Robot Design for Everyone: Computational Tools that Democratize the Design of Robots

Ruta Parimal Desai

CMU-RI-TR-18-46
July 2018

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Stelian Coros (Co-chair)
James McCann (Co-chair)
Scott Hudson
Tovi Grossman, University of Toronto

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: Automatic robot design, design tools, computational design, user-in-the-loop design, simulation-based design, data-driven design, design optimization, digital fabrication, Human-Computer Interaction (HCI), mixed-initiative design, democratized design, ubiquitous robots, DIY, user interfaces.

For my amazing family, because without you – nothing is meaningful.

Abstract

A grand vision in robotics is that of a future wherein robots are integrated in daily human life just as smart phones and computers are today. Such pervasive integration of robots would require faster design and manufacturing of robots that cater to individual needs. For instance, people would be able to obtain customized smart assistant devices such as a home monitor personalized with their child's favorite fantasy character, a helping hand robot for their specific art project, or a cleaning robot that can clean hard to access spaces of their homes. However, robots of today take years to be created by experts, and are often not customizable. To enable widespread integration of robots, we wish to democratize the robot design process in order to support rapid creation of custom robots for the people and by the people.

In recent years, advances in digital fabrication technologies and the availability of affordable electronics such as Rasberry Pi, Arduino etc. is enabling rapid creation of smart devices. Unfortunately, currently these technologies are only accessible to experts because of the skills and domain knowledge needed to build with them. To change this status quo, we present a suite of easy-to-use computational tools for enabling the design of a broad class of robotic devices that casual users might want to create.

In particular, we develop tools for designing physical structure and task-specific behavior of robotic devices with various form factors and functionalities. The key strengths of our tools include intuitive visual interfaces that can support user-in-the-loop interactive design, parameterized domain-specific models and physics-based simulation that can encode complex design aspects, and efficient algorithms that can search high-dimensional design spaces at interactive rates for user-preferred solutions. Our tools also automate tedious design steps allowing users to focus on the creative aspects of the design process. Finally, we show how simulation-based feedback and data-driven design can be used to lower the barriers to entry for casual users. We validate our tools by fabricating various prototypes and by conducting user-studies with novices.

In the past, design tools such as those offered by Adobe creative suite and Autodesk have revolutionized the creation of diverse digital content ranging from images to animation. The tools presented in this thesis take a step towards enabling the same for the domain of robotics.

Acknowledgments

Like any PhD journey, mine has been a bumpy ride. I switched advisors, changed research fields, and even considered quitting at one point. But here I am writing acknowledgments and preparing to wind up this seven year long phase of my life. I still find it hard to believe that I made it. And so here is a toast to the heros of this journey – the many mentors, family, and friends, who truly made it possible.

I am thankful to my advisors, Stelian and Jim (James) for always being patient and supportive towards my crazy ideas, and for teaching me the many essentials of doing research. The hardest part of solving a big problem is formulating the first step, by breaking down the problem into small testable blocks. Stelian always pushed me to systematically break down a huge problem and to identify the relevant core. Much of my research also involved building complex software systems. Inspite of zero background, I could dive right into it thanks to Stelian. I am also grateful to Stelian for teaching me the art of debugging. Jim taught me how to ask the right questions and how to leverage one's strengths when solutions seemed impossible to find. The last phase of a PhD is always so full of self-doubt and stress, and I wouldn't have made it across the finish line had it not been for Jim. Thank you Jim for your encouragement, kindness, invaluable advice on life and jobs, and cookies before paper deadlines. I wish I could've been your mentee for longer, and hope we would someday find time to make crazy-shaped water fountains together.

Before developing computational tools, I spent the initial years of my PhD working with Hartmut and Jessica on motor-skill learning. I wouldn't have been doing research at all, if not for Hartmut. Hartmut taught me what it meant to be truly curious and passionate about a problem, how to thoroughly evaluate one's research approach, and how to be an honest researcher. I would be forever grateful to Jessica and Hartmut for letting me seek my own problems, even when it meant switching advisors. I am also thankful to Jessica, Hartmut, Stelian, and Jim for helping me write and communicate my research better, and for putting in many hours to proof-read my papers. Finally, many thanks to Scott for taking out time to be on my committee and for valuable inputs on my thesis, and to Katharina Muelling and Margarita Safonova for their support in extending my work through collaborations with NREC.

The highest peak of my PhD was my internship with Autodesk Research in Toronto. I am indebted to Tovi for giving me this opportunity. My mentors at Autodesk – Fraser, Justin, and Tovi juggled beautifully between advising and collaborating, thereby enabling me to truly develop into an independent researcher. They also helped me make my way into the HCI community, which I am very grateful for. Thank you Fraser, Justin, and Tovi for putting up with me beyond the time of the internship, for supporting the many weird needs of the project, for teaching me user-study and crowdsurcing knowhow, for valuable inputs that shaped the research contributions, and for never giving up on the project. My time at Autodesk was also made memorable by wonderful, caring friends – Ruslana, Monika, John, Max, Jacobo, and Jenmy. Thanks also to Umang and Gullu for adventures in Toronto.

Beyond mentors, a major source of learning during PhD is lab-mates and peers. I am especially thankful to Ye and Beichen for working with me on some of my research projects. I have learnt a lot through our interactions. Many thanks to Chris for being my C++ and Blender teacher in the early days of working with Stelian, and to Vidya for always being there to discuss both research and life. Thanks Rohan, Nick, Evan, Ticha, Frances, Kai, Katherine, Ravi, and Jenny for helping with user-studies and paper proof-reads. Thanks Aayush and Gunnar for brainstorming with me when I was really stuck with research, and for your insights on the magic of deep learning. Thank you Seungmoon, JW, Akshara, and Abhijeet for research inspirations and discussions; and thank you Sanjiban for teaching me that research can be done anywhere, anytime.

I am also very thankful to the amazing members of CMU staff: Suzanne, Jean, and Jess Butterbaugh. Jess smoothed out all the logistics of lab purchases and travel like a wizard. Suzanne and Jean have always been there to provide hope and motivation. I will miss the warmth of their smiles and hugs, as well as the excursions for hot chocolate and meals.

When life fell apart, friends in Pittsburgh held me together. Bhavana, Meghana, and Anjali transformed from house-mates to elder sisters – always there to share both tears and laughters. Living with them has changed me into a better person, and I am forever indebted for their kindness, encouragement, and love. Sunayana, Kriti, and Rose were amazing sources of inspiration and advice – from research to fashion. Hobart girls gang: I will miss our delicious shared meals, our many celebrations – birthdays, bachelorettes, festivals; gardening, cooking, skating, making snowman, sledding, and even cleaning sessions. I was also fortunate to have crossed paths with many other wonderful people that became lifelong friends. Thank you Emmanuel, Kalyani, Vamshi, Vignesh, Ashwati, and Vishnu dev for memorable hiking and biking trips, and deep conversations. Thank you Vivek for being there through thick and thin, and for accepting me with all of my mess. Thanks Erle for sharing my moral dilemmas; thanks Aimon for your infectious cheerfulness, which was always uplifting; thanks Shinjini, Kristen, and Fatma for making long NSH days of those early years bearable and memorable; and thanks Karthik (SPEVK) and Jess Austin for your unconditional care. I owe it to Marynel for pulling me out of the dark pits of doubts and de-motivations with her never-fading excitement and passion. I will miss our brunches and long discussions on life and research. Finally, thank you Tessa for helping me pull through the last year of my PhD and for introducing me to birding.

Miles apart, friends in India and elsewhere have given me more love than I deserved – even when I missed sharing many milestones from their lives. Thank you Vihang, Nirzaree, Bakli, Vishal Shah, Swati (Paddu), Satarupa, Barman, Mayuresh, Rushi, Abhilash, Yatin, Varsha, and Pooja – for never giving up on me, for always pinging and calling me even when I didn't, and for trying to understand my woes from the academic world even when they were so alien to you. You knew when to yell at me and when to hug me – I can't thank you all enough for that. I also feel blessed for receiving immense love, support, and goodwill from my extended family – cousins, ba, uncles, aunts, nephews, and nieces in Australia, US, and India. Special thanks to Dinesh uncle, without whose support this journey would have never begun.

Lastly, this thesis is dedicated to the four pillars that support my life – my magical parents, Nisu, and Arun. I am not exaggerating when I say I have the best parents in the universe. Their understanding, wisdom, courage, love, and belief in me have given me the strength to go through the darkest of times. They have accepted my horrendous mistakes and unpredictable decisions, given me the wings to fly and explore, and inspired me to seek out adventures that I would have never imagined to be possible otherwise. I do not have the words to describe what Nisu is to me and I cannot imagine a life without him, let alone a PhD. There is nothing in my life that he can't fix with his superpowers of listening, contemplating, and you-tubing :) I wish I end up being at least half as kind, matured, calm, and patient, as him. Finally, Arun has been the never-ending source of energy that kept me going through the long years of PhD. He taught me the art of seeing beyond my biases, and that of rational thinking. He has worn many hats for me: a critical peer when I needed research feedback, a friend when I needed to cry and laugh, an awesome chef when I needed to be fed, and a partner-in-crime for life's many adventures and setbacks.

I am so blessed to have you all in my life, and am eternally grateful for sculpting me into the person I am today. Here is to many more years of evolving, learning, and venturing together!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Why make robotics accessible?

Roboticists envision a future where robots would be integrated in daily human life just as smart phones are today [144, 152] (Figure 1.1a). Such a future of ubiquitous robots is further highlighted in the 2018 call for academic research proposals by the National Science Foundation (NSF) under the National Robotics Initiative (NRI) [1]. This initiative envisages robots to be part of our homes and offices; assisting in hospitals, classrooms, and factories; helping to run farms and mines; and exploring in air, on land, under water, and in space. However, robots of today take years to produce. For example, engineers took five years to create the industrial robot Baxter [170], while Anki took three years to design entertaining personal robot Cozmo [14]. A pervasive integration of robotics in everyday life therefore demands design and fabrication tools that will ***speed up*** the creation of robots. Apart from a faster design cycle, the NRI proposal emphasizes on ***customizability*** of robots with respect to specific tasks, environments, or people; for widespread adoption of robots. The progression towards enabling customization in robotic systems has already started developing in industry, as evident in the recent commercial robotic systems from HEBI Robotics, and Modbots [94, 145]. However, true ubiquity of robots may not be possible without democratizing the robot design process wherein common people can customize robots based on their unique needs and preferences.

