a robot that can maintain its shape. In this chapter, we demonstrate how our methods can be extended to fabricating foldable robots at larger scales, and we experimentally verify these methods by fabricating three medium-to-large scale robots.

8.1 Medium Scale Fabrication and Design

Larger robots require bodies made of stiffer material. At the same time, we would like the robot to remain lightweight and quickly fabricable. For medium-scale robots on the order of 20–40 cm cube, we introduce a new fabrication option that involves laser cutting thick materials to produce larger, more rigid robot bodies than those



(a) Cut rigid faces (b) Attach adhesive film (c) Cut adhesive film (d) Fold

Figure 8-1: Multi-layer process for fabricating medium-scale rigid robot bodies. (a) The fold pattern is first cut from a rigid material. (b) Adhesive film is attached to both sides of the rigid material and (c) then cut into the correct shape. (d) Finally, the layered structure is folded into its final 3-D form. Tabs cut from the adhesive film are folded to secure the shape.

in the previous chapters. Rigid faces are layered on top of a flexible film that allows folding similarly to work in thick origami [175, 183, 208]. Compared to traditional origami fold patterns, however, which contain many cycles of connected faces that can help the structure maintain its shape, the fold patterns for foldable robots and mechanisms are often simpler and more treelike. We therefore use interlocking teeth along the edges of joined faces to prevent slip.[1]

8.1.1 Fabrication Process

We fabricate rigid robots by layering a thick, rigid material and a flexible film in the shape of the fold pattern and then folding the structure into its 3-D form. The process is illustrated in Figure 8-1. All steps except the final assembly step use planar fabrication.

The faces of the fold pattern are first cut out of a rigid material using a laser cutter. We choose to use acrylic sheet for its high surface energy and for ease of laser cutting. To account for material thickness, the faces are shrunk and separated by a gap, similarly to [175]. While cutting individual faces, thin strips that bridge the gaps are kept so that the structure remains a single piece and the correct gap width is preserved. These strips are later cut during folding. Interlocking teeth along each

[1]The majority of this section was published in [171].

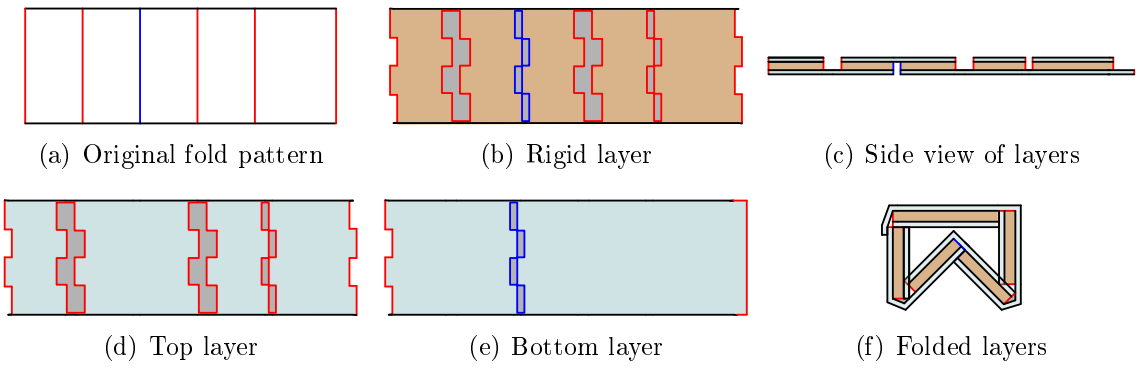
pair of joining edges add structural integrity and keep faces from sliding along the fold line.

Once the rigid material is cut, a thin, adhesive-backed polyester film is layered on the top and bottom. The film maintains the geometry of the rigid material even once the thin strips between faces have been cut. The film is then cut using a laser cutter. For each fold, either the top or bottom film is cut along the toothed edges and removed, depending on the sign of the fold angle, to allow folding. Since the two films are separated by the thickness of the rigid material, they can be cut individually without affecting the integrity of the other side. For nonadjacent joining edges, we add a tab to one of the pair of edges so that the two can be secured together during folding. Finally, the thin strips connecting the rigid faces are also cut and removed.

During assembly, the cutout is folded into its 3-D form. Tabs cut from the adhesive film are wrapped around the corresponding folds to secure the two fold edges together.

8.1.2 Pattern Generation

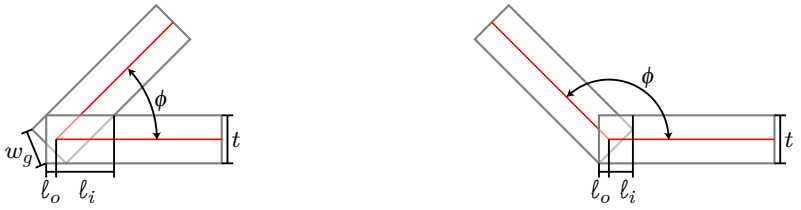
Cut patterns for each of the layers can be automatically generated given a fold pattern. The cut pattern consists of two parts: 1) cuts for the thick rigid layer, and 2) cuts for the thin adhesive layers.



(a) Original fold pattern (b) Rigid layer (c) Side view of layers

(d) Top layer (e) Bottom layer (f) Folded layers

Figure 8-2: Layers for an example fold pattern fabricated using our multi-layered approach. (a) Original fold pattern. Red lines correspond to joining edges with positive angle, and blue lines correspond to negative angle. (b) Cut pattern for rigid layer. (d,e) Resulting cut patterns for top and bottom adhesive-backed layers. (c,f) Side view of the resulting layered structure.



(a) Acute angle ($|\phi| < \pi/2$) (b) Obtuse angle ($|\phi| \geq \pi/2$)

Figure 8-3: Side view of two faces of thickness t that come together at a fold with angle ϕ between them. The midline of the material, indicated in red, shows the original length of the faces. The length ℓ_o is the greatest amount that a face can be lengthened without breaking through the surface of the other face. The length ℓ_i is the length that the face must be shortened to avoid intersection with the other face. The gap width w_g is the amount of space that must be placed between two faces joined at an acute angle once they have been lengthened by ℓ_o.

Rigid Material Three-dimensional bodies with thick walls have faces of different sizes when viewed from one side compared to the other. This creates a challenge for laser cutting the faces of the body since cuts are perpendicular to the faces being cut. As a result, faces cannot always be cut to the same size as they were in the original fold pattern. Figure 8-3 shows the resulting geometry for two faces of thickness t that are joined with an angle $\phi \in [-\pi, \pi]$ between them. Because of the thickness of the material, when the two faces are joined at an angle, there exists either some overlap or a gap between the faces. The red line indicates the slice of the material that has the same dimensions as the original fold pattern. This is equivalent to saying that the material was folded at that slice. The location of the slice can be designed to occur at any position within the thickness of the material, or even outside of the material. We choose for it to be at the midline of the material for symmetry.

Faces joined at an edge must be trimmed so that no intersection occurs during folding. We also add teeth along a joining edge so that the joined faces interlock for greater rigidity. We draw rectangular teeth relative to the original location of the edge of the face and refer to the distances of the outermost and innermost points from the edge location as ℓ_o and ℓ_i respectively. The value of ℓ_o can be calculated as the greatest amount that one of the faces can be lengthened without breaking through the surface of the other face. The value of ℓ_i is the amount that the face must be

shrunk in order to avoid intersection with the other face, once lengthened. These two values vary depending on the angle between the two joined faces as

$$\ell_i = \begin{cases} \frac{t}{2\tan\frac{\phi}{2}} & |\phi| < \pi/2 \\ t\sin(\phi) - \frac{t}{2\tan\frac{\phi}{2}} & |\phi| \geq \pi/2 \end{cases} \tag{8.1}$$

$$\ell_o = \begin{cases} \frac{t}{\sin\phi} - \frac{t}{2\tan\frac{\phi}{2}} & |\phi| < \pi/2 \\ \frac{t}{2\tan\frac{\phi}{2}} & |\phi| \geq \pi/2 \end{cases} \tag{8.2}$$

The cut pattern for the rigid material is drawn by shrinking and shifting the faces and adding teeth to the fold edges. The procedure is as follows:

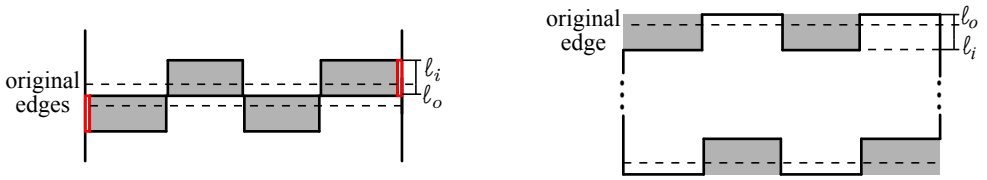
1. **Shrink Faces** For each pair of joined faces, shrink the face by ℓ_i in the direction perpendicular to the joining edge. This results in a gap of width $2\ell_i$ between the two faces.

2. **Shift Faces** When shifting faces, the amount of shift depends on the whether the angle between the faces is acute or obtuse. For obtuse angles (Figure 8-3(b)), the faces must be further separated by a distance of $2\ell_o$. For acute angles (Figure 8-3(a)), the faces must be separated by a distance of $2\ell_o + w_g$, where

$$w_g = 2\sin\frac{\phi}{2}\left(\frac{t}{\tan\frac{\phi}{2}} - \frac{t}{sin\phi}\right) \tag{8.3}$$

   is the distance across the shortened corner caused by cutting the faces so that they do not extend past the angle formed by the outer surfaces.

3. **Add Teeth** For each pair of joining edges, we add alternating rectangular teeth to the two edges. The teeth extend a width $\ell_i + \ell_o$ past the edges of the shrunken faces. The number of teeth is chosen such that each tooth is at most 10 mm long and there are at least three teeth. In addition, teeth are cut 0.3 mm wider than the gap on the opposite edge to account for the beam width of the laser cutter.



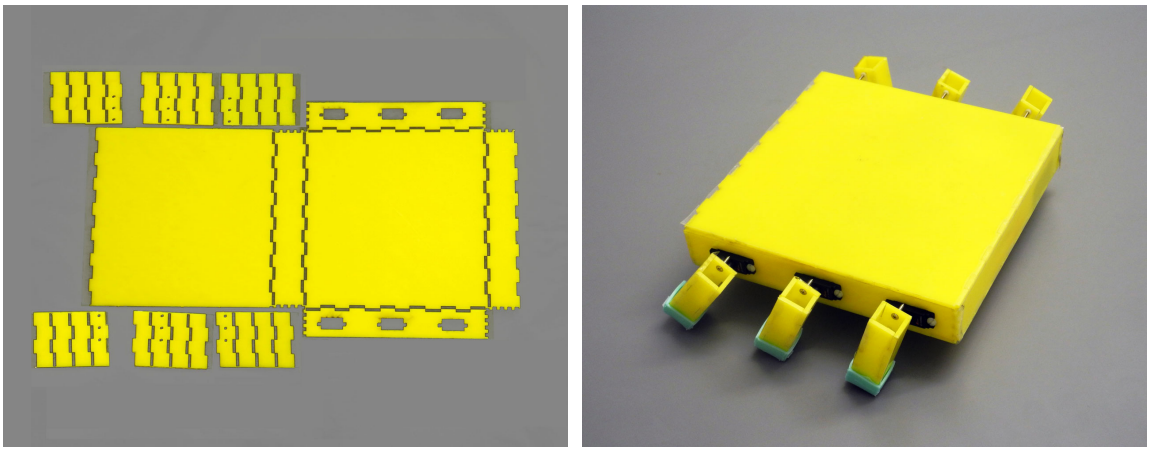
(a) Adjacent joining edges (b) Nonadjacent joining edges

Figure 8-4: Teeth structure for a pair of edges that form a fold. (a) Red lines indicate the pieces that keep the structure connected but are cut prior to assembly.

Finally, in order for the material to remain one piece until assembly, thin strips of material that bridge the gaps between adjacent joining edges are kept. This is done by drawing the teeth for a length 2 mm shorter than the actual edge length. Figure 8-4 shows the resulting teeth structure for both adjacent and nonadjacent joining edges. Red lines on the teeth for adjacent edges show the pieces that are cut off prior to assembly.

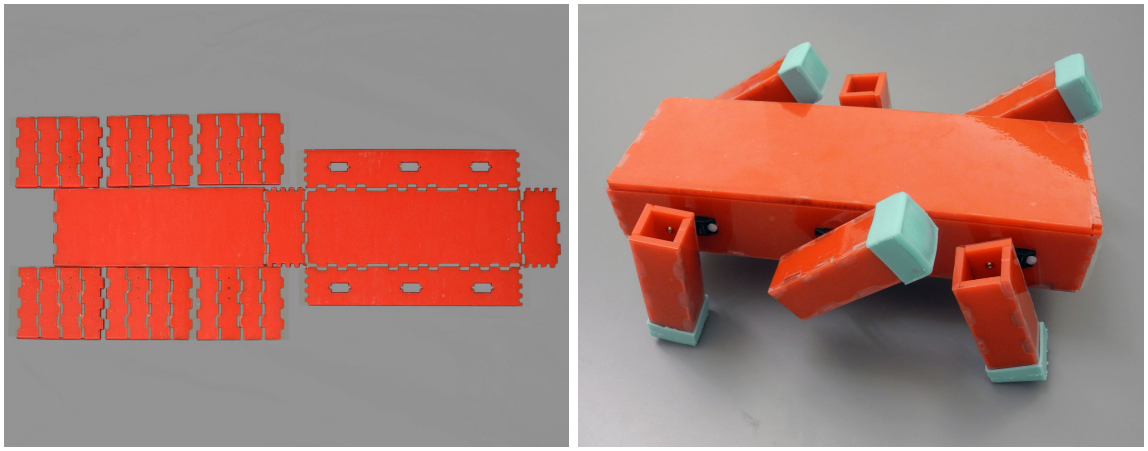
Adhesive Layers Adhesive layers are placed on both the top and the bottom of the rigid layer and are cut away to allow folding. We use the convention that when the angle ϕ between two joined faces is positive, then the top layer is cut away (ref. Figure 8-2(c)); when the angle is negative, then the bottom layer is cut.