Why is such a ***democratization*** of the robot design process essential? The democratization of digital content design powered by Adobe creative suite [8], Autodesk tools [21] etc. has already led to a new generation of creators and designers. Beyond the digital realm, the Do-it-yourself (DIY) community [211] has been a source of many creative solutions such as on demand disaster relief [75], and customizable 3D printed prosthetics [70]. The DIY revolution is thus a testimony that allowing casual people – artists, tinkerers, and designers to build personalized robotic devices empowers them to solve their own problems. Bringing in the creativity and diversity of these casual makers into the design process is therefore important for creating truly novel and out-of-the-box robotic systems of tomorrow.

[1] National Robotics Initiative 2.0: Ubiquitous Collaborative Robots (NRI-2.0) – `https://www.nsf.gov/pubs/2018/nsf18518/nsf18518.htm`

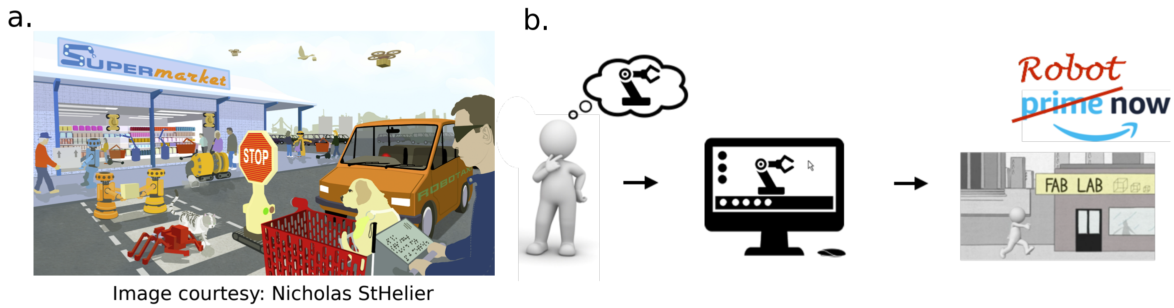


Figure 1.1: Thesis motivation – (a) In not so distant future, robots would be an integral part of our daily life. (b) In particular, people would be able to obtain personalized robotic devices fabricated either locally or online. To enable such a revolution, we need design tools that allow common people to create highly customized robots rapidly. In this thesis, we develop a suite of tools that enable such democratization of robot design.

1.2 The need for design tools

The rapid creation of custom robots for ubiquitous robotics demands the advancement of both hardware and software tools. Fortunately, the recent development of digital fabrication technologies such as 3D printing, laser-cutting, and affordable electronics such as Rasberry Pi [169], Arduino [2], Google's AIY kits [86] is already paving way for fast, on-demand manufacturing. From kinetic art to prototypes for academic research, makers have also been trying to embrace these recent technologies for creating novel robotic devices [3, 98]. While the hardware tools are becoming available, their use remains inaccessible and challenging for casual users because of the required skills and domain knowledge. With intuitive and powerful software design tools, we can change that. In this thesis, we think about what should these tools look like, and what should they do, in order to reduce the entry barrier of robotics (Figure 1.1b).

The goal of this thesis is thus to develop design tools that support casual users in building robots of different form-factors and functionalities.

We next provide an elaborate overview of the difficulties faced by casual users in designing custom robotic devices to motivate the constitution of the design tools in this thesis.

1.2.1 Challenges in the robot design process

We broadly define a robotic device as a functional object composed of various mechanical and electronic components that interact with each other to achieve higher level task goals. Designing such robotic devices is highly skill-intensive, tedious, and time-consuming. One needs to design the device's mechanical structure, as well as decide its functional capabilities and corresponding electro-mechanical components. One must then integrate these design elements together, and control them coherently to achieve a task. Each step in such a design process demands the knowledge of multiple disciplines and design principles. Furthermore, it is highly challenging to gauge how different elements of the design will behave together, and where will they fail. Therefore, designers typically iterate multiple times before converging onto their desired device design. Apart from being cost and time intensive, these iterations on hardware involve repeating many tedious design steps such as adjusting physical dimensions, processing collisions, generating connectors for components, re-assembly, tuning behavior control parameters and so on. Together, these challenges render the robot design and creation process to be highly daunting, and de-motivating especially for novices.

1.2.2 Barriers faced by casual users

Along with the challenges inherent to the design process, casual users such as artists, designers, students face additional challenges owing to the lack of technical skills, and the limitations of the environment they are building things in. Typically, these users operate in volunteer-run maker and hacker spaces, or classrooms, and are working intermittently on one-off projects. As a result, their learning is staggered, and unstructured. Further, failures are often, and hard to rectify. In particular, the following aspects amplify the barriers that casual users face:

1. *Limited resources:* Existing design tools have steep learning curves (ranging in weeks, even for simple designs) [91]. Further, hardware resources are either shared as in classrooms and maker-spaces, or are expensive. Furthermore, many of the casual users have limited ability to invest time in learning design tools, or invest monetary capital for multiple hardware iterations [35].
2. *Difficulty in asking for help owing to lack of context, and technical vocabulary:* When failures happen, casual users struggle with conveying their issues, and seeking relevant help [129].
3. *Dependence on peers for problem solving:* Finally, more often than not, these users have to rely on peers who might have similar technical backgrounds for rectifying failures [98].

1.3 Requirements for design tools

The design tools that can deal with the challenges discussed in Sec. 1.2, would therefore require the following:

1. encode multi-disciplinary domain knowledge
2. detect design failures, and reduce hardware iterations
3. automate tedious design aspects
4. enable intuitive usability so as to ensure smaller learning curve
5. aid user creativity

We achieve these requirements for our tools by adopting a set of design methodologies described next.

1.3.1 User-in-the-loop design

One approach to allow novices to build robotic devices is to develop fully automated tools, that work with minimal user inputs. Rather than such fully automated tools that allow for no control over the design process, we aspire to build user-in-the-loop tools, in order to truly encourage creativity in the design process. Apart from aiding creativity, such an approach eliminates the need for modeling complex user preferences such as aesthetics. Therefore, our tools automate tasks that are knowledge and skill-intensive as well as tedious, while letting users to be in charge of the overall process, and the creative decisions. In broader context, our approach is inspired by mixed-initiative interaction – wherein users and systems (e.g., an interface) take on complementary roles in a task, leading to an optimal collaboration with each being able to do what they do best [97].

To enable this, all our tools have an *intuitive visual interface* that allow users to specify their needs, and to be an integral part of the design process. We ensure that our interfaces have very small learning curves by using simple and familiar control operations such as drag and drop. We provide interaction modalities for users to interact with the automated algorithms running underneath our tools, and to guide these algorithms towards their desired outcomes by providing constant or intermittent feedback during the design process. Allowing users to play an active role in creating robotic devices for personal use has been shown to be essential for not only their sense of self-agency, but also for enhancing the quality of their interactions with these devices [203].

1.3.2 Feedback during design

Instead of having only one-way feedback that goes from the user to the design tool, we aim to establish a two-way feedback mechanism wherein the tool also provides the user with useful feedback about their designs. We achieve this using either relevant *physical simulations*, or through *parametric or data-driven models*. By leveraging simulations and models our tools approximate and show to the users, the real-world behavior of their designs. We also take advantage of intuitive visual highlighting to convey design failures. Such feedback from the system not only aids user creativity and learning about the design process, but it also directly helps users in validating the function of their designs and understanding possible failures prior to any hardware assembly. This, in turn, reduces the hardware iterations needed to get the design right. Such feedback has been found to be highly essential for increasing the accessibility of digital fabrication tools in previous studies [35, 129].

1.3.3 Customizability and fabricability

Empowering people to build personalized robotic devices inherently demands for *customization*. Towards this, we leverage both digital fabrication and traditional mass-manufacturing. Digital fabrication allows necessary customization capability, while mass-manufactured, affordable and durable off-the-shelf components can support diverse needs of robotic devices. Our tools thus enable easy design with customized parts as well as off-the-shelf electromechanical components. As a testament of the variety of devices that can be created using such an approach, we design and build many diverse artifacts ranging from walking robots, to Internet-of-Things (IoT) devices using the tools in this thesis (see Figure 1.2).

To *ease the transition from a design in software, to a fabricated prototype*, our tools export 3D printable geometries of custom parts of the design for fabrication, when relevant. The underlying algorithms of our tools that generate these geometries further respect the constraints necessary for valid functioning of individual device components as well as the feasibility of hardware and

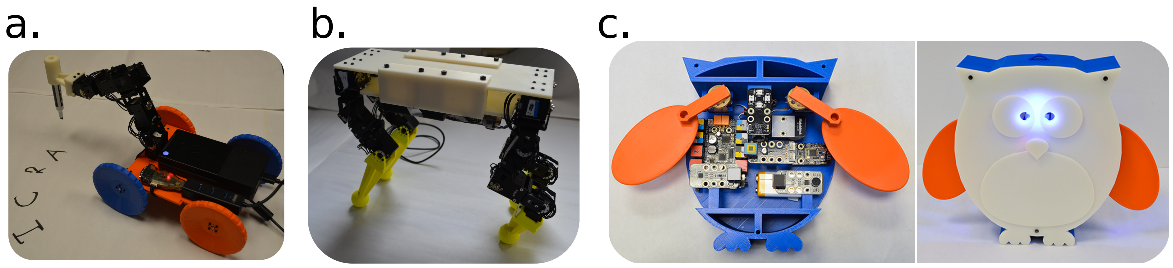


Figure 1.2: Various robotic devices fabricated using tools presented in this thesis (a) Robo-calligrapher: a robot car with a manipulator arm that draws, (b) Puppy: a quadrupedal robot, and (c) Chripy: a smart crib monitoring owl toy.

fabrication processes. For instance, we can account for fasteners holes, inter-connection between components, assembly constraints, 3D printer tolerances etc. Together, these steps ensure that the users are able to fabricate a robotic device with desired functionality in fewer hardware iterations.

1.3.4 User intent encapsulation

Our visual design interface allows users to specify their design requirements and preferences, as well as to provide and obtain feedback about their designs at runtime. Unfortunately, the ability of casual users to specify their needs for the device functionality, as well as to translate the feedback provided by the system into relevant design changes may be limited. Consequently, we explore the ideas of *semantic design* as well as *inverse design* for some of our tools. Defining the design requirements at high-level, using relevant context based semantics has been proven to be a powerful approach for image, and 3D model editing tools [43, 114, 163, 239]. We wish to bring these inspiring ideas to the domain of robot design.

1.3.5 Design space exploration

Exploring the *space of possibilities* and alternative courses of action for a task at hand has been found to be particularly important during the early, formative stages of design [55]. Computational tools can make available a large variety of appropriate solutions to the users at no additional cost. This ability has therefore been a key component in many computational creativity support tools [192]. Most of these tools either provide data-driven suggestions [42], or they present several alternative design suggestions generated using pre-defined rules [212, 213]. Towards our goal of aiding user creativity, we also explore both these approaches for enabling design space exploration in some of our tools. For instance, our semantic function design tool (Chapter 7) uses pre-computed dataset of designs to show alternatives to the users. On the other hand, our modular structure design tool (Chapter 4) generates alternative structure designs on the fly based on valid rules and user preferences.