To draw the cut pattern of the top layer, we begin with the cut pattern of the rigid material. Then, for each adjacent joining edge, the cut lines corresponding to the teeth are removed if ϕ is negative and remain unmodified otherwise. For each nonadjacent edge, the cut lines corresponding to the teeth are similarly modified only if ϕ is negative. In that case, the cut lines corresponding to the teeth are kept unmodified for one edge, and the other edge is replaced by a rectangular tab. The process is repeated in the same fashion for the bottom layer, except that cut lines corresponding to joining edges with positive angle ϕ are modified instead. Figure 8-2 shows the resulting top and bottom layers for an example fold pattern.



(a) Fold pattern (b) Assembled robot

Figure 8-5: Wide hexapod cut out of 3.18 mm thick acrylic sheet using our multi-layer fabrication process.



(a) Fold pattern (b) Assembled robot

Figure 8-6: Long hexapod cut out of 4.50 mm thick acrylic sheet using our multi-layer fabrication process.

8.1.3 Fabricated Results

We have used this fabrication process to create two hexapods of different dimensions designed using the tool described in Chapter 7. The pattern for the hexapod consists of a single rectangular body and six rectangular beam legs. Although the legs are drawn adjacent to the body, they are not connected via any joining edges. To test the system's ability to accommodate different thicknesses of material, we cut a short and wide hexapod (Figure 8-5) out of 3.18 mm thick acrylic sheet and a tall and long

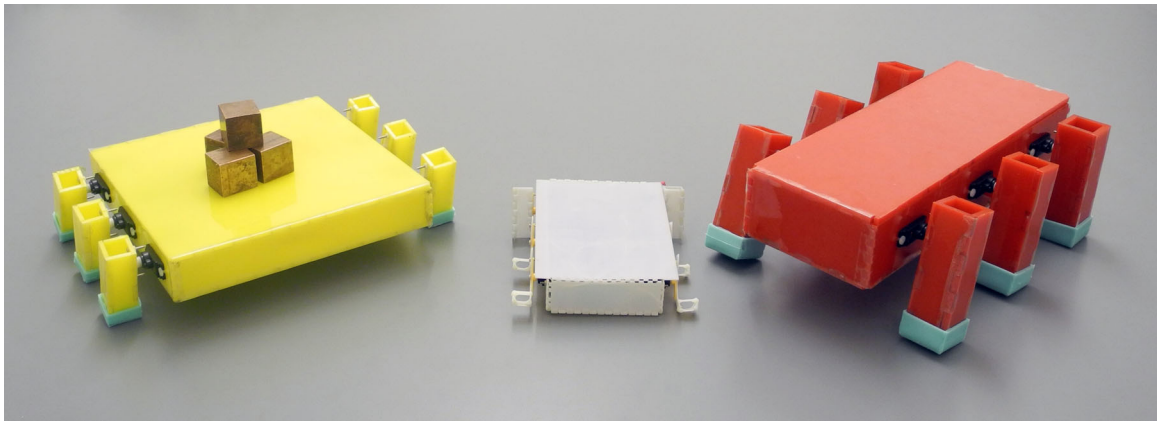


Figure 8-7: Hexapods cut from thick material next to print-and-fold hexapod (middle). The new hexapods are almost twice as large.

Table 8.1: Specifications for medium-scale hexapod robots

	Wide Hexapod	**Long Hexapod**
Thickness	3.18 mm	4.50 mm
Length	192 mm	288 mm
Width	260 mm	190 mm
Height	60 mm	95 mm
Mass	0.672 kg	1.058 kg
Speed	27.7 mm/s	35.9 mm/s
Payload Capacity	2.50 kg	0.76 kg

hexapod out of 4.50 mm thick acrylic sheet (Figure 8-6). We layered both robots with a 0.05 mm thick polyester film backed with acrylic adhesive. The cut patterns for the hexapods both folded into the correct shapes.

Fabrication Because of the heavier bodies, we were unable to use the same servomotors as those used in the 3-D printed robots. However, aside from swapping out the actuators, the electronics plans remain similar. The robots were each actuated using six servomotors with stall torques of 2.7 kg-cm and controlled using an Arduino Uno. They were powered by two 3.7 V, 2600 mAH lithium ion batteries regulated to 6 V to meet the power requirements of the servomotors. Similarly to the servomotors on the smaller robots, these servomotors were modified to allow continuous rotation and provide position feedback from the potentiometer connected to the output shaft.

Table 8.2: Timing per step of fabrication

	Wide Hexapod	**Long Hexapod**
Cut Rigid Layer	10 min.	20 min.
Attach Adhesive Layer	25 min.	50 min.
Cut Adhesive Layer	5 min.	5 min.
Assemble	45 min.	60 min.
Attach Electronics	25 min.	25 min.
Total	1 hr. 50 min.	2 hr. 40 min.
3-D print body	10 hr. 56 min.	14 hr. 26 min.
3-D print faces	9 hr. 28 min.	12 hr 25 min.

Table 8.2 shows the amount of time required to fully fabricate and assemble each robot. Fabrication of the layered structure took approximately 1 hour for each robot, and full assembly, including attaching electronics, took an additional 1.5 hours. The long hexapod, which is the larger and heavier of the two, took more time to fabricate and assemble than the wide hexapod. Since it is longer, the entire fold pattern did not fit on one sheet of acrylic, so the robot was fabricated in two parts that were attached together using the adhesive-backed film. This process doubled the amount of time required to attach the adhesive layer to the rigid layer for the long hexapod compared to the wide hexapod. Rubber feet were placed on each of the legs to prevent slip during the walking gait. The feet were fabricated by 3-D printing a mold, pouring A15 durometer silicone rubber into the mold, and then allowing the rubber to cure. The entire process took about 5 hours.

Compared to 3-D printing the hexapods, laser cutting and folding the robots resulted in substantial time and materials savings. In particular, 3-D printing the long hexapod takes almost 14.5 hours on a Fortus 400mc printer (Table 8.2), and one-quarter of the printed material is support for the hollow body. Even separating the body into individual faces to minimize support material still requires 12.5 hours of printing, not to mention the additional time for assembly. Similarly, printing the wide hexapod takes 11 hours, with one-third of the material being support material, or 9.5 hours for printing just the faces.

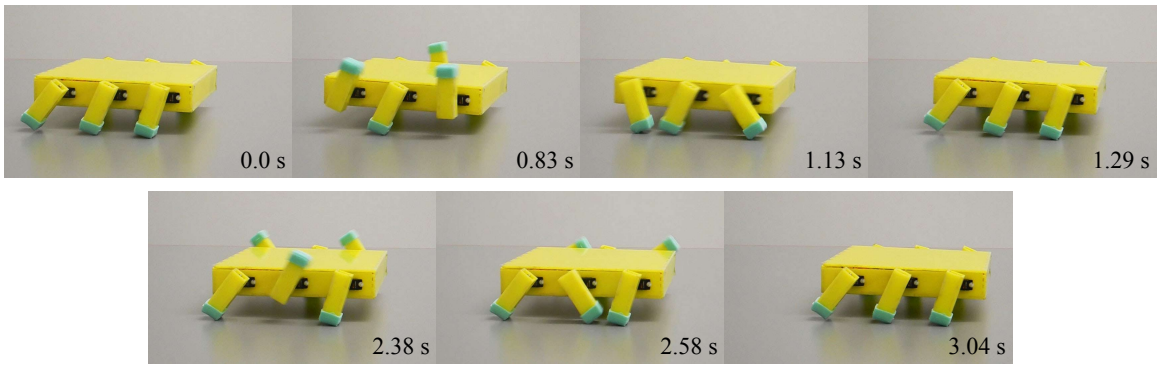


Figure 8-8: One cycle in the walking gait of the wide hexapod. Three legs make one full rotation to shift the robot forward while the other three keep the robot stable. Next the other three legs rotate.

Performance Both hexapods were programmed to follow a tripod gait as suggested by the design system. The resulting walking gait is shown in Figure 8-8. All the legs were initialized to an angle of 0.35 radians (20°) offset from vertical. The legs were then split into two sets of three, with each set containing the front and back legs of one side of the robot and the middle leg of the other. One set of legs rotated one complete revolution in 1.5 s while the other set remained static to maintain the stability of the robot. The sets of legs alternated between rotating and remaining static.

Both robots were able to walk forward stably, although lack of synchrony between the servomotors occasionally caused the robots to shuffle and turn during steps. During experiments, the robots turned at most 0.17 radians (10°) per step in the direction of the side whose middle leg was moving. Table 8.1 describes the size and performance of each robot. Speed was computed by measuring the amount of time required for the robot to walk forward 3 m and averaging the result over 3 trials. Payload capacity was measured by incrementally adding weights on top of the robot until its leg sets were no longer able to complete a full rotation (i.e., the robot could no longer walk forward).

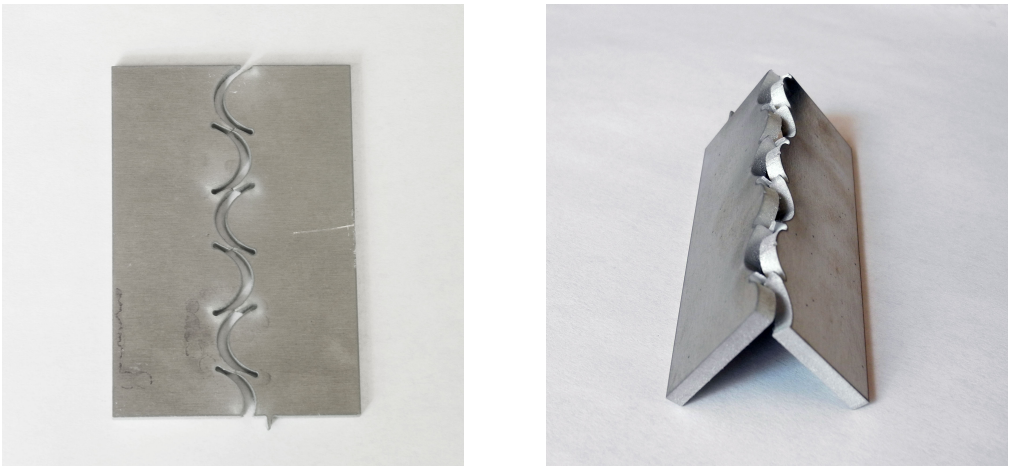
As might be expected, the long hexapod could move faster than the wide hexapod but was able to carry less payload. While the wide hexapod was able to carry more than 3.5 times its own body weight, the long hexapod, which uses the same

servomotors but is taller and heavier, was able to carry only about 0.72 times its own weight. In terms of speed, both hexapods used the same motion sequence, but the long hexapod has longer legs and so moved about 1.30 times as fast as the wide hexapod.

8.2 Large Scale Fabrication and Design

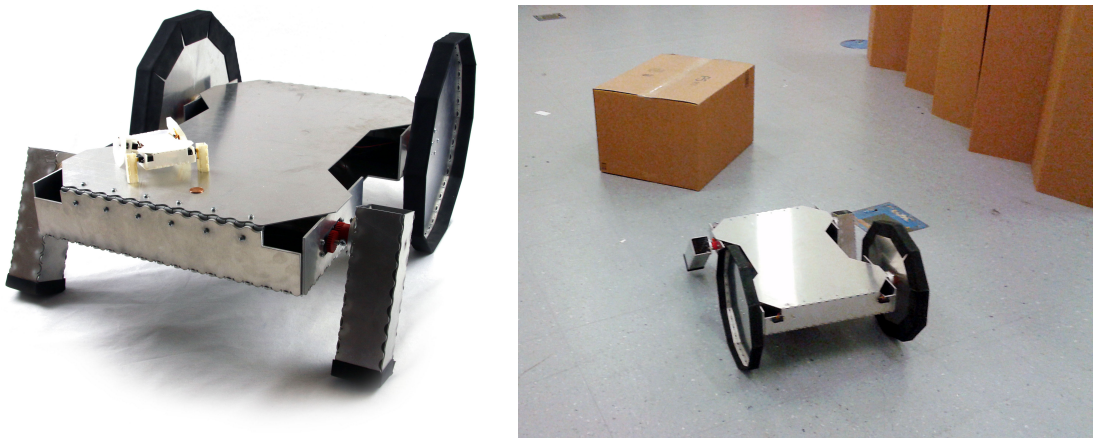
Layers of plastic material work well at the medium scale, but at larger scales and larger loads, they begin to crack and delaminate. To produce robots at the meter scale, we use metal sheets and leverage techniques in sheet metal design. Conventional sheet metal folding requires large machinery specifically set up to cut and fold a particular shape. For customizable one-off robots, we require a more flexible fabrication process. We therefore introduce a fabrication option that involves water jetting metal sheets using a perforation pattern similar to that used for folding robots out of thin films. With this pattern, metal robots can be folded and assembled by hand without external machinery.

The main difficulty with folding metal sheets is in ensuring that folds occur at the correct locations. We use a semicircular pattern centered about the fold line to weaken the material enough for a person to fold it (ref. Figure 8-9). The includes



(a) Cut pattern (b) Folded state

Figure 8-9: Cut pattern for folds on a metal robot



(a) Robot comparison (b) Robot in obstacle course

Figure 8-10: Large metal robot designed in system and folded from aluminum sheet. (a) Robot next to print-and-fold version. (b) Robot navigating an obstacle course.

strategically placed cuts so that the material, rather than bending to a sharp corner and incurring a high stress concentration, twists to produce a larger bending radius. Parameters to the cut pattern include the arc that is cut and the spacing between arcs. Changes to the arc control the fold angle by changing at what point during the fold the two connected faces come into contact, while changes to the spacing between arcs affect the stiffness of the fold and the strength required to fold the pattern.