1.4 Key contributions

Towards the requirements defined in Sec. 1.3, we present various tools for designing robotic devices – two structure design tools for articulated and non-articulated robots, a semantic function design tool for exploring and editing diverse behaviors for articulated robots intuitively, and two co-design tools that simultaneously accounts for an articulated robot's structure and behavior given a task. We define non-articulated robots as those with less than two joints in their structure such as the Internet of things (IoT) devices, while articulated robots such as manipulators and legged robots typically consist of more than or equal to two joints.

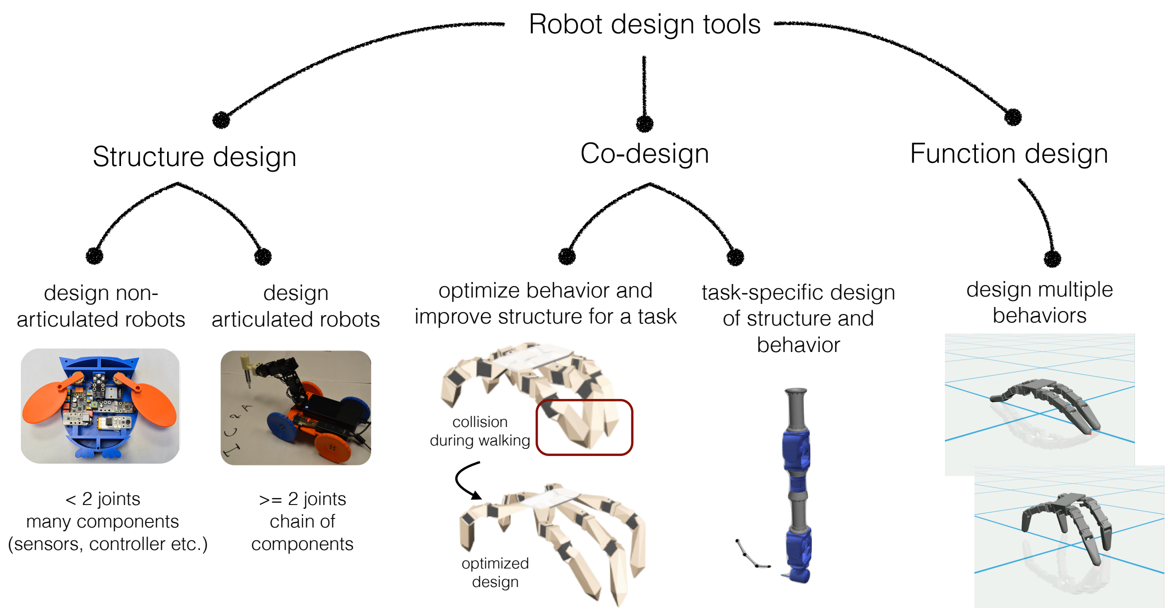


Figure 1.3: Overview of tools – This thesis presents a set of tools for structure and function design of robots. The structure design tools allow designing fabricable structures of non-articulated robots such as IoT devices and articulated robots such as manipulators and legged robots. We also devise function design and co-design tools for articulated robots. The function design tool allow users to explore and edit multiple behaviors for a given robot. While the co-design tools account for structure and function together.

Figure 1.3 gives an overview of these tools. Together these tools focus on design and editing of two major aspects of robotic devices – *structure* and *function*. Structure design entails the design of device's physical form, while function design focuses on achieving the resultant behavior or functionality of the device. For instance, designing the walking gait of a quadruped would pertain to function design, while structure design would involve the design of its articulated limbs. Our tools for function design only edit the device's function, assuming a fixed device structure. Similarly, structure design tools focus solely on the device's physical composition. To account for the inherent coupling between the device's structure and function, we also explore co-design approaches.

The tools developed in this thesis can be further categorized based on the level of details that the users need to provide to the tool for defining their design requirements (Table 1.1).

Table 1.1: Key contributions of this thesis

	Input from the user		
	Low-level	Mid-level	High-level
Tools in this thesis	Structure design of articulated robots Structure design of non-articulated robots	Co-optimize structure and motion of articulated robots	Semantic motion design of articulated robots Task-specific design of robotic arms

Even though we encode many nitty-gritty details of the design process, our structure design tools require certain low-level design details as inputs from the user such as electromechanical components needed in the robot, number and location of actuators, or the articulated structure morphology. As we developed more of our tools, we placed emphasis on on reducing the need of such low-level inputs from the user so as to make the design process even more accessible. For instance, our co-design tool only requires the number of actuators in a robot and not their locations. Moving further, our task-specific robot arm design system even eliminates this need of obtaining the number of actuators (degrees of freedom) in a robot from the users. Instead, the users can define the task using a motion trajectory, while our system automatically creates a corresponding robotic arm and its behavior. We also explore ways for the users to subjectively specify their desired designs with the help of semantic attributes in our semantic motion design tool. Note that, while in general fewer lower-level user-inputs ensure more accessibility, the type of inputs needed from the user are very dependent on the target audience as well as the intended application. For instance, in applications where aesthetics are important, its always preferable to let users be in charge of the creative decisions. Next, we give a brief overview of our tools.

1.5 Overview of devised tools

1.5.1 Structure design tools

Our structure design tools for articulated and non-articulated robots are built upon the observations about the inherent characteristics of these robots.

Assembly-aware design for non-articulated robots: Non-articulated robots typically consist of a physical casing or an enclosure that houses all the electro-mechanical components such as sensors, controllers, actuators, and batteries. Creating such devices therefore requires *packing and assembly* of all components within the enclosure, and involves the use of expert computer-aided design (CAD) tools such as Solidworks. Instead, our tool allow users without any CAD experience to create assembly-aware design of custom enclosures that would contain their desired components. Our tool is based upon a novel spatio-temporal model that captures the entire assembly process, including the placement of the components within the device, appropriate mounting structures and attachment strategies for these components, as well as a valid plan for assembling everything together.

Modular structure design for articulated robots: Articulated robots could also have various electromechanical components such as sensors, controllers etc. but their distinguishing feature is the chains of interconnected components with more than two joints. Based on this observation, we propose a generic computational abstraction that formalizes the creation of such *articulated structures from diverse modular components*. Further, unlike non-articulated robots, the resultant function of the robot is highly dependent on the structural arrangement of components, and is unintuitive. For instance, an improper arrangement of motors in a legged robot's limb may render the robot incapable of walking in a desired manner. We therefore integrate a physics simulation within the design tool. By leveraging continuous feedback from the simulation, users can iteratively improve their designs until individual needs and preferences are met.

1.5.2 Co-design of structure and function

Our structure design tools enable *forward design* of a robotic device wherein a user designs a device and then tests to see if it is desirable. Instead, *inverse design* systems start with user-specified design requirements, and can thus be very powerful tools for casual users. To enable such inverse design, we develop two co-design tools for articulated robots. Apart from inverse design, these tools also enable users to better explore the coupling between structure and function of the articulated robot, given a task. Both of our co-design tools focus on two different fabrication modalities and corresponding constraints.

Task-specific co-design of modular robotic arms: This tool focuses on co-design of structure and function of articulated robots with *discrete* off-the-shelf components with *fixed-sizes* such as brackets, connectors etc. In particular, we extend our modular articulated robot structure design tool to enable task-specific design of such robots, and demonstrate it for robotic arms.

Co-design of 3D printed legged robots: We also explore how co-design can be supported for articulated robots consisting of custom 3D printed parts, whose *physical dimensions can be tailored* to one's need. Specifically, we formulate a mathematical model that maps the morphological parameters of a robot to its motor capabilities. We then leverage this model to automatically improve user-designs for a given task.

1.5.3 Function design tools

Finally, we experiment with a semantic design tool to capture the user-intent about design functionality in an intuitive manner. We focus entirely on exploring the design space of a given robot's functionality, without modifying its structure.

Semantic motion design for articulated robots: We propose a functionality design tool for the domain of motion behavior design of articulated robots such as walking robots and robotic arms. We choose this domain because motion design of articulated robots is highly unintuitive and challenging for novices, and has a very high-dimensional parameter space effecting the resultant behavior. In particular, instead of using low-level and unintuitive parameters such as speed, pose etc. to describe the desired motion, an intuitive way would be to define the desired motion behavior characteristics through *perceptual attributes*. For instance, using attributes such as happy and angry one could easily describe two different variations of a robot's behavior. We thus enable the users to explore the design space of robot motions using the semantic space of expressions such as emotions. To this end, we present a data-driven framework that maps the semantic expression space and the robot motion parameter space. Using this mapping, we develop an intuitive user interface (UI) that supports easy editing and visual exploration of the space of diverse robot motions.

1.6 Thesis outline

We begin with a discussion on existing tools for robot design in Chapter 2. Following this, Chapters 3 and 4 describe the structure design tools – assembly-aware design of non-articulated robots and modular structure design for articulated robots respectively. Chapter 5 discusses task-specific design of robotic arms, while Chapter 6 explains our co-design approach for 3D printed legged robots. Finally, Chapter 7 discusses our work on semantic functionality design, before diving into the future work in Chapter 8. Apart from describing the model and the computational approach, each chapter also discusses relevant related work, corresponding evaluation strategy, and limitations in detail. Further, we clarify some of our design decisions, assumptions, and use cases for these tools within a frequently asked questions (FAQ) section in each chapter.

Chapter 2

Background

Designing robots involves the design of mechanical structure, electrical subsystems as well as that of robot's behavior or control. For the robots that we built, we used off-the-shelf electronic subsystems such as Arduino [2], owing to their affordability and accessibility. Thus, the tools developed in this thesis primarily focus on structure and behavior design (Figure 2.1). We therefore provide a brief overview of existing expert and novice design tools for structure and behavior of robots in this chapter. Since there is a plethora of available commercial, open-source as well as academically developed tools for robot design, our overview is by no means exhaustive. Instead, we provide a flavor of most relevant tools out there to emphasize the need for tools in this thesis. Related work specific to each tool is further discussed in individual chapters.

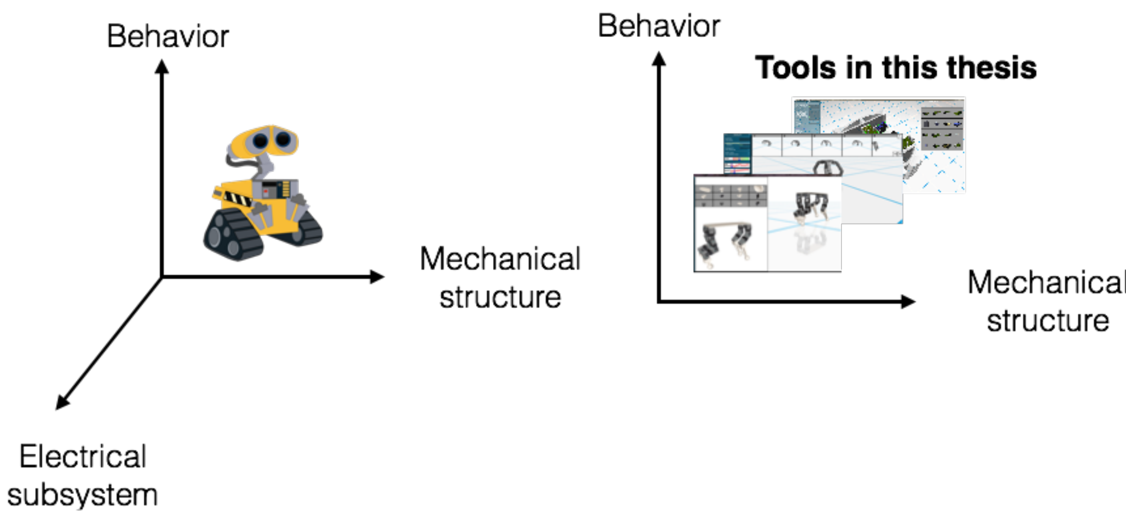


Figure 2.1: Robot design consists of mechanical structure, electrical subsystem, and behavior control design. Tools in this thesis focus on mechanical structure and behavior design of robots.

Figure 2.2 gives an overview of existing commercial and open-source structure and behavior design tools. We also discuss tools from academia at the end of the chapter.

2.1 Robot design tools for experts

The field of computer-aided design offers many tools to experts for mechanical structure design. Some of the prominent examples of these tools include Solidworks [196], Autodesk's Inventor [100] and Fusion 360 [7], PRO-E [74], Rhinoceros [173] etc. In order to be generic and to support structure design of a wide range of objects, these tools consist of a lot of features that are necessary from engineering design perspective such as 3D modeling, 3D assembly design, structure analysis etc. As a result, these tools typically have a steep learning curve [91]. Furthermore, many of them require an expensive license for use. While companies have started offering free license to students (e.g., [76]), hobbyists and artists still face challenges in accessing them. Instead, our goal is to create freely accessible and specialized tools that focus on particular design tasks and audience, and their corresponding needs.

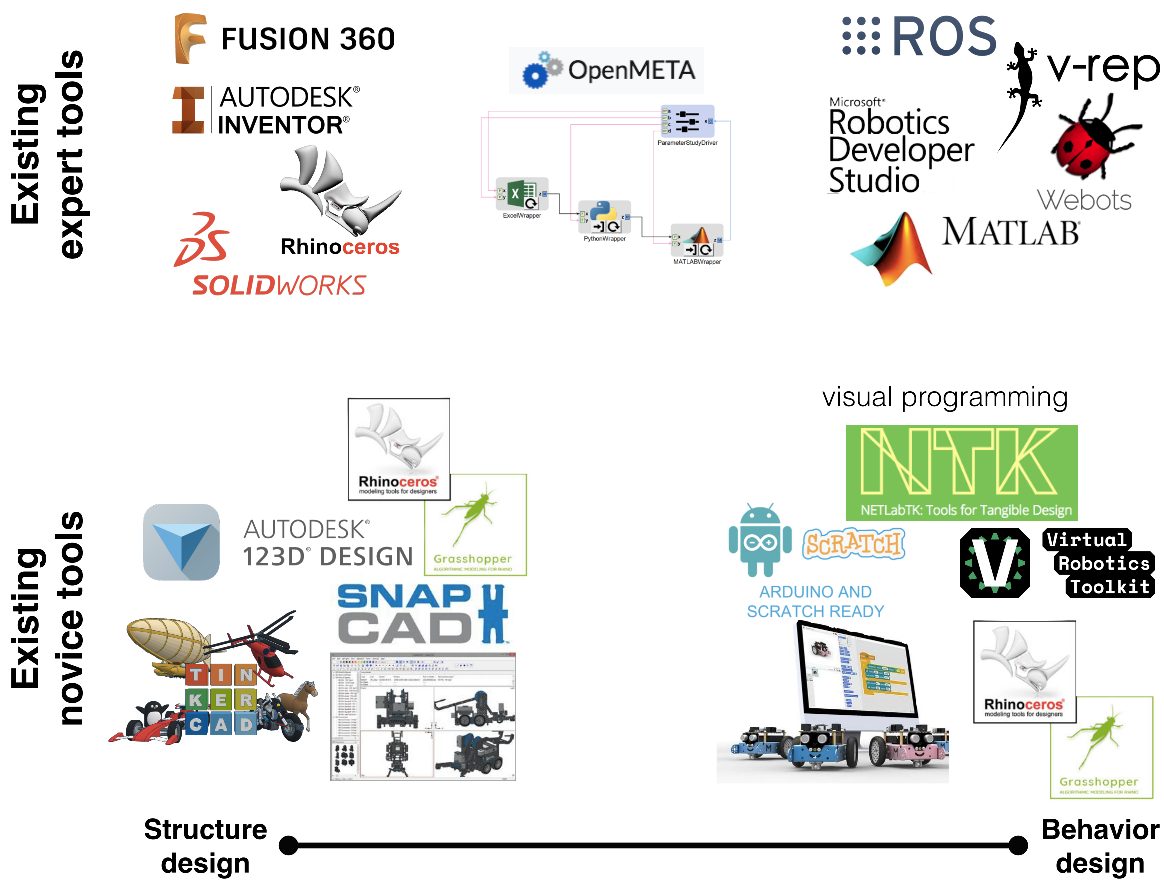


Figure 2.2: Prominent commercial and open-source tools for structure and behavior design of robots are shown here. Expert tools tend to have many features for a variety of design scenarios, while novice tools are more specialized for certain tasks. Intuitive visual design is also a critical component of novice tools.

Similar to structure design tools, robot function/behavior design tools for experts such as Robot Operating Systems (ROS) together with OpenRAVE and Gazebo [81, 157, 178], Microsoft's robotics developer studio [202], Matlab and Simulink [132], Labview's robotic simulator [194], V-REP [220], Webots [229] and many more[1] provide a lot of features. These features include dynamic modeling of robots, simulation of robot behaviors, interfacing and monitoring of robot hardware, analysis and testing of robot behaviors in realistic scenarios etc. These tools are spread over a wide spectrum of usage scenarios. For instance, Matlab and Simulink are preferred for quick prototyping and analysis of control, while ROS and Microsoft's robotic developers studio are mostly used for system integration and transfer to hardware. Apart from these generic tools that support a variety of robotic systems, platform specific tools are also available. Examples include design tools for ABB robots [201] and NAO robots [77] tools for ground robots [15] etc. Proper use of these tools demands a lot of domain knowledge. Further, many of them require the use of programming languages such as C++, Python, Java, making them less accessible to casual users. Our function design tool instead enables users to interact with a robot simulation visually. Further, it allows users to design behaviors based on high-level/semantic intent without worrying about robot's dynamics or low-level control parameters.

Finally, tools that enable co-design of robot structure and behavior have been developed in the cyber-physical systems community such as the OpenMETA [156]. OpenMETA integrates modeling tools such as Solidworks, Pro-E, and control design tools such as ROS, Matlab and Simulink, along with optimization tools for trade-off studies and design synthesis. Our co-design tools are motivated by the ability of such tools to optimize robot designs, structurally and functionally, for a given task.

2.2 Robot design tools for novices

Existing design tools for novices try to overcome the accessibility issues corresponding to the expert tools. To do so, first, these tools enable visual design and programming of robots. For instance, Scratch for Arduino [48], NETLab Toolkit (NTK) [217] allow programming behaviors of IoT robotic devices using visual drag-and-drop blocks. Visual block based programming is combined with simulation for LEGO and VEX robots by the Virtual Robotics Toolkit [218] and for NAO robots by Choreographe [45]. Likewise, Microsoft's robotic developers studio enables behavior design of more generic robotic systems using its visual programming language (VPL) [120]. For 3D design, Rhinoceros features a visual block-based graphical algorithm editor called Grasshoper, which is popular among artists and architects [88]. Grasshopper also features plugins for visual parametric control and simulation of industrial robots such as KUKA [166].

Secondly, tools for casual users try to encode the domain knowledge and simplify user interactions. For example, structure design tools such as Tinkercad and Autodesk's 123D Design [19, 22] use 3D primitives for design, instead of engineering sketching followed by extrusion operations in expert CAD tools[2]. Similarly, Sketchup leverages existing templates to aid the design [5]. Tools such as Leopoly and morphi combine such easy design with other modalities such as VR to enhance understanding of casual users [123, 147]. Finally, accessibility is further achieved by developing specialized tools that focus on particular tasks. For instance,

[1]A good overview of robot simulation based tools is available at – `https://www.smashingrobotics.com/most-advanced-and-used-robotics-simulation-software/`

[2]A list of 3D modeling tools for novice users can be found here – `https://www.3dnatives.com/en/3d-software-beginners100420174/`

CAD tools such as SnapCAD, LDraw, Lego Digital Designer etc. have been developed specially for LEGO robots so as to encode all the necessary design constraints and capabilities of LEGOS [62, 102, 225]. We built upon these ideas put forth by existing tools for novices, for developing tools in this thesis.

2.3 Design tools for novices from academia

Robotics research community has also started emphasizing on design tools that support on-demand generation of robotic devices by the people [6, 28]. Similar to existing commercial and open-source tools, researchers have developed tools for structure design [136, 138], function design [33, 54], as well as co-design of robots [67, 134, 135, 137, 188, 199]. In particular, many of the structure and co-design tools have focused on print-and-fold origami robots [135, 136, 137, 199]. Other than origami-based robots, design tools for multicopters and quadcopters [67, 138], and walking creatures [134] have also been developed. Apart from tools that focus on behavior and structure design of specific types of robots, application and audience specific tools have also been developed. Examples include visual behavior design tools such as CREATE Lab Visual programmer for Art bots [54] and Ruru for ground robots [63] that are focused on students and education. Likewise, a specialized visual programming tool focused on the needs of artists and creative tasks has also been devised [33]. Finally, tools that increase accessibility of existing frameworks have also been developed. For example, Bezzo *et al.* developed a visual environment called RosLab based on ROS to enable simplified and intuitive programming and hardware design of robots [27, 180].