8.2.1 Fabricated Results

We have used this fabrication approach to create a 627 mm × 602 mm × 368 mm robot designed using the tool in Chapter 7. The robot has two front legs and two back wheels. The legs and wheels are folded out of 1.59 mm thick 3003 aluminum sheet, and the body is folded out of 3.18 mm thick 3003 aluminum sheet. Figure 8-10(a) shows the fabricated result next to a robot of the same design that was fabricated using our print-and-fold method. The fold pattern of the two designs was the same but scaled by a factor of about 6. Both robots walk forward by first rotating one leg, then the other, and then shifting all limbs simultaneously.

Because of its size, we had to make a few changes to the electronics and actuation of the metal robot. First, rather than using an off-the-shelf servomotor, we created a

Table 8.3: Specifications for metal robot and print-and-fold version

	Metal Robot	**Print-and-fold Robot**
Length	627 mm	103 mm
Width	602 mm	103 mm
Height	368 mm	63 mm
Mass	9.300 kg	75 g
Speed	18.91 mm/s	31.2 mm/s
Payload Capacity	14.456 kg	188 g

custom servo using a 12 V Pittman® gearmotor and a magnetic non-contacting rotary position sensor. Similarly to the medium-scale robots, this robot was controlled using an Arduino Uno and similar software to that outputted from the design tool. The robot was powered using a 12 V battery pack.

The fabricated large robot was able to walk forward stably using the designed gait. Rubber sheet was attached to the bottoms of the legs and to the rims of the wheels to prevent the sharp corners of the metal from snagging on or scratching the floor. Table 8.3 shows a comparison of the large robot to its print-and-fold counterpart. The metal robot is able to carry a payload in excess of 14 kg, which is approximately 1.5 times its own body weight. It walks at about half the speed of the print-and-fold robot due to the gearing in the motors used. In both robots, the battery and motors accounted for approximately the same proportion of the total robot mass (28% of metal robot mass, 29% of print-and-fold robot mass).

As a final test, we further instrumented the metal robot with four range sensors that could detect obstacles up to 30.0 cm away, one on each corner of the robot's body facing forward and backward. We programmed the robot with four gaits from the design tool (straight forward motion, straight backward motion, left turn, and right turn), and we manually programmed the robot to choose a gait depending on its sensor input. The robot was able to navigate an obstacle course of cardboard boxes without colliding with the environment (ref. Figure 8-10(b)). This exercise demonstrates the ability to program high level behaviors that incorporate sensory information and build upon the robot gaits designed using the methods in this thesis.

8.3 Summary

We have described and demonstrated two methods for rapid fabrication that can be used to create medium-to-large scale foldable robots. Previous work in designing foldable robots often assumes that the material is infinitely thin, resulting in physical implementations that can not sustain much load [170]. Unlike previous work that uses flexible materials to fold robots, we use thick, rigid material, and we show how these materials increase the load-bearing capabilities of the resulting devices. We discuss how to generate the patterns for these fabrication processes given the same fold patterns that are created using our design-by-composition approach. We have verified our method by fabricating two medium-scale hexapods and one larger-scale legged-wheeled robot with different dimensions and using different thicknesses of material.

One limitation of these approaches is that the thickness of the material, unlike thin films, is non-neglible. The fabrication plan must take this thickness into account. In this chapter, we trim the material and shift faces into order to avoid self-intersection during folding. However, this approach is not guaranteed to work in the general case. Since most of our robot designs are simple trees, we have not yet encountered any issues. Future work includes further investigating these fabrication approaches and leveraging theoretical work in rigid origami [93, 175] to address these issues.

A main difference between these fabrication approaches and print-and-fold is that complex 3-D joints could be incorporated into 3-D printed designs without adding additional complexity to the fabrication process. In the future, we would like to investigate what other kinds of joints we can achieve starting from rigid materials.

Finally, changing the scale of the robots required making modifications to the electronics and software for the robots. In particular, stronger servomotors were used compared to the smaller robots. This resulted in higher voltage and current requirements, which necessitated more complex circuitry. Future work includes incorporating information about actuation and electronics requirements into our design system and fabrication process.

Chapter 9

Conclusion

In this thesis, we have presented a composition approach to foldable robot design and prototyping. Our approach is based on the idea that robots can be composed from a base set of design modules. We have described the minimal set of foldable joints that can be used in general robot designs, as well as composition algorithms for combining them with each other and with rigid bodies with validity guarantees. Additionally, we have described joint controller modules that can be used in ground robot designs and outlined how designs can be simulated to predict performance. All of our modules are parameterized to provide designers with greater flexibility beyond mere composition. Finally, we have integrated these technical contributions into an interactive design tool that provides users with real time feedback on their robot's expected performance as they put it together.

Our approach has allowed us to design a collection of folded mechanisms and robots created using multiple different fabrication methods, each taking less than a day to fabricate and assemble. The robots span many different geometries, motions, and scales. A user study also demonstrated that our methods are accessible and intuitive to designers with no previous robotics experience, and they were able to use our interactive tool to produce a variety of virtual designs.

Our results show that design by composition simplifies robot specification and can enhance current rapid prototyping methods by providing roboticists with an intuitive framework for robot design and exploration. We envision that such methods will

lower the barrier to robot design, not only for aspiring robotics engineers, but also for nontechnical end users, and that similar systems and tools will bring customized robots into the hands of the general public.

9.1 Lessons Learned

Throughout the course of this thesis and beyond the technical contributions in folding and design methodology, I have learned a great deal about the process of robot design and the effect of design tools on designers.

First, the most important lesson I have learned is that physically implementing a robot design is a must. Simulations and theory are only as good as the assumptions that are made. At every step of this thesis, I have encountered practical concerns that have shaped the next steps. Flexibility of material led to the development of methods for fabricating more rigid robot bodies. Complexity of our joint designs encouraged us to explore the combination of folding and 3-D printing. Even now at the conclusion, the robots continue to demonstrate interesting limitations and unexpected behaviors that motivate our future work.

Second, I have learned that sacrificing some optimality for ease of use is a fair trade. The direct application here is that designing fold patterns is generally difficult for people with limited folding experience, but attempts to automate fold pattern design under the stringent requirements imposed by traditional folding theorists have made little headway. By allowing for waste material to be added to a folded structure, or by adding design flexibility by using 3-D printing, we were able to produce foldable robot designs with guaranteed fold pattern validity. As a result, users interacting with our design-by-composition framework can focus on high-level 3-D aesthetics and functionality without concerning themselves with the details of implementation.

Third, I have learned that visualization tools are key, particularly when designing physical objects. It is entirely possible to create a design tool in which users do not see the final design until it is fabricated, and this is the approach taken by some previous systems [48, 113, 163]. However, visualization tools provide quick feedback that have

been invaluable for testing and verifying our methods. From simple visualizations, such as plots of fold patterns and their simulated folded state, to more complex ones, such as displaying the effect of direct parameter changes on designs both in 2-D and in 3-D, all the way to our complete interactive design tool for full pipeline testing, these tools have all played a major role in the success of this work.

Fourth, I have learned that user studies are a necessary part of the development of any design framework. During the course of this research, many decisions had to be made about what elements of design are the most important and what features are key for users. Most of these decisions were resolved by considering what challenges were most technically interesting and what directions of work would lead to the most significant affects on the design process. From the point of view of the users who interacted with our system, however, technical novelty paled in comparison to ease of use and intuitiveness. Even design decisions made specifically for the interactive tool showed a gap between our expectations and the desires of users. Metrics that we thought were important, such as wobbliness or slip, were declared not meaningful by users in our study. They also requested features such as connecting patch visualization and undo buttons, which we never considered because we had never needed them ourselves. The study has provided us with invaluable feedback for future development.

Finally, I have learned that many designers desire low-level control over their design. While discussing this project with other engineers, I have received pushback at times that there are particular aspects of a design that they would like to control. This feedback led us to develop our particular composition implementation, which allows for varying levels of design specification. Novice users interacting with our tool can use the drag and drop interface without thinking about the internals of the system. For more experienced designers, our database is written in a plain text format accessible to people who would like to create custom modules themselves. Our methods also provide key points at which designers can impose their will by directly specifying particular parameter values, indicating which edges in a fold pattern to join, etc. The design-by-composition framework is especially well suited to tasks performed by users at a variety of skill levels.

9.2 Limitations

Of course our system is subject to a few limitations. First and foremost, as with any data-driven method, our design space is restricted to the designs that can be composed from the database using our grammar. Expert designers can expand the set of building blocks from which users can compose their own designs, but novice users have little recourse to create fully customized parts. Methods for expanding the database that are accessible to novice users are needed to overcome this limitation.

With regards to the animation and simulation, our tool currently only considers the geometry and kinematics of the robot. As a result, the autogenerated electronics modules in all of our robots are the same. However, actuation is often a large challenge in robot design. Simulations that incorporate environmental or task constraints such as load, dynamic forces, and robustness in order to generate more applicable robot models are needed. Material properties can also be incorporated to check whether the thin-shell structures produced by folding are appropriate to a task.

For applications, our joint controllers and design tool are currently limited to ground locomotion of a particular type. As we showed, it is possible to expand beyond this task to other similar motions using the modules in the database. However, expanding to other robot applications will require not only expanding the database, but also designing new simulation techniques and performance metrics.

Finally, our feedback and guidance tools are limited to local design changes. Users have the ability to change individual parameters, and the system will ensure that the other parameters change accordingly. Our tools have no ability to suggest module replacements or additions to achieve particular goals. The classification of robot parts lends itself to module replacement, but determining when new modules need to be added in order to achieve certain metrics (e.g., adding another leg to decrease wobbliness) will require more in-depth reasoning about the design.

9.3 Future Work

This thesis has only scratched the surface of robot design tools and design by composition. There are many possible future directions to take this work.

One direction is the question of concurrent design. This work considers the interplay between geometry and motion in a design's functionality. However, there are many aspects of a robot's design: not just the geometrical, but also issues of material properties, fabrication tolerances, actuation requirements, electrical requirements, sensory feedback, etc. Each of these subsystems has its own set of required knowledge and design principles. Many of them already have existing tools to aid in their design. But it is the interplay of these systems that becomes interesting. Take our example of a walking robot. The robot is fabricated out of a particular material. This material has weight, which induces certain actuation requirements. The motors and computing infrastructure in turn have certain power requirements. However, motors and batteries are high-mass components that increase the required stiffness of the robot body and the required actuation. A true robot design tool must be able to incorporate these types of inter-system relationships.

Specifically, we are interested in expanding design to include robot behaviors. In our current tools, users are able to design different gaits that they can compose manually. By contrast, in real-world applications that contain some uncertainty, robots must have the ability to react to differences between their current environment and the simulation. Designing for this behavior requires incorporating sensory and information modules into the system, as well as controller modules that use feedback from these sensors to make gait and other behavioral decisions.

Verifying that these behaviors achieve the user's desired goal requires future work in the realm of functional specifications. Our system currently requires users to specify the exact modules and parameters for a robot design. We have included the ability to automatically optimize design parameters, but this is the extent of the system's design reasoning. Casual users, by comparison, often will not want to design a complete robot, but instead will want something that simply served a

particular function. For these users, the ability to provide robot functionality and have the system automatically compose and specify the design will be important. There exists some previous work on design automation [30,99] and formal methods [82] that provide clues to how functionality can be described, but these methods still require a large amount of user expertise.

The particular challenge with functional specification is that it is not always clear to inexperienced designers what design criteria are most important. For manipulation tasks, payload capacity and position accuracy may be most important, while for exploration tasks, robustness against environmental disturbances may be the dominating requirement. Therefore, future work in design tools should be able to infer what design metrics are important for particular tasks and provide users with feedback not only on how to change their design to achieve their desired metrics but also on which metrics they might want to consider.

Solving challenges such as these will lower the barrier to entry for users to have input into the design of the robots they use.

Appendix A

Sample Output Fabrication Plan

This section provides the fabrication plans for the robot in Figure A-1. The plans include a 3-D mesh to print on a 3-D printer, a list of electronics components and pin connections, and software to load onto the Arduino. The robot consists of an L-shaped body, two front legs, two back wheels, and two roof-like peripherals. It is 127 mm $\times$ 82 mm $\times$ 140 mm and walks forwards by rotating first the left leg, then the right leg, and then shifting both legs and wheels forward simultaneously. The robot walks forward at a speed of 14.56 mm/s. It was printed in 4.4 hr. and assembled in 40 min.