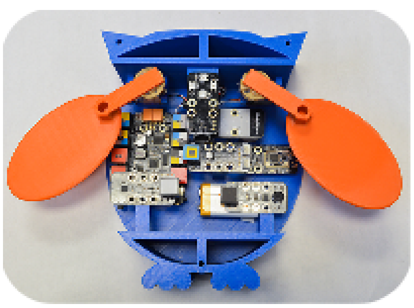
We take inspiration from this plethora of academic tools for enabling design of more generic robots with different form factors and functionalities, which can be built using custom digitally fabricated parts and off-the-shelf electromechanical components. We note that many of these existing tools leverage expert-designed building blocks or design templates [136, 188]. Our tools refrain from using such expert-created domain knowledge in order to be more scalable. Further, some of the design tools use custom scripting language [136, 138], while others leverage visual design [67, 134, 188]. We prefer to use visual interfaces for our tools inspired by the success of visual programming paradigm for novices [41], and the prevalence of visual interaction in existing tools for novices. We are also motivated by co-design tools to leverage integrated simulation-based feedback for design [134]. Finally, our vision of moving towards tools that enable design based on high-level user specifications is also shared by other researchers in the community [137, 139, 199].

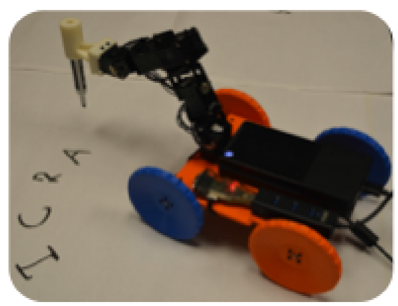
Part I

Structure design tools

design of
non-articulated robots

design of
articulated robots





< 2 joints
many components
(sensors, controller etc.)

>= 2 joints
chain of
components

Chapter 3

Assembly-aware design for non-articulated robotic devices

3.1 Preamble

This chapter describes our first structure design tool meant to enable the creation of non-articulated robots. This tool enables casual users to create a wide class of functional and customized 3D printed gadgets such as smart home appliances, toys, interactive art, and so on. In particular, our tool empowers novices to build personalized gadgets with any 3D shape, and electromechanical components. Figure 3.1 shows one such example of a smart crib monitoring toy shaped as an owl.

3.2 Introduction

The emergence of easy-to-use computer-aided design (CAD) software, such as SketchUp and Autodesks 123D suite, has enabled the general public to create content for digital fabrication [5, 19]. The goal of this work is to make equally accessible the creation of 3D printed devices capable of rich interactions with the world around them. Our work is inspired by *Voxel8*, a recently introduced 3D electronics printer [227]. Voxel8 employs a print-in-place strategy where external components are completely encased into 3D printed objects during the fabrication process. Such

Figure 3.1: Our design tool allows casual users to make complex 3D printed devices with integrated off-the-shelf electromechanical components. Chirpy – a smart crib monitoring toy made with our system is shown here.

embedding of interactive elements during the printing process have also been explored for optical and pneumatic 3D printed interactive devices [223, 233]. While very promising for certain application domains, this approach does not allow the embedded components to be accessed for repair or upgrades after the device is created. This limitation highlights the importance of assembly-based approaches, more prevalent in traditional manufacturing wherein complex artifacts are designed to be put together, and take apart as needed. Inspired by this, we therefore advocate a complementary approach that we call *assembly-aware* design in our work, where off-the-shelf components are mounted within a 3D printed *enclosure* as a post process (see Fig. 3.2).

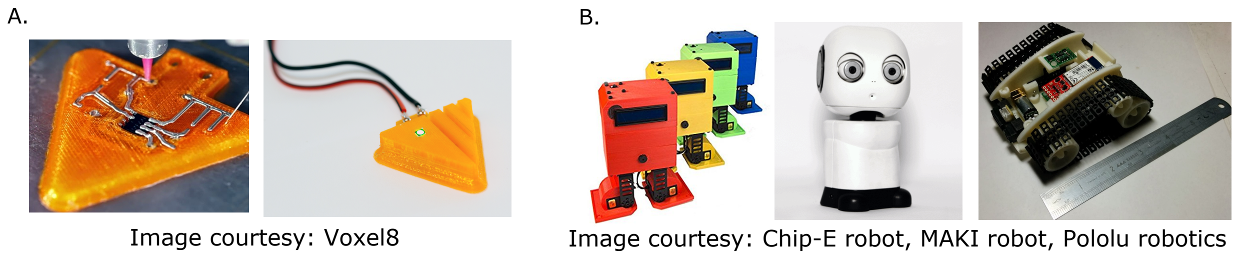


Figure 3.2: Assembly-based vs. print-in-place design approach – (A) Voxel8 uses a print-in-place design strategy for creating functional 3D printed devices with integrated electronics. (B) Instead, we propose to design devices that allow integrating electronics via assembly, so as to enable creation of much wider category of artifacts.

To design a physical device according to a desired functionality, one must first choose a suitable set of electromechanical components. The layout of the components within the device has to then be generated. For designing functional devices, the problem of choosing right components for a functionality and layout design are both important, but complementary. Many solutions have started to emerge for the former task. In particular, Censi proposed an approach to select discrete robot components, including batteries and actuators, based on constraints operating on mixed discrete and continuous variables [39]. Likewise, Ramesh *et al.* use constraint solvers to generate circuit level block diagrams, starting from user-specified requirements and a library of available components [168]. A system to automate component selection given robot function in natural language has also been proposed [139]. Therefore, we focus on the latter problem of device layout design in this paper.

Layout design for physical devices is a highly challenging task. The placement of the components within the enclosure, the configuration of mounting structures and the assembly process itself (e.g., assembly order, collision-free assembly paths, attachment strategies, etc) are all highly coupled and have to be concurrently considered. The difficulty of this problem led to the emergence of the Design for Assembly (DFA) sub-field of engineering, where product design is studied from the point of view of 'ease of assembly' [32]. However, within the computer-aided design and manufacturing community (CAD-CAM), product design is an iterative but sequential process. A product's layout design (parts and their connections) using DFA guidelines is first created by hand. For this manually created layout design, automatic assembly sequence generators [103], and industrial DFA softwares are used to evaluate the assembly process based on metrics such as time and cost [32], which allow the input design to be refined. However, creating an initial design or adapting an existing one remains a time-consuming, manually intensive task that is well beyond the capabilities of non-experts.

Overview and contributions: We develop a novel design system that allows users with no prior computer-aided design (CAD) experience to create physical devices that cater to their individual needs and preferences. In order to provide ample room for control over the functional and aesthetic form of the devices, our system lets the users specify a virtual object corresponding to the desired physical device, as well as a list of electromechanical components. Along with an easy to use user-interface, our system consists of a powerful computational method for automatically optimizing the integration of components into the final 3D printed device. In particular, we encode the design and assembly process using a spatio-temporal model. This model captures the layout of the components within the device, the paths traveled to reach their final placement, the support structures that they will be mounted on, and the relative order in which they are to be assembled. To the best of our knowledge, our model is the first to approach the layout and assembly of 3D printable devices in a coupled manner. Furthermore, our formulation is rooted in engineering design principles. By encapsulating domain knowledge, our model can better serve the needs of DIY makers, and artists (our target audience) for creating one-off designs.

Apart from modeling the spatio-temporal assembly process, we also develop an efficient algorithm to concurrently optimize all aspects of this model. The algorithm we propose couples a Markov Chain Monte Carlo based optimization strategy [83] with a gradient-based method, while utilizing heuristics that encode insights from the CAD community. The resulting scheme handles both continuous and discrete model parameters, and features increased robustness to local minima. We demonstrate the effectiveness of our computational approach by designing and fabricating an assortment of electromechanical devices. Our examples are representative of the types of devices available in online-community driven repositories. Each device features unique form factors and functional capabilities, and employs standard off-the-shelf electromechanical components embedded into 3D printed enclosure. We also show that such devices are time consuming and difficult to design, especially by non-experts, through a user-study. While many participants failed to design certain devices in 45 minutes, our computational approach created valid designs for those devices within 4 minutes.

3.3 Related work

Design for fabrication: The research community has contributed heavily to the development of powerful computational tools that fuel the personal fabrication revolution. Examples include methods to generate 3D printable objects that are lightweight yet strong [128, 200], objects whose optimized mass distribution allows them to stand, spin or float stably [23, 148], and mechanical automata capable of creating compelling motions [40, 51]. These tools share the same high-level goal as ours: empowering casual users in creating complex physical artifacts without requiring domain specific knowledge. However, these artifacts are limited by the abilities of 3D printers. To create objects with diverse functionalities, we aim to develop a computational framework to seamlessly integrates off-the-shelf components and 3D printed structures.

Combining off-the-shelf components with 3D printed parts allows us to harness the best of both worlds – traditionally manufactured parts offer cost-effectiveness, durability, and advanced functionality while 3D printing allows customization. Motivated by this, other researchers have also recently proposed tools for fabricating 3D printed artifacts with sensors and motors such as walking creatures [60, 134], and multicopters [67]. Our system intends to provide similar ease of design for making generic smart devices with a wide range of functional repertoires.

3D printed smart objects: Researchers in the HCI community have also been interested in enabling users to build 3D printed objects with embedded electronics [16, 105, 184, 226]. Savage et al. [184] and Jones et al. [105] present a fabrication pipeline using tangible means, while Ashbrook et al. [16] use an augmented fabrication system to enable fabrication of functional and assemblable objects with embedded components. Tools for tangibly designing laser-cut custom enclosures for prototyping interactive objects [186], and retrofitting existing devices with sensors and actuators [167] have also been proposed. Some of these systems also automatically create mounting structures for the components. However, the onus of deciding component layout such that the design is assemblable lies on the end user in their systems. Our system offloads many of the decisions requiring engineering knowledge from the user, while involving them in the creative aspects of the design process. In a complementary approach, Weichel et.al [230] propose to design an enclosure that 'fits' all the desired components. Instead, we assume a fixed, pre-defined enclosure considering applications where the users might care about functionality or aesthetics, such as a bunny-shaped table lamp.

Layout design and Stochastic optimization: Computational layout design has been addressed for a wide range of applications such as webpage and document designs [153], VLSI design [110], city and architectural layouts [140], furniture layout for interior design [141, 237], and for the design of simulated worlds [78, 236]. Many of these approaches apply Markov Chain Monte Carlo (MCMC) based techniques to optimize the layouts, owing to the highly multimodal nature of the layout cost function. Motivated by their success, we also apply MCMC based optimization for our problem. We are also inspired by other stochastic optimization approaches that utilize gradient information for efficient sampling such as Hamiltonian Monte Carlo (HMC) [150], Latin Complement Sampling (LCS) [29], and Sequential Monte Carlo mixed with gradient descent [124]. While HMC and LCS are more suitable for continuous domains, our cost function has both discrete and continuous parameters. To handle the additional assembly and fabrication constraints that our problem poses, we propose an interleaved optimization strategy.