Figure A-1: Fabricated moving house robot, left and right views

A.1 Generated Print

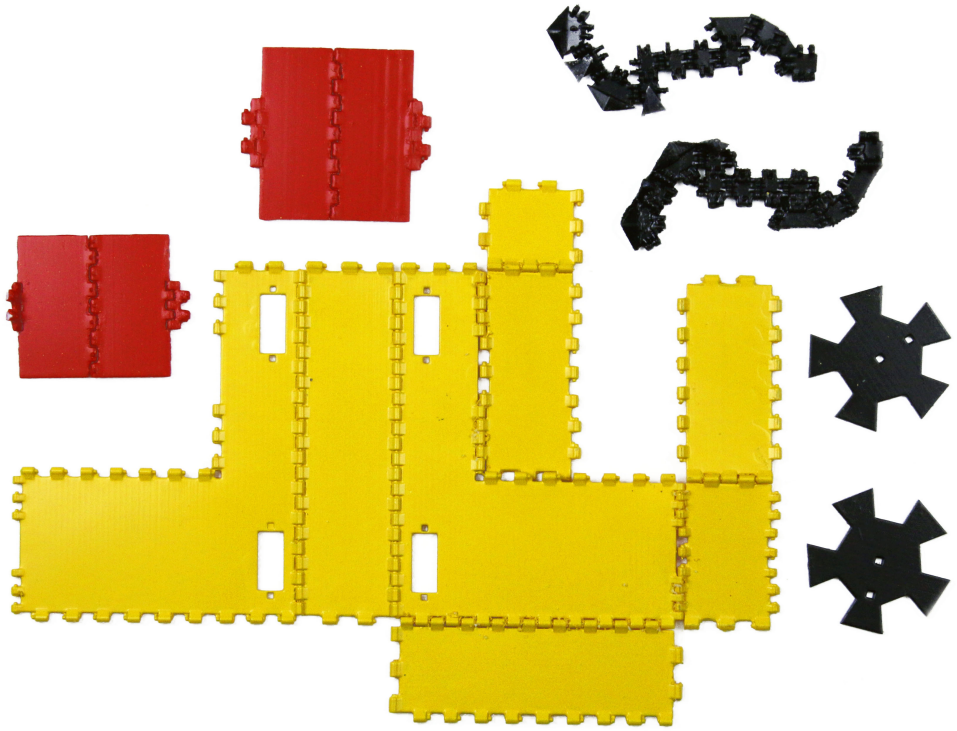


Figure A-2: Generated print for fold pattern of the moving house

A.2 Electronics Plan

```
PARTS LIST:
  1 x  Arduino
  1 x  Battery
  4 x  Servo


CONNECTIONS:
  Connect Battery0:+ to Arduino0:+
  Connect Battery0:- to Arduino0:-
  Connect Servo0:+ to Arduino0:+
  Connect Servo0:- to Arduino0:-
  Connect Servo0:in to Arduino0:D03
  Connect Servo0:out to Arduino0:A0
  Connect Servo1:+ to Arduino0:+
```

```
Connect Servo1:- to Arduino0:-
Connect Servo1:in to Arduino0:D05
Connect Servo1:out to Arduino0:A1
Connect Servo2:+ to Arduino0:+
Connect Servo2:- to Arduino0:-
Connect Servo2:in to Arduino0:D06
Connect Servo2:out to Arduino0:A2
Connect Servo3:+ to Arduino0:+
Connect Servo3:- to Arduino0:-
Connect Servo3:in to Arduino0:D09
Connect Servo3:out to Arduino0:A3
```

A.3 Generated Code

gaitdef.h (Auto-generated file)

```
1  // GAIT-SPECIFIC VARS
2  #define numUsedServos 4
3  #define numGait       2
4  #define trajSize      3
5  #define numWheelPose  6
6
7  const int TYPE[numUsedServos] = {WHEEL, SINGLE, SINGLE, WHEEL};
8
9  const int numPhase[numGait] = {2, 1};
10 const int phases[numGait][numUsedServos] = {{2, 0, 1, 2},
11                                             {1, 0, 0, 1}};
12
13 volatile int STATE[numUsedServos] = {BACKMOVE, BACKMOVE,
14                                      BACKMOVE, BACKMOVE};
15 volatile int valback[numUsedServos];
16
17 const int forpos[numGait][numUsedServos]  =  {{100, 100, 100, 100},
18                                                { 90,  90,  90,  90}};
19 const int backpos[numGait][numUsedServos] = {{150, 150, 150, 150},
20                                               {160, 160, 160, 160}};
21 const int midpos[numGait][numUsedServos]  = {{125, 125, 125, 125},
22                                               {125, 125, 125, 125}};
23 const int retrpos[numGait][numUsedServos] = {{100, 100, 100, 100},
24                                               { 90,  90,  90,  90}};
25
26 const int wheelposes[numWheelPose] = {155, 131, 107, 83, 59, 35};
27
28 const int trajectory[trajSize] = {0,1,0};
```

main.ino (Template file)

```
#include "arduino.h"
#include "robotLibrary.h"

// gait and trajectory definitions
#include "gaitinfo.h"
#include "gaitdef.h"

int iGAIT; // index in trajectory
int GAIT;  // id of gait currectly executing
int PHASE; // phase of gait
int val;   // temp variable, value read from analog pin
int whichWheelPose; // current position of wheels

void setup()
{
  Serial.begin(9600);

  // setup servos:
  // servo0: pin 3
  // servo1: pin 5
  // servo2: pin 6
  // servo3: pin 9
  // servo4: pin 10
  // servo5: pin 11
  robotSetup();

  //setup servo potentiometer analog input
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);
  pinMode(A4, INPUT);
  pinMode(A5, INPUT);

  // initialize vars tracking gait and trajectory
  iGAIT = numGait-1;
  GAIT = trajectory[iGAIT];
  PHASE = numPhase[GAIT];
  whichWheelPose = 0;

  // intialize servo positions
  for (int i = 0; i < numUsedServos; ++i) {
    if (i == 1 || i == 2) {
      setPWM(servoPins[i], backpos[GAIT][i]);
    } else {
      setPWM(servoPins[i], backpos[GAIT][i]/2);
    }
  }
}

// FOLLOW DESIGNED GAIT
```

```
52  void loop()
53  {
54    boolean done = true;
55
56    if (PHASE == numPhase[GAIT]) {
57      // LAST PHASE, SHIFT ALL LIMBS
58
59      for (int i = 0; i < numUsedServos; ++i) {
60        if (TYPE[i] == SINGLE || TYPE[i] == DOUBLE) {
61          // legs
62          if (STATE[i] == FORWMOVE) {
63            continue;
64          }
65
66          valback[i] = analogRead(servoPot[i]);
67          if (i == 1 || i == 2) {
68            setPWM(servoPins[i], forpos[GAIT][i]);
69          } else {
70            setPWM(servoPins[i], forpos[GAIT][i]/2);
71          }
72          STATE[i] = FORWMOVE;
73        } else {
74          // wheels
75          if (whichWheelPose == numWheelPose - 1) {
76            valback[i] = analogRead(servoPot[i]);
77          }
78
79          val = analogRead(servoPot[i]);
80          if ((whichWheelPose == 0) && (val < valback[i])) {
81            setPWM(servoPins[i], controt);
82            done = false;
83          } else {
84            setPWM(servoPins[i], wheelposes[whichWheelPose]);
85            STATE[i] = FORWMOVE;
86          }
87        }
88      }
89
90      if (done) {
91        whichWheelPose = (whichWheelPose + 1) % numWheelPose;
92        delay(100);
93        iGAIT = (iGAIT + 1) % trajSize;
94        GAIT = trajectory[iGAIT];
95        PHASE = 0;
96      }
97    } else {
98      // MOVE THE LIMBS IN THE STEP
99      for (int i = 0; i < numUsedServos; ++i) {
100       if (phases[GAIT][i] != PHASE) {
101         continue;
102       }
103
104       switch (STATE[i]) {
105         case FORWMOVE: {
```

```
106           if (TYPE[i] == DOUBLE) {
107             if (i == 1 || i == 2) {
108               setPWM(servoPins[i], retrpos[GAIT][i]);
109             } else {
110               setPWM(servoPins[i], retrpos[GAIT][i]/2);
111             }
112             STATE[i] = PAUSE1;
113           } else {
114             STATE[i] = CONTMOVE;
115           }
116           done = false;
117           } break;
118         case PAUSE1: {
119           STATE[i] = CONTMOVE;
120           done = false;
121           } break;
122         case CONTMOVE: {
123           if (TYPE[i] == SINGLE) {
124             setPWM(servoPins[i], controt);
125             val = analogRead(servoPot[i]);
126             if (val > valback[i]) {
127               STATE[i] = PAUSE2;
128             }
129           } else if (TYPE[i] == DOUBLE) {
130             if (i == 1 || i == 2) {
131               setPWM(servoPins[i], midpos[GAIT][i]);
132             } else {
133               setPWM(servoPins[i], midpos[GAIT][i]/2);
134             }
135             STATE[i] = PAUSE2;
136           } else {
137             STATE[i] = BACKMOVE;
138           }
139           done = false;
140           } break;
141         case PAUSE2: {
142           STATE[i] = BACKMOVE;
143           done = false;
144           } break;
145         case BACKMOVE: {
146           if (TYPE[i] == SINGLE || TYPE[i] == DOUBLE) {
147             if (i==1 || i==2) {
148               setPWM(servoPins[i], backpos[GAIT][i]);
149             } else {
150               setPWM(servoPins[i], backpos[GAIT][i]/2);
151             }
152           }
153           } break;
154       }
155     }
156     delay(150);
157     if (done) {
158       delay(100);
159       PHASE = PHASE + 1;
```

```
    }
  }
}
```

gaitinfo.h (Template file)

```
// GENERAL GAIT VARS
#define controt   4
#define rconrot 250

#define SINGLE   0
#define DOUBLE   1
#define WHEEL    2

#define BACKMOVE  0
#define FORWMOVE  1
#define CONTMOVE  2
#define PAUSE1    3
#define PAUSE2    4

const int servoPot[numServos] = {A0, A1, A2, A3, A4, A5};
```

robotLibrary.h (Template file)

```
#ifndef INCL_ROBOTLIBRARY_H
#define INCL_ROBOTLIBRARY_H

#define numServos 6
#include "string_functions.h"
#include "arduino.h"
#define DO 0
#define DI 1
#define AO 2
#define AI 3
#define PWM 4
#define SERVO 5
#define numPins 28

extern int servoPins[numServos];

int getPinMateType(int pin);
int getPinType(int pin);
int setPWMFrequency(int pin, long frequency);
long getPWMFrequency(int pin);
void robotSetup();
void setPinMode(int pin, int mode);
void setPWM(int pin, int duty);

#endif
```

robotLibrary.cpp (Template file)

```
#include "robotLibrary.h"

int servoPins[numServos];
long PWMFrequency[3];
int controllerPins[] = {-1, 3,-1,-1,-1,-1,-1,-1, 5,-1,-1,-1,-1,-1,-1,
                        6,-1,-1,-1,-1,-1,-1, 9,10,11,-1,-1,-1};
const char* pinTypes[] = {"DataInputPort","ServoInputPort",
                          "PowerInputPort","Ground","DataInputPort",
                          "DataOutputPort","DataOutputPort",
                          "DataInputPort","ServoInputPort",
                          "PowerInputPort","Ground","DataInputPort",
                          "DataOutputPort","DataOutputPort",
                          "DataInputPort", "ServoInputPort",
                          "PowerInputPort","Ground","DataInputPort",
                          "DataOutputPort","DataOutputPort",
                          "DataInputPort","ServoInputPort",
                          "ServoInputPort","ServoInputPort",
                          "DataInputPort","DataOutputPort",
                          "DataOutputPort"};


int getPinMateType(int pin)
{
  if(contains(pinTypes[pin], "DigitalOutput"))
    return DI;
  else if(contains(pinTypes[pin], "DigitalInput"))
    return DO;
  else if(contains(pinTypes[pin], "PWMOutput"))
    return DI;
  else if(contains(pinTypes[pin], "AnalogOutput"))
    return AI;
  else if(contains(pinTypes[pin], "AnalogInput"))
    return PWM;
  else if(contains(pinTypes[pin], "ServoInput"))
    return SERVO;
  return -1;
}

int getPinType(int pin)
{
  if(contains(pinTypes[pin], "DigitalOutput"))
    return DO;
  else if(contains(pinTypes[pin], "DigitalInput"))
    return DI;
  else if(contains(pinTypes[pin], "PWMOutput"))
    return PWM;
  else if(contains(pinTypes[pin], "AnalogOutput"))
    return AO;
  else if(contains(pinTypes[pin], "AnalogInput"))
    return AI;
  else if(contains(pinTypes[pin], "Servo"))
```

```
52      return SERVO;
53    return -1;
54  }
55
56  // Set the PWM Frequency for the given pin
57  // Will return the frequency achieved, or -1 if arguments are invalid
58  int setPWMFrequency(int pin, long freq)
59  {
60    byte mode;
61    long baseFrequency;
62    if(pin == 3 || pin == 9 || pin == 10 || pin == 11)
63        baseFrequency = 31250;
64    if(pin == 5 || pin == 6)
65        baseFrequency = 62500;
66    long error = baseFrequency;
67    int divisor = 1;
68    if(pin == 5 || pin == 6 || pin == 9 || pin == 10)
69    {
70      int divisors[] = {1,8,64,256,1024};
71      for(int i = 0; i < 5; i++)
72      {
73        long newError = freq - baseFrequency / divisors[i];
74        newError *= newError < 0 ? -1 : 1;
75        if(newError < error)
76        {
77          error = newError;
78          divisor = divisors[i];
79        }
80      }
81      Serial.print("divisor:␣");
82      Serial.println(divisor);
83
84      Serial.print("frequency:␣");
85      Serial.println(baseFrequency / divisor);
86
87      Serial.print("error:␣");
88      Serial.println(error);
89
90      switch(divisor)
91      {
92        case 1: mode = 0x01; break;
93        case 8: mode = 0x02; break;
94        case 64: mode = 0x03; break;
95        case 256: mode = 0x04; break;
96        case 1024: mode = 0x05; break;
97        default: return -1;
98      }
99      if(pin == 5 || pin == 6)
100       TCCR0B = TCCR0B & 0b11111000 | mode;
101     else
102       TCCR1B = TCCR1B & 0b11111000 | mode;
103   }
104   else if(pin == 3 || pin == 11)
105   {
```

```
106     int divisors[] = {1,8,32,64,128,256,1024};
107     for(int i = 0; i < 7; i++)
108     {
109       long newError = freq - baseFrequency / divisors[i];
110       newError *= newError < 0 ? -1 : 1;
111       if(newError < error)
112       {
113         error = newError;
114         divisor = divisors[i];
115       }
116     }
117     switch(divisor)
118     {
119       case 1: mode = 0x01; break;
120       case 8: mode = 0x02; break;
121       case 32: mode = 0x03; break;
122       case 64: mode = 0x04; break;
123       case 128: mode = 0x05; break;
124       case 256: mode = 0x06; break;
125       case 1024: mode = 0x7; break;
126       default: return -1;
127     }
128     TCCR2B = TCCR2B & 0b11111000 | mode;
129   }
130   return baseFrequency / divisor;
131 }
132
133 long getPWMFrequency(int pin)
134 {
135   switch(controllerPins[pin])
136   {
137     case 5:
138     case 6:
139       return PWMFrequency[0];
140     case 9:
141     case 10:
142       return PWMFrequency[1];
143     case 3:
144     case 11:
145       return PWMFrequency[2];
146   }
147 }
148
149 void robotSetup()
150 {
151   // Set each pin to the correct mode
152   for(int pinIndex = 0; pinIndex < numPins; pinIndex++)
153   {
154     if(controllerPins[pinIndex] >= 0 && getPinType(pinIndex) >= 0)
155       setPinMode(controllerPins[pinIndex], getPinMateType(pinIndex));
156   }
157   servoPins[0] = 1;
158   setPWM(servoPins[0], 128);
159   servoPins[1] = 8;
```

```
160   setPWM(servoPins[1], 128);
161   servoPins[2] = 15;
162   setPWM(servoPins[2], 128);
163   servoPins[3] = 22;
164   setPWM(servoPins[3], 128);
165   servoPins[4] = 23;
166   setPWM(servoPins[4], 128);
167   servoPins[5] = 24;
168   setPWM(servoPins[5], 128);
169 }
170 
171 void setPinMode(int pin, int mode)
172 {
173   switch(mode)
174   {
175     case DO: pinMode(pin, OUTPUT); break;
176     case DI: pinMode(pin, INPUT); break;
177     case AO:
178     case PWM: pinMode(pin, OUTPUT); break;
179     case SERVO:
180       pinMode(pin, OUTPUT);
181       switch(pin)
182       {
183         case 5:
184         case 6:
185           PWMFrequency[0] = setPWMFrequency(pin, 980);
186           break;
187         case 9:
188         case 10:
189           PWMFrequency[1] = setPWMFrequency(pin, 480);
190           break;
191         case 3:
192         case 11:
193           PWMFrequency[2] = setPWMFrequency(pin, 480);
194           break;
195       }
196       break;
197     case AI: break;
198   }
199 }
200 
201 void setPWM(int pin, int duty)
202 {
203   analogWrite(controllerPins[pin], duty);
204 }
```