Assembly planning: Assembly planning is a well-studied problem in automated manufacturing and robotics [103, 177, 234, 235]. Traditionally, an assembly planner computes all geometrically feasible sequences of assembly operations, given a known layout. However, a design's layout and assembly plan generation are highly coupled. Therefore, our solution concurrently optimizes with the help of a new model that captures spatio-temporal aspects of the assembly process.

3.4 Designing smart 3D printed devices: Overview

We present a comprehensive system for designing functional 3D printable devices. As illustrated in Fig. 3.3, our interactive design system allows users to incorporate different types of off-the-shelf components into the device of their choice. It then computationally generates a 3D layout for the user-selected components, as well as an assembly plan. The resulting design is guided by engineering design principles and promotes ease of assembly. In this section, we give an overview of the design process that our computational approach enables, and describe the engineering design principles that we incorporate, in detail.

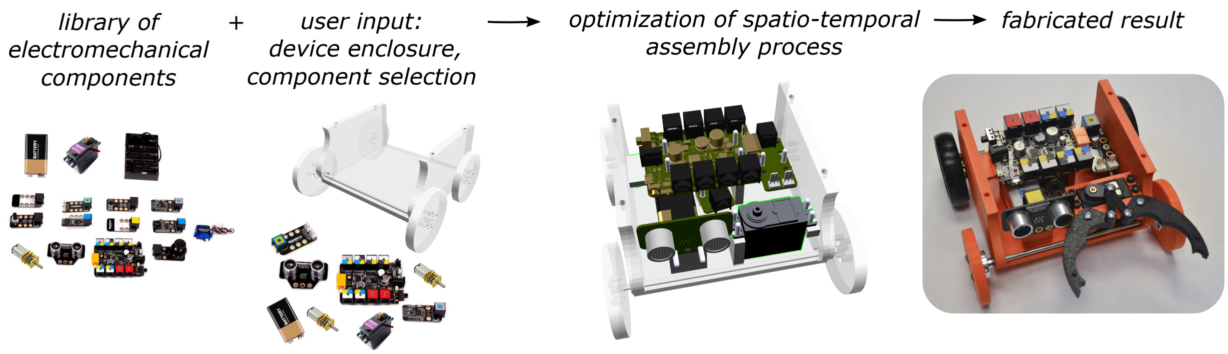


Figure 3.3: Overview of assembly-aware design system – Given a user-defined device enclosure, and a set of components selected from a library, our system optimizes an assembly-aware design of the device.

3.4.1 Design process

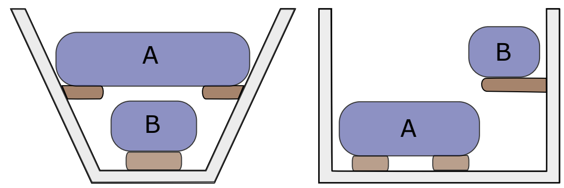
In creating a functional electromechanical device, informed decisions must be made with respect to four highly intertwined aspects of the design:

Conceptual design: The off-the-shelf components that are added to a design depend on cost and availability, and they directly shape the functional capabilities of the resulting device. Our graphical design system allows users to select electromechanical components from a library through familiar drag-and-drop operations. While adding and removing components is made effortless by our design system, the decision of which components to employ is left entirely to the user.

Layout design: The spatial arrangement of components into a device constitutes the layout design. Given the geometric description of a device's internal enclosure and a set of components to incorporate, many collision-free layout designs might exist. From this space of feasible solutions, our optimization-based approach aims to find a layout plan that simplifies the assembly process as much as possible.

Mounting structures: While users explicitly provide the list of electromechanical components they wish to employ, our design system generates appropriate mounting structures automatically. The configuration of these mounts is directly governed by the layout design. Conversely, the design layout needs to be aware of mounting structures and assembly strategies. For example, if fasteners are needed to secure a component to its corresponding support, then care must be taken to ensure that nothing is placed in their path. We assume that mounting structures will be seamlessly assimilated into the 3D printed enclosure of the final design. Therefore, to generate mount geometry while optimizing the entire assembly process, each off-the-shelf component stores auxiliary data about compatible fastener configuration and mounting structures.

Assembly planning: The assembly plan stores the sequence of operations needed to integrate all off-the-shelf components into the 3D printed enclosure of a device. To ensure that components can be assembled, their layout, the order in which they are added to the design, and the collision-free path they must travel to reach their final placement need to be concurrently considered. The simple scenarios shown in the inset figure illustrate this challenge. In the example on the left, only one possible layout exists due to the shape of the enclosure and the shape of the components. This layout dictates that component A should be assembled after component B. In the example shown on the right, component A can be placed either on top of or below B. However, this layout choice is dependent on the order in which the components are to be integrated into the device. Imposing any specific ordering ahead of time restricts the space of feasible component layouts once the entire assembly process is considered. Unfortunately, this can all-too-easily lead to design choices that admit no feasible solutions. To make matters worse, as the number of components in a design increases, it becomes impossible to predict which assembly orderings will lead to good designs and which ones will not.



The highly coupled nature of these different design aspects leads to a technically challenging problem that exhibits both continuous and discrete parameters. Our system (Fig. 3.3) simplifies the design process by providing tools to visualize conflicts during manual design, as well as providing the option to automatically and simultaneously determine a layout, mounting design, and assembly plan from a set of components and an enclosure. To model the design process in a manner that is amenable for manual assembly by novices, we incorporate Design for Assembly (DFA) guidelines, described next.

3.4.2 Ease of assembly

Design for Assembly (DFA) guidelines are a result of extensive research in mechanical design and manufacturing community. These guidelines are aimed at improving product designs for easy and low-cost assembly [32, 159]. In particular, the DFA guidelines for positioning and handling components recommend the following for easy manual assembly [12, 159]:

- Simple linear path for component assembly
- Translational movements along assembly paths
- Assembly of one component at a time directly into component's final configuration

The first two guidelines are based on the observation that multiple orientations, and complex paths during manual assembly can be confusing, and prone to errors. The final guideline aims at reducing the complexity of the assembly sequence by assuming that the assembly can be done using only two hands in a monotonic manner (without the need of intermediate placement of components). Similar guidelines are also used for designing assembly instructions for existing products [9], and for IKEA furniture.

3.5 User Interface

Figure 3.4(A) illustrates our graphical user interface (GUI). It consists of a workspace window in the center, a palette displaying various electromechanical components on the right, and a menu bar with various editing options on the left. A design session typically begins with users dragging a desired 3D enclosure for their device into the workspace (for instance, a car enclosure in Fig. 3.4(A)). Users can then add components of their choice from the palette using drag and drop operations. Mounting structures and fasteners are automatically added along with each component, and are updated whenever the component is moved. The layout of the components within the device as well as corresponding assembly plan can be designed using either manual or automatic design modes.

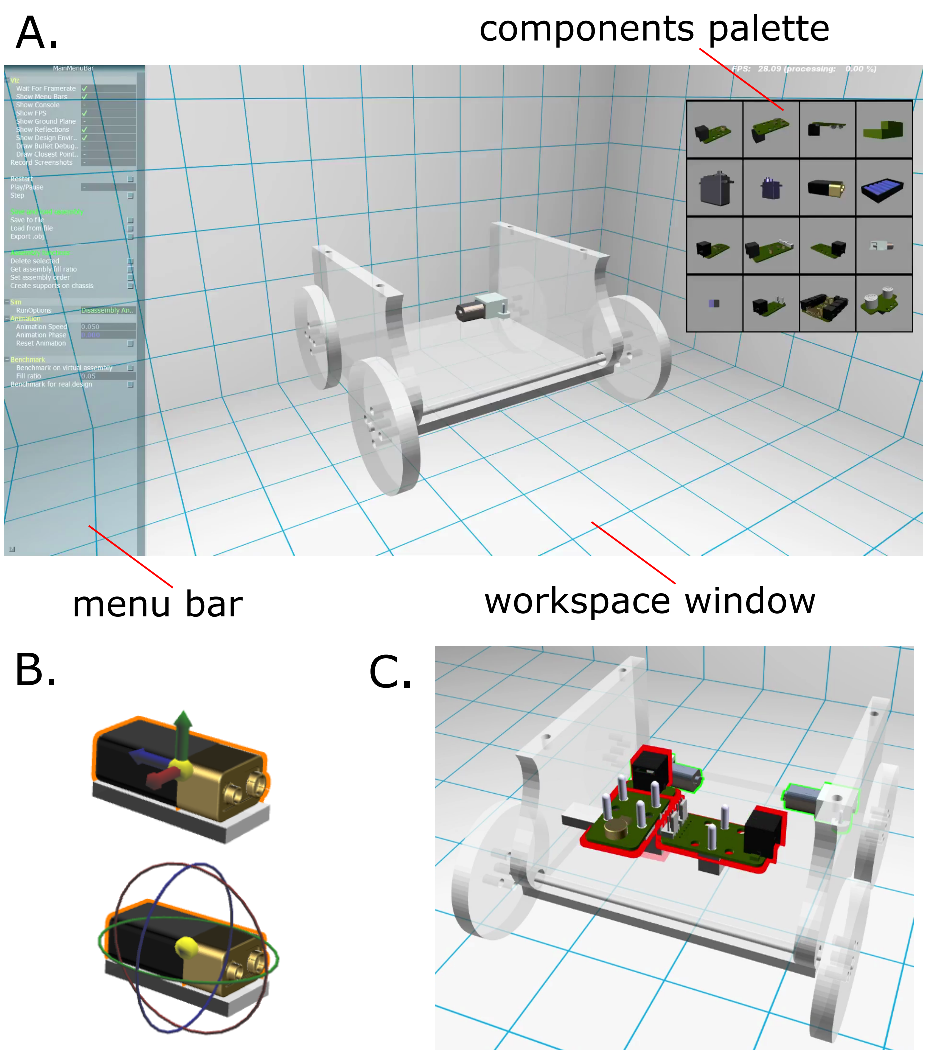


Figure 3.4: (A) Our user interface is shown here with its three main elements – components palette, main workspace window, and editing menu bar. (B) Translation and rotation widgets can be used to manually configure a component. (C) In order to provide guidance to the user, our system highlights components that lead to infeasible designs in red.

In *manual design mode*, the user uses translation and rotation widgets to place each component (Fig. 3.4(B)); and sets the assembly plan using options in the menu bar. Our system provides assistance by highlighting components that lead to design infeasibility (Fig. 3.4(C)). Such infeasibility may occur when components are colliding with each-other, or when a component or its fasteners' assembly is blocked by other components in the device. In contrast, *automatic design mode* determines a valid layout and assembly plan using optimization. As it optimizes, our system displays the current best configuration, and allows users to pause and modify the design if desired.

Once the device is designed, users can export the enclosure (including synthesized mounting structures) for 3D printing, and the assembly plan can be animated for guidance during fabrication. Figures. 3.1, 3.9 show some custom enclosures generated in this manner. Our video[1] illustrates various capabilities of our system, along with automatic design.

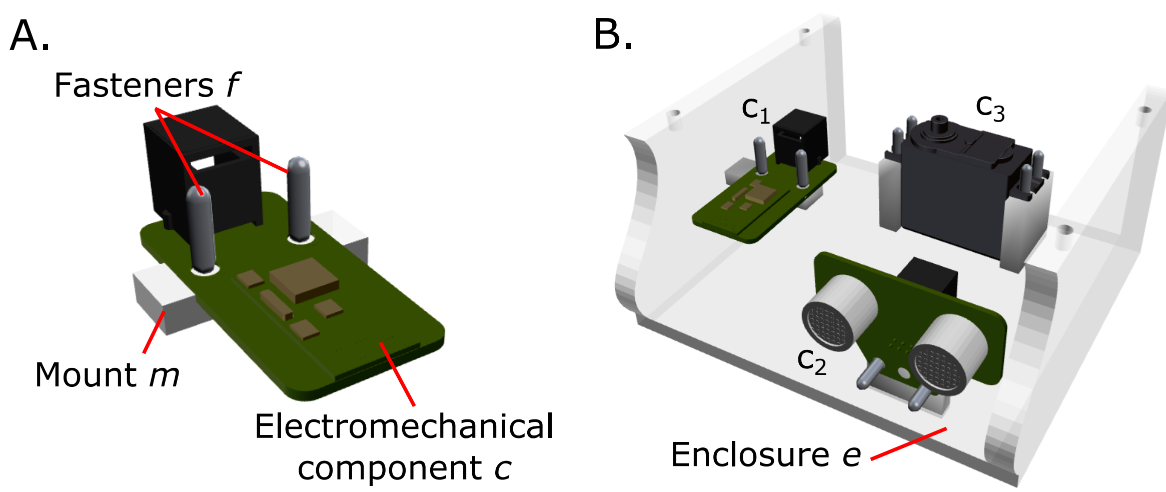


Figure 3.5: Component and device definitions – (A) An electromechanical component c is shown with its fasteners f and mount m. The configurations of f and m are defined with respect to c's local coordinate system. (B) A device $\mathbb{D}$ consisting of electromechanical components c_1, c_2, c_3 bounded within an enclosure e. c_i can be supported by a single or multiple mounts and their unique set of fasteners attach to c_i in a specific manner. c_i's mounts get extended to rigidly attach to e's walls.

3.6 Modeling electromechanical devices

Formally, our system represents an electromechanical device (Fig. 3.5B) and its assembly plan as an ordered tuple $\mathbb{D}$ of parts. Each part d_i has a configuration $\phi_i = (x_i, R_i)$ consisting of 3D position x_i and a 3D rotation R_i, shape attribute S_i at ϕ_i, and an assembly path $P_i(t)$ that defines configuration for the part at every time t during its assembly:

[1]Video illustrating our assembly-aware design system for 3D printable electromechanical devices is available at – `https://youtu.be/DrIXD5Fpg0I`

$$
\begin{aligned}
\mathbb{D} &\equiv (d_1, \ldots, d_n)\,, \\
d_i &\equiv (S_i, \phi_i, P_i(t))\,.
\end{aligned}
\tag{3.1}
$$

Notice that this representation stores both the layout of the device – each d_i ends up assembled at configuration ϕ_i – and the assembly plan – start with all components at $P_i(0)$, then in order from 1 to n move each part i along path P_i to its final configuration $P_i(1) = \phi_i$. We therefore call this a spatio-temporal model of device assembly. Note that assembly plans represented in this model always assemble parts into their final configuration one at a time as suggested by the Design For Assembly (DFA) guidelines [32, 159] (Sec. 3.4). DFA guidelines also suggest that multiple rotations and complex paths during assembly can be confusing. Therefore, we only allow piece-wise linear assembly paths P_i, and translational motions along P_i (R_i is kept fixed).

The shape attributes S encode the space occupied by components, fasteners, and mounts within $\mathbb{D}$. S enables collision detection between these parts and forms the basis of our optimization process. The shape attribute of the enclosure S_e represents the total space in which all components of $\mathbb{D}$ need to fit.

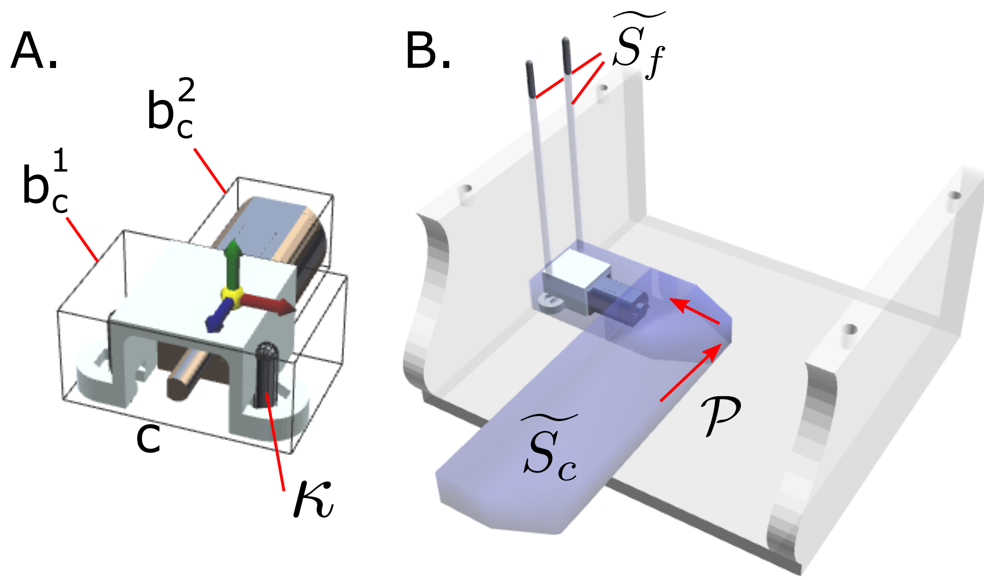


Figure 3.6: Shape attributes – (A) An electromechanical component c is shown with its shape attribute – a set of bounding boxes ($S_c = \bigcup b_i$). Fasteners are encoded with capsule shapes (shown by black mesh). These shapes are defined in the component's local coordinate frame shown at its center with a triad. (B) The swept shape $\widetilde{S}_c$ of c along a piecewise linear path P and that of its fasteners ($\widetilde{S_f}$) along their default paths are shown.

3.6.1 Validity

Now that we have given a description of a electromechanical device, we can explicitly define the notions of a valid layout and assembly plan.

For a device to have a valid layout, it must be the case that none of its parts have overlapping shapes in their assembled positions:

$$\forall i \neq j : \; S_i \cap S_j = \emptyset. \tag{3.2}$$

While for a device to have a valid assembly plan, the volume swept out by moving a part's shape along its assembly path must not intersect any previously-assembled part:

$$\forall i < j : \; S_i \cap \widetilde{S}_j = \emptyset. \tag{3.3}$$

where the "swept path" $\widetilde{S}_j$ of a component is the union of its shape S_j over all positions along its assembly path P_j. (Fig. 3.6B):

$$\widetilde{S}_j = \bigcup_{P_j} S_j. \tag{3.4}$$

3.6.2 Parts

The values and degrees of freedom in S_i, ϕ_i, and P_i are determined by the type of part being represented:

The **enclosure** represents the case surrounding the device. Every device contains exactly one enclosure e. The enclosure is always configured at world's origin O with identity rotation $\phi_e = (O, I)$. It does not move during assembly:

$$e \equiv (S_e, \phi_e, 0)\,. \tag{3.5}$$

In our system, enclosures are always convex polytopes (S_e) with one or more solid faces and one or more lids (which are assumed to be assembled last). When checking that the layout is valid (eq. 3.2), collision checks are performed against the exterior of the polytope; while when checking for valid assembly paths (eq. 3.3) collision checks are performed against only the solid faces of the polytope.

Components are electromechanical components (e.g., microcontrollers, sensors, motors, batteries). Every component c has a shape given by a union of axis-aligned boxes (Fig. 3.6A), a configuration ϕ_c, and an assembly path P_c:

$$\begin{aligned} c &\equiv (S_c, \phi_c, P_c)\,, \\ S_c &\equiv \bigcup b_i, \end{aligned} \tag{3.6}$$

$$P_c(t) \equiv \begin{cases} x - \vec{x_2} - (1-2t)\vec{x_1} & \text{if } t < 0.5 \\ x - (2-2t)\vec{x_2} & \text{otherwise} \end{cases} \tag{3.7}$$

Assembly paths P_c are piece-wise linear (as per DFA) and can be precisely represented as functions of time. In our implementation, P_c are defined by at most two linear segments – $\vec{x_1}$ and $\vec{x_2}$. Note that, x denotes the component's final position as in ϕ_c, while rotation of the component along P_c is assumed to be unchanged from its value in ϕ_c. The first assembly step, $\vec{x_1}$, is parameterized by two spherical angles, α, β, and a radius, r:

$$\vec{x_1} \equiv r\left[\cos\alpha \cdot \cos\beta, \cos\alpha \cdot \sin\beta, \sin\alpha\right]^t \tag{3.8}$$

This parameterization is used because fixing r to a sufficiently large value ensures that components always start outside the enclosure during assembly, leaving only α and β to be optimized over (or set by hand).

The second assembly step in the path, $\vec{x_2}$, is used for components that need to be slid into mounts or through holes in the enclosure, like motors (e.g., Fig. 3.6B). The direction and length of this vector is set based on a fixed value stored in the component library, so it does not contribute any degrees of freedom to the optimization. For components that do not need to be slid into mounting structures, $\vec{x_2}$ is set to zero.

Components have associated mounts and fasteners (Fig. 3.5A), which are also represented as parts in the device.

Mounts, m, are structures added to the enclosure to give components something to attach to. Each mount is associated with a component c. The mount's shape S_m and configuration ϕ_m depend on c. S_m is a convex hull and is determined by extending a convex polytope spanning c's fastener sockets to the closest wall of the enclosure. Because mounts are printed as a part of the enclosure, their assembly path are the same as that of the enclosure (i.e. $P_c = 0$).

$$m \equiv \left(S_m^c, \phi_m^c, 0\right), \tag{3.9}$$

where superscript ^{c} shows the dependence on c.