string_functions.h (Template file)

```
1  #ifndef INCL_STRING_FUNCTIONS
2  #define INCL_STRING_FUNCTIONS
3  
4  #include <stdlib.h>
5  
```

```
6  inline int length(const char* source);
7  inline int indexOf(const char* source, const char* target);
8  inline bool contains(const char* source, const char* target);
9
10 // Gets the length of the character array
11 int length(const char* source)
12 {
13   int length = 0;
14   for(; source[length] != '\0'; length++);
15   return length;
16 }
17
18 // Gets the index of the given string, or -1 if not found
19 int indexOf(const char* source, const char* target)
20 {
21   int targetLength = length(target);
22   int sourceLength = length(source);
23   int index = -1;
24   for(int i = 0;  i <= sourceLength-targetLength && index == -1; i++)
25   {
26     bool foundTarget = true;
27     for(int n = 0; n < targetLength && i+n < sourceLength; n++)
28     {
29       if(source[i+n] != target[n])
30         foundTarget = false;
31     }
32     if(foundTarget)
33       index = i;
34   }
35   return index;
36 }
37
38 // Checks if the source string contains the target string
39 bool contains(const char* source, const char* target)
40 {
41   return indexOf(source, target) >= 0;
42 }
43
44 #endif
```

Appendix B

User Study Questionnaire Responses

The following questionnaire responses taken from the user study described in Chapter 7.2 are organized by question and then subject number.

B.1 Pre-study Questionnaire

Self-reported measures of competency Rating: (1 = None, 5 = Very familiar)

	Familiarity with CAD tools	**Experience with robots**
1:	2	1
2:	1	2
3:	1	1
4:	5	3
5:	2	3
6:	3	3
7:	5	3
8:	2	1

B.2 Task 1 Questionnaire

Task Description

Users were asked to design "an interesting car for a parade." They had access to the entire geometry database. They were given 10 min. to complete the task.

Questionnaire Responses

Describe your strategy in designing the car.

1: Find random parts, try to connect.

2: It was mostly random. I put different pieces together and they fit in some unexpected way that was interesting. I keep some stability in mind, and try to make things symmetric, which is not easy to do when you use graphical tools to specify dimensions of two wheels.

3: start with body, then add parts that will move the car, make sure it's stable, then start adding peripherals while checking that it remains stable (motion sequence is reasonable)

4: Symmetrical with angled pieces for aerodynamics

5: I picked an interesting, not traditionally shaped car body, then picked out interesting wheels and some peripheries and attached them.

6: I started with a simple body template. then added wheels. I resized the body to be similar to the base of a parade float, with the intention of adding structures on top of it. I then tried to add another body to the top of the parade float. Based on the fabrication constraints, this proved somewhat difficult, as you are required to connect edge to edge for fabrication purposes, so I had to try two different bodies to find one that worked. I then added a fun peripheral to the back of the car.

7: I started with the bias that a car has four wheels and a body. Then I chose an attractive body based on the choices and added peripherals that I thought added attractiveness to the car. My choice and creativity in designing the car was heavily influenced by the geometry choices available.

8: I chose a body, and the wheels, then the peripherals. I dragged around parts to see suggestions of where it could be attached. After attaching the parts, I adjusted the size and position.

How satisfied were you with the design tool?

1: Good. It was inspirational.

2: Moderately. The scaling and translating behave in some unexpected way. For example, the wheel doesn't move the same amount that I'm moving my cursor. Additionally, it may be interesting to preview how each components will be attached as we move the new piece around.

3: pretty easy to use, easy to connect parts to each other (with the guides), and I like that changing one part changes the other parts (e.g. rescales or moves them to accommodate)

4: Fairly. Wish connection points remained highlighted all the time so I know where they are, then secondary highlight color when 'active'.

5: I was satisfied until I reached the stage where I was adding peripheries. After attempting to attach peripheries and the tool shrinking my design with no apparent way to undo what I had done, I was a bit dissatisfied.

6: The design tool was very creative at interpreting what I wanted to do, but it was sometimes difficult to line up the parts that I wanted to line up.

7: Mildly satisfied

8: The guides (constraints on where parts could be attached) helped me. On the other hand, sometimes I wanted to do things that the constraints did not allow me to do (e.g. attach small peripherals as decoration on a place where there was no "tabs"). Overall, the process was easier than free-form design, especially if I accept the given constraints.

How satisfied were you with the creative freedom/constraints in the system? Was it expressive enough to make the car you wanted?

1: Not really. The variation of parts is rather restricted.

2: Very, I think we were offer diverse components and many parameters to adjust in a very convenient way. Having graphics input is great when we want to customize each piece to our liking. It's definitely expressive enough. However, there are some unexpected behavior mentioned in the previous question that makes it challenging to get the specification we want within the limited time of the study.

3: yeah, i only needed to figure out that the wheels need to touch the ground, so that meant I had to change the scale of the wheels for the robot shape I chose because adjusting their position didn't work

4: Largely yes

5: There were many options to choose from in designing the car.

6: It was definitely expressive enough to make the car that I wanted, but there was certainly some limitations associated with the edge to edge matching. Having to match edge to edge limits the design space, which I think is actually interesting.

7: The available choices did influence my creativity - I am mildly satisfied with the system. It would have been hard to design my own design - it constrained my design to what was feasible with the preexisting geometric shapes. It expanded some creative front as it provided some shapes that I would have not considered hadn't they've been available in the system. Even though the system limits creativity to some extent due to the limited number of geometric shapes, it does make the design process faster due to the easy scaling, translating, and snapping features.

8: The constraints hindered me from designing a free-form car, but it did help me to easily design a car that would work ("be printable" with foldable parts).

What was the most useful feature that you used?

1: CONNECT

2: I like the animating feature. I used it very briefly, but I think it was a good sanity check tool.

3: connection of parts!

4: Scale

5: scaling.

6: The most useful feature was the rescaling tool. Being able to granularly change part size was incredibly expressive.

7: The scaling and translating tools were the most useful.

8: Suggestion on where partscould be attached.

List any design tool features that you would have liked to use that were not present.

1: Redo/Undo. Simultaneous scaling of multiple parts.

2: The ability to preview how each pieces would connect before having to press connect.

3: adjust scale or position of multiple parts at the same time

4: I see there is a symmetry tab but I haven't used it. Something that does standard mirroring about a midplane would be nice, if it's not in there.

5: An undo option.

6: I would have liked a point and click feature that allowed me to connect side A of object 1 to side B of object 2.

7: The snapping/mating surfaces were limited in some shapes. I would have liked to have the option to snap to different areas of a shape.

8: Automatic symmetry; making parts attachable.

Additional comments

1:

2:

3: fun! lots of flexibility

4:

5:

6:

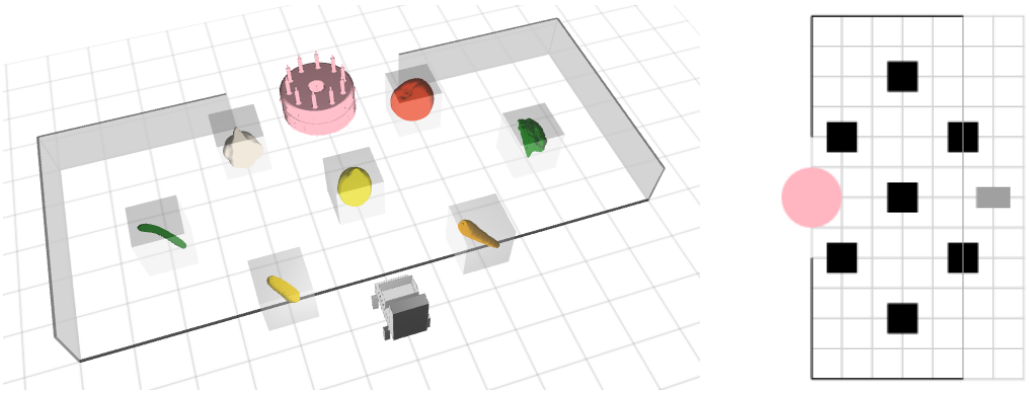
7:

8:

B.3 Task 2 Questionnaire

Task Description

Users were given a robot design and asked to make it walk from the start location to the goal (the cake) without hitting any obstacles (the vegetables). All of the users were given full access to the gait design tool. Half of the users were allowed to modify the robot geometry. They were given 15 min. to complete the task.



(a) 3-D view (b) 2-D top view

Figure B-1: Obstacle course used in task 2 of user study. Users were asked to make the robot walk from the starting location to the cake without hitting any vegetables.

Questionnaire Responses

Describe your strategy in getting the robot to the goal.

1: Create two "turning" gaits, use them alternatively.

2: I try to build forward, and turning, and build the trajectory from there.

3: First figure out if robot is stable, then figure out how gaits work, then compose gait sequence, and iterate between sequencing gaits and adjusting robot dimensions to avoid obstacles.

4: Move fwd, slight turns, stay close to center obstacle

5: Create 2 gaits: 1 that moves the robot in a straight line, another that turns the robot.

6: Develop a clear left and right turn sequence, find a forward sequence, then move right to go around the obstacle in my way, then go straight, then turn left to reach the cake. I did not modify the geometry of the robot because I was told not to.

7: The strategy was to select the right sequence of forward moves and rotations to achieve the right composed path towards the cake. I knew that the robot would have to move forward, then turn right, and then turn left and forward towards the goal. I just adjusted the number of movements for each type. The strategy required that each of the movements worked well. So the first task was to check the 3 basic movements (forward, right and left turns) to make sure that they worked properly (and modify the robot to make sure that they did).

8: First, I reduced the wobbliness of the robot by changing its geometry using the manipulated guidance. Then, I designed three gaits: (1) straight forward, (2) turn left, and (3) turn right. Once I had these gaits, it was easy to compose them into the motion path I wanted. I used the motion path guide to plan my trajectory and animated to confirm that the robot did not collide.

How satisfied were you with your design?

1: 90

2: Mostly, I hit the corn a little bit, but we get to the cake. I wish I figured out how to walk faster.

3: robot design worked out, just needed to adjust gait a bit to reduce angle and avoid collision

4: Very

5: Fairly satisfied.

6: I was very happy with my overall approach, but I suspect that there is a faster forward gait than the one I used.

7: Very satisfied!

8: I was very satisfied!

Rate your agreement with the following statements.

	I completed the task.	**I experimented with the gait.**	**I experimented with the geometry.**
1:	Neutral	Agree strongly	Disagree strongly
2:	Agree	Agree strongly	NA
3:	Agree	Agree strongly	Agree strongly
4:	Agree strongly	Agree	Agree
5:	Agree	Agree strongly	NA
6:	Agree strongly	Agree strongly	Disagree strongly
7:	Agree strongly	Agree strongly	Agree strongly
8:	Agree strongly	Agree strongly	Agree strongly

What was the most useful feature that you used?

1: Trajectories

2: trajectory planner

3: animation sequence, projected path and watching speed metrics

4: Gait leg angle edit to get desired turn angle

5: The map and arrows that shows the trajectory of the robot.

6: The animation tool.

7: The gait animator

8: The motion path preview.

List any design tool features that you would have liked to use that were not present.

1: It would be great if the trajectories are associated with a "beam" that takes the size of the robot into account.

2: reset button to reset the position of the robot after it had toppled while I experiment with my gait

3: ability to go back to a particular point in time (animate from then on, or adjust the robot from then on); ability to see stop-frame animation frames to view whole trajectory (to see where robot would have gotten stuck)

4: Way to drop gaits in between others in the full animation. copy/paste/delete of animated gaits in right hand list

5: Suggestions or guidance on what thetas to select to move the robot from one position to another.

6: A way to stop the animation tool.

7:

8: It would be useful to see the "shadow" of the robot on top of the trajectory of its center of mass, so we can know for sure that it does not collide with objects.

Additional comments

1:

2:

3:

4:

5: Experimenting with thetas was a bit slow because I would wait for the map on the right to update.

6:

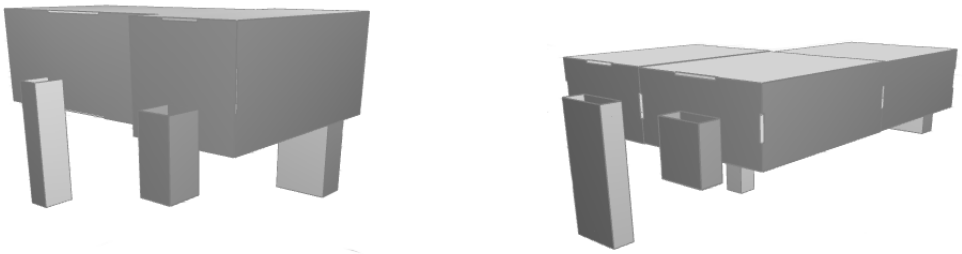
7:

8:

B.4 Task 3 Questionnaire

Task Description

Users given two robot designs and asked to change their dimensions to achieve certain velocity and wobbliness goals. They modified one design with feedback and guidance and one without. The order of the robots and task conditions was randomized. They were given 5 min. to complete each task.



Robot A
$V > 150$ mm/s
$W < 0.5$ rad

Robot B
$V > 190$ mm/s
$W < 0.5$ rad

Figure B-2: Robots and metric goals used in task 3 of user study

Questionnaire Responses

Measures of usage Rating: (1 = Did not use, 5 = Used a lot, NA = no answer)

	With feedback			Without feedback		
	Guidance	**Metrics**	**Animate**	**Guidance**	**Metrics**	**Animate**
1:	NA	5	1	1	1	5
2:	2	3	1	1	1	1
3:	4	5	1	1	5	1
4:	5	5	2	1	5	4
5:	5	5	1	1	2	5
6:	5	5	3	1	3	4
7:	5	4	1	1	4	1
8:	5	4	1	1	1	3

Describe your strategy for stabilizing the robot. (with feedback)

1: Click on "suspecticle parts" and follow the guidance

2: I try to symmetrize the robot as much as possible. Additionally, I observe that wider-based robots are less likely to wobble, so I try to translate the legs as far apart as I can.

3:

4: Find the largest suggestion arrows, move as suggested

5: I iterated through the scale and translate features for each leg using the guidance arrows. Then I focused on scaling the body using the guidance arrows.

6: To stabilize the robot, I equalized leg lengths and widths and made it a symmetric 2x2 pattern (to reduce side to side tipping during motion), then lengthened the main body (to minimize forward-backward tipping). I really like having the guidance arrows, because they provided real time feedback to reinforce my intuition and provided an indicator of when I was getting carried away with something (thinning the body, for example).

7: I relied heavily on the guidance arrows. I did use intuition to determine the components that I wanted to modify and the type of modification, but relied on the guidance arrows to determine the extent and direction of the modification.

8: I was sequentially trying to scale the legs according to the guide's suggestions

Describe your strategy for stabilizing the robot. (without feedback)

1: Find the legs that seem "non-symmetric" to the others, look at the animation to see what's wrong.

2: The same as before

3: Starting with legs, first adjusting legs to be the same size then at the same locations, then playing with width of legs while maintaining speed

4: Tried for symmetry, but didn't seem to help much given the gait

5: I tried to make the legs even heights. After each change I used the animate feature to try to observe "wobbliness."

6: My strategy here was very similar to my strategy for the first case, with the added 'cheat' of collapsing the width of the side panel on the back right to allow the main body of the robot to be similar to the first robot that I had solved in the previous exercise. I didn't use the guidance arrows because they were turned off for this exercise (but I wanted to use them), and I consulted the metrics tab quite frequently as I approached the end of the exercise to see if I was hitting my speed and wobbliness targets. Compared to the first exercise, this one was slightly more stressful, because I didn't have feedback to validate my intuition as I went along.

7: I followed a strategy of modifying two main parameters: the width of the body of the robot, and the length of the legs. My heuristic for the body was: a wider body will be more stable than a thin body. My heuristic for the legs was: legs have to be of equal length.

8: Trying to sequentially scale the legs based on geometric reasoning

Additional comments

1: The UI is a bit laggy in Model #2 (with feedback). I also feel that in Model #2 I am mostly doing repetitive jobs which can be done automatically.

2: The feedback from the arrow was too slow. I happened to have found the solution using my initial strategy before having to rely on the arrow. I do imagine if you give me a highly asymmetric robots, my strategy would have broken down, and I will have to rely on the arrow much more though.

3: lagged a bit for model 2, so took longer because of this, but at least i wasn't in the dark on what changes i had to make

4: Much easier with guidance

5: In model 1 (without guidance) I forgot that you could adjust the body, and this did not occur to me until halfway through model 2. If I had to redo the task (with or without guidance arrows), I would focus on the body first since the guidance arrows and observing the metrics indicated that this makes the biggest change in "wobbliness." Even after the task, I am still unsure what "wobbliness" is supposed to measure.

6:

7: The first task's starting geometry was much more intuitive because the body was close to symmetric. The second task had a more asymmetric design which led me to heavily rely on the guidance arrows to determine the right modification.

8: Instead of showing guides as per user's selection, you can start with showing all available guides such that the user can know where he should focus his efforts

Bibliography

[1] Zachary Abel, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Anna Lubiw, André Schulz, Diane L. Souvaine, Giovanni Viglietta, and Andrew Winslow. Algorithms for designing pop-up cards. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Science*, pages 269–280, 2013.

[2] Adafruit, 2016. `http://www.adafruit.com/`.

[3] Oswin Aichholzer, Franz Aurenhammer, David Alberta, and Bernd Gärtner. A novel type of skeleton for polygons. *Journal of Universal Computer Science*, 1(12):752–761, 1995.

[4] Byoungkwon An and Daniela Rus. Designing and programming self-folding sheets. *Robotics and Autonomous Systems*, 62(7):976–1001, 2014.

[5] Erik K. Antonsson. The potential for mechanical design compilation. *Research in Engineering Design*, 9(4):191–194, 1997.

[6] Daniel M. Aukes, Benjamin Goldberg, Mark R. Cutkosky, and Robert J. Wood. An analytic framework for developing inherently-manufacturable pop-up laminate devices. *Smart Materials and Structures*, 23(9):094013, 2014.

[7] Daniel M. Aukes and Robert J. Wood. PopupCAD: a tool for automated design, fabrication, and analysis of laminate devices. In *Proceedings of SPIE Micro- and Nanotechnology Sensors, Systems, and Applications VII*, page 94671B, 2015.

[8] Ed Ayyappa. Normal human locomotion, part 1: Basic concepts and terminology. *Journal of Prosthetics and Orthotics*, 9(1):10–17, 1997.

[9] Moritz Bächer, Bernd Bickel, Doug L. James, and Hanspeter Pfister. Fabricating articulated characters from skinned meshes. *ACM Transactions on Graphics*, 31(4):47, 2012.

[10] Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. LinkEdit: Interactive linkage editing using symbolic kinematics. *ACM Transactions on Graphics*, 34(4):99, 2015.

[11] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics*, 33(4):96, 2014.

[12] Devin J. Balkcom and Matthew T. Mason. Robotic origami folding. *International Journal of Robotics Research*, 27(5):613–627, 2008.

[13] Noy Bassik, George M. Stern, and David H. Gracias. Microassembly based on hands free origami with bidirectional curvature. *Applied Physics Letters*, 95(9):091901, 2009.

[14] Jon L. Bentley and Thomas A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 100(9):643–647, 1979.

[15] Marshall Bern, Erik D. Demaine, David Eppstein, Eric Kuo, Andrea Mantler, and Jack Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry*, 24(2):51–62, 2003.

[16] Dustin Beyer, Serafima Gurevich, Stefanie Mueller, Hsiang-Ting Chen, and Patrick Baudisch. Platener: Low-fidelity fabrication of 3D objects by substituting 3D print with laser-cut plates. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2015.

[17] Nicola Bezzo, Peter Gebhard, Insup Lee, Matthew Piccoli, Vijay Kumar, and Mark Yim. Rapid co-design of electro-mechanical specifications for robotic systems. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*, pages DETC2015–47472, 2015.

[18] Nicola Bezzo, Ankur Mehta, Cagdas Denizel Onal, and Michael Thomas Tolley. Robot makers: The future of digital rapid design and fabrication of robots. *IEEE Robotics & Automation Magazine*, 22(4):27–36, 2015.

[19] Gaurav Bharaj, Stelian Coros, Bernhard Thomaszewski, James Tompkin, Bernd Bickel, and Hanspeter Pfister. Computational design of walking automata. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 93–100, 2015.

[20] Bernd Bickel, Moritz Bächer, Miguel A. Otaduy, Hyunho Richard Lee, Hanspeter Pfister, Markus Gross, and Wojciech Matusik. Design and fabrication of materials with desired deformation behavior. *ACM Transactions on Graphics*, 29(4):63, 2010.

[21] Landen A. Bowen, Clayton L. Grames, Spencer P. Magleby, Larry L. Howell, and Robert J. Lang. A classification of action origami as systems of spherical mechanisms. *Journal of Mechanical Design*, 135(11):111008, 2013.

[22] David Brandt and David Johan Christensen. A new meta-module for controlling large sheets of ATRON modules. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2375–2380, 2007.

[23] Roger Bush and Carlo Sèquin. Synthesis of bent sheet metal parts from design features. In *Proceedings of the ACM Symposium on Solid Modeling and Applications*, pages 119–129, 1999.

[24] Jacques Calì, Dan A. Calian, Cristina Amati, Rebecca Kleinberger, Anthony Steed, Jan Kautz, and Tim Weyrich. 3D-printing of non-assembly, articulated models. *ACM Transactions on Graphics*, 31(6):130, 2012.

[25] Andrea Censi. Monotone co-design problems; or, everything is the same. In *Proceedings of the American Control Conference (ACC)*, pages 1227–1234, 2016.

[26] Duygu Ceylan, Wilmot Li, Niloy J. Mitra, Maneesh Agrawala, and Mark Pauly. Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics*, 32(6):186, 2013.

[27] Desai Chen, David I. W. Levin, Piotr Didyk, Pitchaya Sitthi-Amorn, and Wojciech Matusik. Spec2Fab: A reducer-tuner model for translating specifications to 3D prints. *ACM Transactions on Graphics*, 32(4):135, 2013.

[28] Desai Chen, Pitchaya Sitthi-amorn, Justin T. Lan, and Wojciech Matusik. Computing and fabricating multiplanar models. *Computer Graphics Forum*, 32(2):305–315, 2013.

[29] Yan Chen, Rui Peng, and Zhong You. Origami of thick panels. *Science*, 349(6246):396–400, 2015.

[30] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pages 167–174, 2013.

[31] Herng Yi Cheng and Kang Hao Cheong. Designing crease patterns for polyhedra by composing right frusta. *Computer-Aided Design*, 44(4):331–342, 2012.

[32] Shean-Juinn Chiou and Kota Sridhar. Automated conceptual design of mechanisms. *Mechanism and Machine Theory*, 34(3):467–495, 1999.

[33] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics*, 30(4):59, 2011.

[34] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Transactions on Graphics*, 32(4):83, 2013.

[35] David W. Currier. Automation of sheet metal design and manufacturing. In *Proceedings of the Conference on Design Automation*, pages 134–138, 1980.

[36] Jian S. Dai and Ferdinando Cannella. Stiffness characteristics of carton folds for packaging. *Journal of Mechanical Design*, 130(2):022305, 2008.

[37] Jay Davey, Ngai Kwok, and Mark Yim. Emulating self-reconfigurable robots-design of the SMORES system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4464–4469, 2012.

[38] Mark De Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, Germany, 2008.

[39] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, New York, NY, USA, 2008.

[40] Dongping Deng and Yong Chen. Assembled additive manufacturing–a hybrid fabrication process inspired by origami design. *Solid Freeform Fabrication*, pages 174–187, 2013.

[41] Dongping Deng and Yong Chen. Origami-based self-folding structure design and fabrication using projection based stereolithography. *ASME Journal of Mechanical Design*, 137(2):021701, 2015.

[42] Krishnamanaswi M. Digumarti, Christian Gehring, Stelian Coros, Je Min Hwangbo, and Roland Siegwart. Concurrent optimization of mechanical design and locomotion control of a legged robot. In *Climbing and Walking Robots (CLAWAR)*, 2014.

[43] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[44] Neel Doshi, Benjamin Goldberg, Ranjana Sahai, Noah Jafferis, Daniel Aukes, Robert J. Wood, and John A. Paulson. Model driven design for flexure-based microrobots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4119–4126, 2015.

[45] Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. Computational multicopter design. In *Proceedings of ACM SIGGRAPH Asia*, 2016. to appear.

[46] Bryce J. Edmondson, Landen A. Bowen, Clayton L. Grames, Spencer P. Magleby, Larry L. Howell, and Terri C. Bateman. Oriceps: Origami-inspired forceps. In *Proceedings of the ASME Conference on Smart Materials, Adaptive Structures and Intelligent Systems (SMASIS)*, pages SMASIS2013–3299, 2013.

[47] David Eppstein and Jeff Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete & Computational Geometry*, 22(4):569–592, 1999.

[48] Arthur G. Erdman, Thomas Thompson, and Donald R. Riley. Type selection of robot and gripper kinematic topology using expert systems. *International Journal of Robotics Research*, 5(2):183–189, 1986.

[49] Siamak G. Faal, Fuchen Chen, Weijia Tao, Mahdi Agheli, Shadi Tasdighikalat, and Cagdas D. Onal. Hierarchical kinematic design of foldable hexapedal locomotion platforms. *Journal of Mechanisms and Robotics*, 8(1):011005, 2016.

[50] Mohsen Falahi, Zhenishbek Zhakypov, Manan Shah, and Jamie Paik. The design and control of the multi-modal locomotion origami robot, Tribot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4349–4355, 2015.

[51] Zhun Fan, Xinye Cai, Wenji Li, Huibiao Lin, Shuxiang Xie, and Sheng Wang. Evolutionary synthesis of dynamical systems: The past, current, and future. In *Genetic and Evolutionary Computation Conference*, pages 1169–1174, 2014.

[52] Samuel Felton, Michael Tolley, Erik Demaine, Daniela Rus, and Robert Wood. A method for building self-folding machines. *Science*, 345(6197):644–646, 2014.

[53] Samuel M. Felton, Michael T. Tolley, Cagdas D. Onal, Daniela Rus, and Robert J. Wood. Towards autonomous self-folding: A printed inchworm robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[54] Amir Firouzeh and Jamie Paik. Robogami: A fully integrated low-profile robotic origami. *Journal of Mechanisms and Robotics*, 7(2):021009, 2015.

[55] fischertechnik GmbH, 2016. `http://www.fischertechnik.de`.

[56] Mark Fuge, Greg Carmean, Jessica Cornelius, and Ryan Elder. The MechProcessor: Helping novices design printable mechanisms across different printers. *Journal of Mechanical Design*, 137(11):111415, 2015.

[57] Thomas A. Funkhouser, Michael M. Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David P. Dobkin. Modeling by example. *ACM Transactions on Graphics*, 23(3):652–663, 2004.

[58] Joshua B. Gafford, Samuel B. Kesner, Robert J. Wood, and Conor J. Walsh. Microsurgical devices by pop-up book MEMS. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*, pages DETC2013–13086, 2013.

[59] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics*, 28(3):33, 2009.

[60] Wei Gao, Karthik Ramani, Raymond J. Cipra, and Thomas Siegmund. Kinetogami: A reconfigurable, combinatorial, and printable sheet folding. *Journal of Mechanical Design*, 135(11):111009, 2013.

[61] Wei Gao, Yunbo Zhang, Devarajan Ramanujan, Karthik Ramani, Yong Chen, Christopher B. Williams, Charlie C. L. Wang, Yung C. Shin, Song Zhang, and Pablo D. Zavattier. The status, challenges, and future of additive manufacturing in engineering. *Computer-Aided Design*, 69:65–89, 2015.

[62] Akash Garg, Alec Jacobson, and Eitan Grinspun. Computational design of reconfigurables. *ACM Transactions on Graphics*, 35(4):90, 2016.

[63] Andrew T. Gaynor, Nicholas A. Meisel, Christopher B. Williams, and James K. Guest. Multiple-material topology optimization of compliant mechanisms created via polyjet three-dimensional printing. *ASME Journal of Manufacturing Science and Engineering*, 136(6):061015, 2014.

[64] Qi Ge, Conner K. Dunn, H. Jerry Qi, and Martin L. Dunn. Active origami by 4D printing. *Smart Materials and Structures*, 23(9):094007, 2014.

[65] Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2485–2492, 2010.

[66] Jiangtao Gong, Jue Wang, and Yingqing Xu. PaperLego: Component-based papercraft designing tool for children. In *Proceedings of ACM SIGGRAPH Asia*, page 3, 2014.

[67] Steven Gray, Nathan Zeichner, Mark Yim, and Vijay Kumar. A simulator for origami-inspired self-reconfigurable robots. In *Origami 5: Proceedings of the 5th International Conference of Origami Science, Mathematics, and Education*, pages 323–333, 2011.

[68] The LEGO Group. Mindstorms, 2016. `http://mindstorms.lego.com`.

[69] Jingjiao Guan, Hongyan He, Derek J. Hansford, and L. James Lee. Self-folding of three-dimensional hydrogel microstructures. *Journal of Physical Chemistry B*, 109(49):23134–23137, 2005.

[70] Satyandra K. Gupta and Dana S. Nau. Systematic approach to analysing the manufacturability of machined parts. *Computer-Aided Design*, 27(5):323–342, 1995.

[71] Elliot Hawkes, Byoungkwon An, Nadia M. Benbernou, H. Tanaka, S. Kim, Erik D. Demaine, Daniela Rus, and Robert J. Wood. Programmable matter by folding. *Proceedings of the National Academy of Sciences*, 107(28):12441–12445, 2010.

[72] Jonathan Hiller and Hod Lipson. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466, 2012.

[73] Aaron M. Hoover and Ronald S. Fearing. Fast scale prototyping for folded millirobots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1777–1778, 2008.

[74] Aaron M. Hoover, Erik Steltz, and Ronald S. Fearing. RoACH: An autonomous 2.4g crawling hexapod robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 26–33, 2008.

[75] Gregory S. Hornby and Jordan B. Pollack. Body-brain co-evolution using L-systems as a generative encoding. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, pages 868–875, 2001.

[76] Larry L. Howell. *Compliant mechanisms*. John Wiley & Sons, New York, NY, USA, 2001.

[77] Larry L. Howell, A. Midha, and T. W. Norton. Evaluation of equivalent spring stiffness for use in a pseudo-rigid-body model of large-deflection compliant mechanisms. *Journal of Mechanical Design*, 118(1):126–131, 1996.

[78] Pu Huang, Dongping Deng, and Yong Chen. Modeling and fabrication of heterogeneous three-dimensional objects based on additive manufacturing. In *Proceedings of the ASME International Mechanical Engineering Congress and Exposition*, pages IMECE2013–65724, 2013.

[79] Bart Huthwaite. Designing in quality. *Quality*, 27(11):34, 1988.

[80] Tetsuo Ida, Hidekazu Takahashi, Mircea Marin, Asem Kasem, and Fadoua Ghourabi. Computational origami system Eos. *Origami 4: Proceedings of the 4th International Meeting of Origami Science, Mathematics, and Education*, pages 285–293, 2009.

[81] iRobot. Roomba, 2016. `http://www.irobot.com/For-the-Home/Vacuuming/Roomba.aspx`.

[82] Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. An end-to-end system for accomplishing tasks with modular robots. In *Proceedings of the Robotics: Science and Systems Conference*, 2016.

[83] Steven G. Johnson. The NLopt nonlinear-optimization package, 2014. `http://ab-initio.mit.edu/nlopt`.

[84] Jack W. Judy and Richard S. Muller. Magnetically actuated, addressable microstructures. *Journal of Microelectromechanical Systems*, 6(3):249–256, 1997.

[85] Alfred Bray Kempe. On a general method of describing plane curves of the nth degree by linkwork. *Proceedings of the London Mathematical Society*, s1-7(1):213–216, 1875.

[86] Jongwoo Kim, Dae-Young Lee, Sa-Reum Kim, and Kyu-Jin Cho. A self-deployable origami structure with locking mechanism induced by buckling effect. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3166–3171, 2015.

[87] Henry C. King. Planar linkages and algebraic sets. *Turkish Journal of Mathematics*, 23(1):33–56, 1999.

[88] Hideo Kodama. Automatic method for fabricating a three-dimensional plastic model with photo-hardening polymer. *Review of Scientific Instruments*, 52(11):1770–1773, 1981.

[89] Je-Sung Koh and Kyu-Jin Cho. Omega-shaped inchworm-inspired crawling robot with large-index-and-pitch (lip) sma spring actuators. *IEEE/ASME Transactions on Mechatronics*, 18(2):419–429, 2013.

[90] Je-Sung Koh, Sa-Reum Kim, and Kyu-Jin Cho. Self-folding origami using torsion shape memory alloy wire actuators. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*, pages DETC2014–34822, 2014.

[91] Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J. Mitra. Creating works-like prototypes of mechanical objects. *ACM Transactions on Graphics*, 33(6):217, 2014.

[92] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[93] Jason S. Ku and Erik D. Demaine. Folding flat crease patterns with thick materials. *Journal of Mechanisms and Robotics*, 8(3):031003, 2016.

[94] Robert J. Lang. A computational algorithm for origami design. In *Proceedings of the Annual Symposium on Computational Geometry*, pages 98–105, 1996.

[95] Robert J. Lang. *Origami Design Secrets: Mathematical Methods for an Ancient Art*. CRC Press, Boca Raton, FL, USA, 2001.

[96] Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. Converting 3D furniture models to fabricatable parts and connectors. *ACM Transactions on Graphics*, 30(4):85, 2011.

[97] DaeYoung Lee, JiSuk Kim, SaReum Kim, JeSung Koh, and KyuJin Cho. The deformable wheel robot using magic-ball origami structure. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*, pages DETC2013–13016, 2013.

[98] Xian-Ying Li, Chao-Hui Shen, Shi-Sheng Huang, Tao Ju, and Shi-Min Hu. Popup: Automatic paper architectures from 3D models. *ACM Transactions on Graphics*, 29(4):111, 2010.

[99] Hod Lipson and Jordan B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[100] Kui-Yip Lo, Chi-Wing Fu, and Hongwei Li. 3D polyomino puzzle. *ACM Transactions on Graphics*, 28(5):157, 2009.

[101] Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. Chopper: Partitioning models into 3D-printable parts. *ACM Transactions on Graphics*, 31(6):129, 2012.

[102] Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. Legolization: Optimizing lego designs. *ACM Transactions on Graphics*, 34(6):222, 2015.

[103] Xiaotian Ma, Dana Vogtmann, and Sarah Bergbreiter. Dynamics and scaling of magnetically folding multi-material structures. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1899–1906, 2016.

[104] Robert MacCurdy, Robert Katzschmann, Youbin Kim, and Daniela Rus. Printable hydraulics: A method for fabricating robots by 3D co-printing solids and liquids. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3878–3885, 2016.

[105] Robert MacCurdy, Jeffrey Lipton, Shuguang Li, and Daniela Rus. Printable programmable viscoelastic materials for robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[106] Robert MacCurdy, Anthony McNicoll, and Hod Lipson. Bitblox: Printable digital materials for electromechanical machines. *International Journal of Robotics Research*, 33(10):1342–1360, 2014.

[107] Constantinos Mavroidis, Kathryn J. DeLaurentis, Jey Won, and Munshi Alam. Fabrication of non-assembly mechanisms and robotic systems using rapid prototyping. *Journal of Mechanical Design*, pages 516–524, 2000.

[108] Duncan McCallum and David Avis. A linear algorithm for finding the convex hull of a simple polygon. *Information Processing Letters*, 9(5):201–206, 1979.

[109] James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. A compiler for 3D machine knitting. *ACM Transactions on Graphics*, 35(4):49, 2016.

[110] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3D-printable robotic creatures. *ACM Transactions on Graphics*, 34(6):216, 2015.

[111] Ankur Mehta, Nicola Bezzo, Byoungkwon An, Peter Gebhard, Vijay Kumar, Insup Lee, and Daniela Rus. A design environment for the rapid specification and fabrication of printable robots. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2014.

[112] Ankur Mehta, Joseph DelPreto, Kai Weng Wong, Scott Hamill, Hadas Kress-Gazit, and Daniela Rus. Robot creation from functional specifications. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2015.

[113] Ankur Mehta and Daniela Rus. An end-to-end system for designing mechanical structures for print-and-fold robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[114] Ankur M. Mehta, Joseph DelPreto, Benjamin Shaya, and Daniela Rus. Cogeneration of mechanical, electrical, and software designs for printable robots from structural specifications. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2892–2897, 2014.

[115] Avraham A. Melkman. On-line construction of the convex hull of a simple polyline. *Information Processing Letters*, 25(1):11–12, 1987.

[116] Yan Meng, Yuyang Zhang, Abhay Sampath, Yaochu Jin, and Bernhard Sendhoff. Cross-Ball: A new morphogenetic self-reconfigurable modular robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 267–272, 2011.

[117] Jun Mitani. A design method for 3D origami based on rotational sweep. *Computer-Aided Design and Applications*, 6(1):69–79, 2009.

[118] Jun Mitani. A design method for axisymmetric curved origami with triangular prism protrusions. In *Origami 5: Proceedings of the 5th International Meeting of Origami Science, Mathematics, and Education*, 2011.

[119] Jun Mitani and Hiromasa Suzuki. Computer aided design for origamic architecture models with polygonal representation. In *Proceedings of Computer Graphics International*, pages 93–99, 2004.

[120] Jun Mitani and Hiromasa Suzuki. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Transactions on Graphics*, 23(3):259–263, 2004.

[121] Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. Illustrating how mechanical assemblies work. *ACM Transactions on Graphics*, 29(4):58, 2010.

[122] Shuhei Miyashita, Laura Meeker, Michael T. Tolley, Robert J. Wood, and Daniela Rus. Self-folding miniature elastic electric devices. *Smart Materials and Structures*, 23(9):094005, 2014.

[123] Yuki Mori and Takeo Igarashi. Plushie: An interactive design system for plush toys. *ACM Transactions on Graphics*, 26(3):45, 2007.

[124] Stefanie Mueller, Bastian Kruck, and Patrick Baudisch. LaserOrigami: laser-cutting 3D objects. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 2585–2592, 2013.

[125] Stefanie Mueller, Tobias Mohr, Kerstin Guenther, Johannes Frohnhofen, and Patrick Baudisch. faBrickation: Fast 3D printing of functional objects by integrating construction kit building blocks. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 3827–3834, 2014.

[126] Tamal Mukherjee and Gary K Fedder. Hierarchical mixed-domain circuit simulation, synthesis and extraction methodology for mems. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 21(3):233–249, 1999.

[127] Yash Mulgaonkar, Brandon Araki, Je-sung Koh, Luis Guerrero-Bonilla, Daniel M. Aukes, Anurag Makineni, Michael T. Tolley, Daniela Rus, Robert J. Wood, and Vijay Kumar. The flying monkey: A mesoscale robot that can run, fly, and grasp. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4672–4679, 2016.

[128] Przemyslaw Musialski, Thomas Auzinger, Michael Birsak, Michael Wimmer, and Leif Kobbelt. Reduced-order shape optimization using offset surfaces. *ACM Transactions on Graphics*, 34(4):102, 2015.

[129] Ryuma Niiyama, Daniela Rus, and Sangbae Kim. Pouch motors: Printable/inflatable soft actuators for robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 6332–6337, 2014.

[130] Tadashi Odashima, Zhiwei Luo, and Shigeyuki Hosoe. Hierarchical control structure of a multilegged robot for environmental adaptive locomotion. *Artificial Life and Robotics*, 6:44–51, 2002.

[131] H. Okuzaki, T. Saido, H. Suzuki, Y. Hara, and H. Yan. A biomorphic origami actuator fabricated by folding a conducting paper. *Journal of Physics: Conference Series*, 127(1):012001, 2008.

[132] Cagdas D. Onal, Robert J. Wood, and Daniela Rus. An origami-inspired approach to worm robots. *IEEE/ASME Transactions on Mechatronics*, 18(2):430–438, 2013.

[133] Johannes T. B. Overvelde, Twan A. de Jong, Yanina Shevchenko, Sergio A. Becerra, George M. Whitesides, James C. Weaver, Chuck Hoberman, and Katia Bertoldi. A three-dimensional actuated origami-inspired transformable metamaterial with multiple degrees of freedom. *Nature Communications*, 7:10929, 2016.

[134] Jahng-Hyon Park and Haruhiko Asada. Concurrent design optimization of mechanical structure and control for high speed robots. *Journal of Dynamic Systems, Measurement, and Control*, 116(3):344–356, 1994.

[135] Jay Patel and Matthew I. Campbell. An approach to automate concept generation of sheet metal parts based on manufacturing operations. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*, pages DETC2008–49648, 2008.

[136] Jay Patel and Matthew I. Campbell. An approach to automate and optimize concept generation of sheet metal parts by topological and parametric decoupling. *Journal of Mechanical Design*, 132(5):051001, 2010.

[137] Edwin A. Peraza-Hernandez, Darren J. Hartl, Richard J. Malak Jr., and Dimitris C. Lagoudas. Origami-inspired active structures: A synthesis and review. *Smart Materials and Structures*, 23(9):094001, 2014.

[138] Pololu Corporation, 2016. `http://www.pololu.com/`.

[139] Michael J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. *Advances in Optimization and Numerical Analysis*, pages 51–67, 1994.

[140] Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. Make it stand: Balancing shapes for 3D fabrication. *ACM Transactions on Graphics*, 32(4):81, 2013.

[141] Chen Qiu, Vahid Aminzadeh, and Jian S. Dai. Kinematic analysis and stiffness validation of origami cartons. *Journal of Mechanical Design*, 135(11):111004, 2013.

[142] Hayes Solos Raffle, Amanda J. Parkes, and Hiroshi Ishii. Topobo: A constructive assembly system with kinetic memory. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 647–654, 2004.

[143] Raf Ramakers, Kashyap Todi, and Kris Luyten. PaperPulse: An integrated approach for embedding electronics in paper designs. In *Proceedings of the*

ACM Conference on Human Factors in Computing Systems, pages 2457–2466, 2015.

[144] John W. Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus. 3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1925–1932, 2015.

[145] John W. Romanishin, Kyle Gilpin, and Daniela Rus. M-blocks: Momentum-driven, magnetic modular robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4288–4295, 2013.

[146] Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[147] Lindsay Sanneman, Deborah Ajilo, Joseph DelPreto, Ankur Mehta, Shuhei Miyashita, Negin Abdolrahim Poorheravi, Cami Ramirez, Sehyuk Yim, Sangbae Kim, and Daniela Rus. A distributed robot garden system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 6120–6127, 2015.

[148] sarah-marie belcastro and Thomas C. Hull. Modelling the folding of paper into three dimensions using affine transformations. *Linear Algebra and its Applications*, 348(1):273–282, 2002.

[149] Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. SketchChair: An all-in-one chair design system for end users. In *Proceedings of the International Conference on Tangible, Embedded, and Embodied Interaction*, pages 73–80, 2011.

[150] Wolfram Schlickenrieder. *Nets of Polyhedra*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 1997.

[151] Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. Analytic drawing of 3D scaffolds. *ACM Transactions on Graphics*, 28(5):149, 2009.

[152] Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. Design and fabrication by example. *ACM Transactions on Graphics*, 33(4):62, 2014.

[153] Adriana Schulz, Cynthia Sung, Andrew Spielberg, Wei Zhao, Yu Cheng, Ankur Mehta, Eitan Grinspun, Daniela Rus, and Wojciech Matusik. Interactive robogami: Data-driven design for 3D print-and-fold robots with ground locomotion. In *ACM SIGGRAPH Talks*, 2015.

[154] Hiroki Shigemune, Shingo Maeda, Yusuke Hara, and Shuji Hashimoto. Design of paper mechatronics: Towards a fully printed robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 536–541, 2014.

[155] Hiroki Shigemune, Shingo Maeda, Yusuke Hara, Uori Koike, and Shuji Hashimoto. Kirigami robot: Making paper robot using desktop cutting plotter and inkjet printer. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1091–1096, 2015.

[156] ByungHyun Shin, Samuel M. Felton, Michael T. Tolley, and Robert J. Wood. Self-assembling sensors for printable machines. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4417–4422, 2014.

[157] Adam C. Siegel, Scott T. Phillips, Michael D. Dickey, Nanshu Lu, Zhigang Suo, and George M. Whitesides. Foldable printed circuit boards on paper substrates. *Advanced Functional Materials*, 20(1):28–35, 2009.

[158] Karl Sims. Evolving 3D morphology and behavior by competition. *Artificial Life*, 1(4):353–372, 1994.

[159] Karl Sims. Evolving virtual creatures. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 15–22, 1994.

[160] Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. Designing inflatable structures. *ACM Transactions on Graphics*, 33(4):63, 2014.

[161] Daniel E. Soltero, Brian J. Julian, Cagdas D. Onal, and Daniela Rus. A lightweight modular 12-DOF print-and-fold hexapod. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1465–1471, 2013.

[162] Seungmoon Song, Joohyung Kim, and Katsu Yamane. Development of a bipedal robot that walks like an animation character. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3596–3602, 2015.

[163] A. H. Soni, Mohammad H. F. Dado, and Yicheng Weng. An automated procedure for intelligent mechanism selection and dimensional synthesis. *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, 110(2):130–137, 1988.

[164] Alexander Spröwitz, Rico Moeckel, Massimo Vespignani, Stéphane Bonardi, and Auke Jan Ijspeert. Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot. *Robotics and Autonomous Systems*, 62(7):1016–1033, 2014.

[165] Kasper Støy, David Brandt, and David J. Christensen. *Self-Reconfigurable Robots*. The MIT Press, Cambridge, MA, USA, 2010.

[166] Timothy Sun and Changxi Zheng. Computational design of twisty joints and puzzles. *ACM Transactions on Graphics*, 34(4):101, 2015.

[167] Xu Sun, Samuel M. Felton, Robert J. Wood, and Sangbae Kim. Printing angle sensors for foldable robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1725–1731, 2015.

[168] Cynthia Sung, Erik D. Demaine, Martin L. Demaine, and Daniela Rus. Edge-compositions of 3D surfaces. *ASME Journal of Mechanical Design*, 135(11):111001, 2013.

[169] Cynthia Sung, Erik D. Demaine, Martin L. Demaine, and Daniela Rus. Joining unfoldings of 3-D surfaces. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*, pages DETC2013–12692, 2013.

[170] Cynthia Sung and Daniela Rus. Foldable joints for foldable robots. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2014.

[171] Cynthia Sung and Daniela Rus. Automated fabrication of foldable robots using thick materials. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2015.

[172] Cynthia Sung and Daniela Rus. Foldable joints for foldable robots. *ASME Journal of Mechanisms and Robotics*, 7(2):021012, 2015.

[173] Tomohiro Tachi. 3D origami design based on tucking molecule. In *Origami 4: Proceedings of the 4th International Meeting of Origami Science, Mathematics, and Education*, 2006.

[174] Tomohiro Tachi. Origamizing polyhedral surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):298–311, 2010.

[175] Tomohiro Tachi. Rigid-foldable thick origami. *Origami 5: Proceedings of the 5th International Meeting of Origami Science, Mathematics, and Education*, pages 253–264, 2011.

[176] Tomohiro Tachi and Koryo Miura. Rigid-foldable cylinders and cells. *Journal of the International Association for Shell and Spatial Structures (IASS)*, 53(4):217–226, 2012.

[177] Tama Software Ltd. Pepakura Designer, 2016. `http://www.tamasoft.co.jp/pepakura-en`.

[178] Kenta Tanaka, Yusuke Kamotani, and Yasuyoshi Yokokohji. Origami folding by a robotic hand. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2540–2547, 2007.

[179] Henry E. Theis. *Handbook of Metalforming Processes*. CRC Press, New York, NY, USA, 1999.

[180] Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. Computational design of linkage-based characters. *ACM Transactions on Graphics*, 33(4):64, 2014.

[181] Emanuel Todorov, Weiwei Li, and Xiuchuan Pan. From task parameters to motor synergies: A hierarchical framework for approximately optimal control of redundant manipulators. *Journal of Robotic Systems*, 22(11):691–710, 2005.

[182] Michael T. Tolley, Samuel M. Felton, Shuhei Miyashita, Daniel Aukes, Daniela Rus, and Robert J. Wood. Self-folding origami: Shape memory composites activated by uniform heating. *Smart Materials and Structures*, 23(9):094006, 2014.

[183] Michael T. Tolley, Samuel M. Felton, Shuhei Miyashita, Lily Xu, ByungHyun Shin, Monica Zhou, Daniela Rus, and Robert J. Wood. Self-folding shape memory laminates for automated fabrication. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4931–4936, 2013.

[184] Nicholas Turner, Bill Goodwine, and Mihir Sen. A review of origami applications in mechanical engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 230(14):2345–2362, 2016.

[185] Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Transactions on Graphics*, 31(4):86, 2012.

[186] Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive couture for interactive garment modeling and editing. *ACM Transactions on Graphics*, 30(4):90, 2011.

[187] Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Transactions on Graphics*, 33(4):65, 2014.

[188] Carlos A. Vanegas, Daniel G. Aliaga, Bedrich Benes, and Paul A. Waddell. Interactive design of urban spaces using geometrical and behavioral modeling. *ACM Transactions on Graphics*, 28(5):111, 2009.

[189] Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik. OpenFab: A programmable pipeline for multi-material fabrication. *ACM Transactions on Graphics*, 32(4):136, 2013.

[190] Etienne Vouga, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. Design of self-supporting surfaces. *ACM Transactions on Graphics*, 31(4), 2012.

[191] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Transactions on Graphics*, 28(3):60, 2009.

[192] Cheng-Hua Wang. *Manufacturability-Driven Decomposition of Sheet Metal Products*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 1997.

[193] Allen C. Ward. Mechanical design compilers. In Erik K. Antonsson and Jonathan Cagan, editors, *Formal Engineering Design Synthesis*. Cambridge University Press, Cambridge, UK, 2001.

[194] Allen C. Ward and Warren P. Seering. Quantitative inference in a mechanical design compiler. *Journal of Mechanical Design*, 115(1):29–35, 1993.

[195] Paul J. White and Mark Yim. Reliable external actuation for full reachability in robotic modular self-reconfiguration. *International Journal of Robotics Research*, 29(5):598–612, 2010.

[196] Emily Whiting, Hijung Shin, Robert Wang, John Ochsendorf, and Frédo Durand. Structural optimization of 3D masonry buildings. *ACM Transactions on Graphics*, 31(6):159, 2012.

[197] John P. Whitney, Pratheev S. Sreetharan, Kevin Y. Ma, and Robert J. Wood. Pop-up book MEMS. *Journal of Micromechanics and Microengineering*, 21(11):115021, 2011.

[198] Eric W. Wilcox, Adam Shrager, Landen Bowen, Mary Frecker, Paris Von Lockette, Timothy Simpson, Spencer Magleby, Robert J. Lang, and Larry L. Howell. Considering mechanical advantage in the design and actuation of an origami-based mechanism. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE)*, pages DETC2015–47708, 2015.

[199] Brian G. Winder, Spencer P. Magleby, and Larry L. Howell. Kinematic representations of pop-up paper mechanisms. *Journal of Mechanisms and Robotics*, 1(2):021009, 2009.

[200] Hongyi Xu, Yijing Li, Yong Chen, and Jernej Barbič. Interactive material design using model reduction. *ACM Transactions on Graphics*, 34(2):18, 2015.

[201] Xue Yan and P. Gu. A review of rapid prototyping technologies and systems. *Computer-Aided Design*, 28(4):307–318, 1996.

[202] Zheng Yan, Fan Zhang, Jiechen Wang, Fei Liu, Xuelin Guo, Kewang Nan, Qing Lin, Mingye Gao, Dongqing Xiao, Yan Shi, Yitao Qiu, Haiwen Luan, Jung Hwan Kim, Yiqi Wang, Hongying Luo, Mengdi Han, Yonggang Huang, Yihui Zhang, and John A. Rogers. Controlled mechanical buckling for origami-inspired construction of 3D microstructures in advanced materials. *Advanced Functional Materials*, 26(16):2629–2639, 2016.

[203] Kentaro Yasu. MOR4R: Microwave oven recipes for resins. In *ACM SIGGRAPH Talks*, 2015.

[204] Yong W. Yi and Chang Liu. Magnetic actuation of hinged microstructures. *Journal of Microelectromechanical Systems*, 8(1):10–17, 1999.

[205] Sehyuk Yim and Sangbae Kim. Origami-inspired printable telemicromanipulation system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2704–2709, 2015.

[206] Yahan Zhou, Shinjiro Sueda, Wojciech Matusik, and Ariel Shamir. Boxelization: Folding 3D objects into boxes. *ACM Transactions on Graphics*, 33(4):71, 2014.

[207] Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided mechanical toy modeling. *ACM Transactions on Graphics*, 31(6):127, 2012.

[208] Shannon A. Zirbel, Robert J. Lang, Mark W. Thomson, Deborah A. Sigel, Phillip E. Walkemeyer, Brian P. Trease, Spencer P. Magleby, and Larry L. Howell. Accommodating thickness in origami-based deployable arrays. *Journal of Mechanical Design*, 135(11):111005, 2013.