Fasteners (e.g., rivets, screws) are small parts used to affix components to their mounts. Each fastener f has shape S_f given by a bounding capsule, while its configuration ϕ_f depend on the sockets on c, and can thus be determined using fixed orientation (R_f) and position offsets (x) from the orientation and position of its associated component c (Fig. 3.5A).

$$f \equiv \left(S_f, \phi_f^c, P_f^c\right), \tag{3.10}$$

$$\phi_f^c \equiv \left(P_c(1) + R_c x, R_c \cdot R_f\right), \tag{3.11}$$

$$P_f(t) \equiv P_c(1) + R_c\left(x - (1-t)(\vec{x_1})\right), \tag{3.12}$$

where superscript ^{c} shows the dependence on c. As with components, the assembly path $\vec{x_1}$ is set to be sufficiently long that the fastener must start outside the enclosure. Unlike components, however, the path's direction is fixed based on sockets on c (Fig. 3.6B).

Our model does not represent wires or account for their routing during design. Instead, the availability of wires in desired lengths, and their flexibility enable users to insert them as per choice during assembly.

3.6.3 Degrees of freedom

Components in our model have six layout degrees of freedom for configuration ϕ (a 3D position x and a 3D orientation R). These are the only degrees of freedom (DOF) in the device – the enclosure is fixed, and the fasteners and mounts are computed based on the layout of the component. Our interface also allows the user to further lock particular layout DOF of components. For example, a range sensor or a light emitting diode (LED) may need to be fixed to a specific location or in a plane for aesthetic or functional requirements, and thereby may expose only two layout DOF. On the other hand, a controller or a battery that can be configured without restrictions may expose all six layout DOF. For ease of use, we represent the exposed layout DOF as $p_i^l \subseteq \phi_i$ for each component in a device. The overall device layout $L(\mathbb{D})$ then becomes:

$$L(\mathbb{D}) = \left\{ p_i^l \mid \forall c_i \in \mathbb{D} \right\} \tag{3.13}$$

Components also have two assembly DOF (the spherical angles α and β used to define the assembly path P). Further, all parts in our model have an assembly order given by their index in the device tuple (eq. 3.1). Because mounts and the enclosure don't move during assembly, our system always places them first in the tuple; similarly, fasteners always appear immediately after their associated components in the assembly order (see Fig. 3.7). Thus, the order of the components determines the overall assembly order of the device. Similar to $L(\mathbb{D})$, we succinctly define the device assembly plan $L(\mathbb{D})$ as:

$$A(\mathbb{D}) = \{ i, p_i^a \mid \forall c_i \in \mathbb{D} \} \ . \tag{3.14}$$

where $p_i^a = (\alpha_i, \beta_i)$ are the assembly path P's parameters, and i is the index of component c_i in the device tuple. (eq. 3.1)

In summary, when designing a device with C components, our optimizer must determine values for up to $8C$ continuous variables, and select a discrete ordering among the C components.

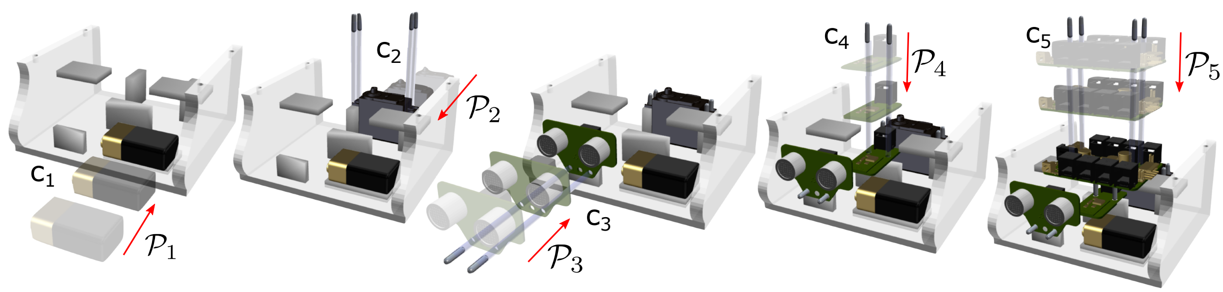


Figure 3.7: Assembly process – The assembly of a device with 5 components is shown. Each component c_i is assembled one at a time by translating along its assembly path P_i. The assembly process is parameterized by component's assembly order (i in $\mathbb{D}$), and the parameters of P_i. After assembly, c_i is affixed by assembling its fasteners. The fasteners need to be assembled along a specific path. There is a strong interplay between the layout of the components, and their assembly order and paths. This can be seen during the assembly of c_4. Owing to its configuration, c_4 can only be assembled along P_4 before c_5.

3.7 Assembly-aware device optimization

Our optimization aims to find a device design $\mathbb{D}$ with valid layout (eq. 3.2), and a valid assembly plan (eq. 3.3), by simultaneously searching over both layout and assembly degrees of freedom (eq. 3.13, 3.14).

This simultaneous optimization stands in contrast to previous work which either optimizes layout for applications of furniture, and virtual world layout design [78, 141, 237] or optimizes assembly given layout for various manufacturing and engineering design applications [103, 234].

3.7.1 Cost function

We define a cost J for a device $\mathbb{D}$ to characterize how assemblable it is.

$$\begin{aligned} &\underset{L(\mathbb{D}),A(\mathbb{D})}{\text{minimize}} \quad J(\mathbb{D})\,, \\ &J(\mathbb{D}) \equiv J_c + J_b\,. \end{aligned} \tag{3.15}$$

$J(\mathbb{D})$ is the summation of collision penalty J_c, and bounding penalty J_b defined over all elements d_i in $\mathbb{D}$. J_c penalizes the collisions during assembly, while J_b constrains all the elements to stay within the enclosure. In order to define these penalties, we need to quantify overlap between shapes, which we do with a smoothed signed distance overlap cost.

Signed distance measure δ

A signed distance measure δ between a pair of shape attributes S_i and S_j is defined as follows:

$$\delta(S_i, S_j) = \begin{cases} -PD(S_i, S_j)\,, & \text{if } S_i \cap S_j \neq \emptyset \\ \Delta\,, & \text{otherwise} \end{cases} \tag{3.16}$$

where S_i and S_j are defined for each part as explained in Sec. 3.6, and Δ is the shortest distance between them using Euclidean norm $\|\cdot\|$. Δ can be defined as $\min(\|x_i - x_j\|, \;|\; x_i \in S_i, x_j \in S_j)$. Penetration depth (PD) is a natural extension of Euclidean distance when d_i and d_j are overlapping and is defined as the minimum translation distance that one of them undergoes to make the interiors of their shape attributes S_i and S_j disjoint [108] (see Fig. 3.8A). Mathematically, $PD(S_i, S_j) = \min(\Delta \;|\; interior(S_i + \Delta) \cap S_j = \emptyset)$. We use a publicly available implementation based on the Expanding Polytope Algorithm (EPA) [1, 221] to compute $\delta(\cdot)$.

Smooth overlap cost $o(\delta)$

Our signed distance measure δ is not amenable to gradient-based methods. We therefore define a smooth overlap cost $o(\delta)$ to penalize overlapping elements in $\mathbb{D}$. The overlap cost $o(\delta)$ is a function which is quadratic when distance between shapes (δ) is less than zero, and is zero when shapes are sufficiently far from each other (see Fig. 3.8B). To ensure smoothness, a cubic function is defined over the intermediate distance range $0 \leq \delta < \epsilon$, where $\epsilon > 0$ determines the minimum separation between the elements in a device. ϵ allows us to define a safety distance margin between elements, which further aids easy assembly. We set it empirically.

$$o(\delta) = \begin{cases} a_1\delta^2 + b_1\delta + c_1, & \text{if } \delta < 0 \\ a_2\delta^3 + b_2\delta^2 + b_1\delta + c_1, & \text{if } 0 \leq \delta < \epsilon \\ 0\,, & \text{otherwise} \end{cases} \tag{3.17}$$

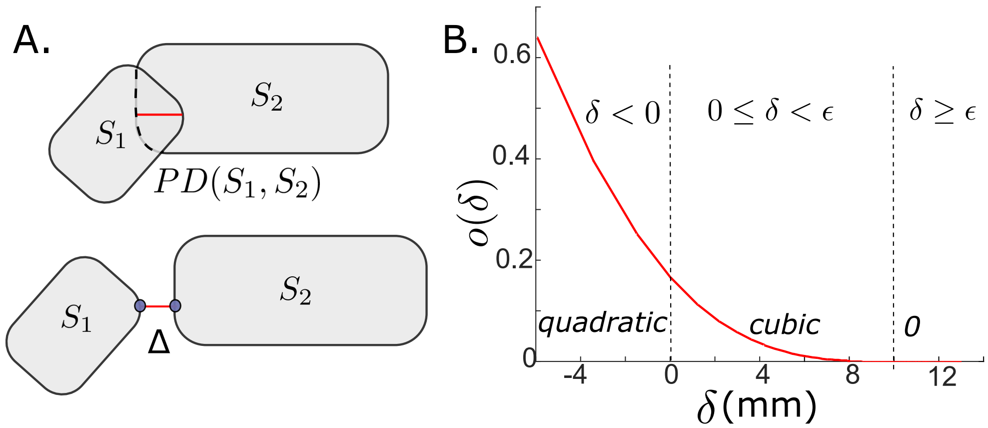


Figure 3.8: Measuring component collisions – (A) Signed distance measure δ defines distances between overlapping and non-overlapping shapes. When two shapes S_1 and S_2 overlap each other, δ is computed using the minimum translational length that will separate them (called penetration depth (PD)). Otherwise, δ is calculated as the Euclidean distance between the closest points of S_1 and S_2. (B) We use a C^2 continuous cost $o(\delta)$ to penalize overlapping shapes.

where a_1 corresponds to the stiffness of the quadratic cost. $b_1 = -\frac{a_1 \epsilon}{2}$, $c_1 = \frac{a_1 \epsilon^2}{6}$, $a_2 = -\frac{a_1}{6\epsilon}$, and $b_2 = \frac{a_1}{2}$ are constant weights that are determined such that the resultant overlap cost function is C^2 continuous. a_1 and ϵ are empirically set. Such a construct allows us to define a safety distance margin between elements using ϵ, which further aids easy assembly. The cubic function in o can also be interpreted as a C^2 continuous interpolation between the quadratic cost at $\delta < 0$ and the zero cost for $\delta \geq \epsilon$.

Collision and bounding penalties

Equipped with the concept of overlap cost, we are now able to define the collision penalty J_c, and the bounding penalty J_b. Driving these two penalties to zero will result in a valid layout and assembly plan.

The collision penalty, J_c, penalizes collisions between assembly paths of parts and those parts assembled earlier:

$$J_c \equiv \sum_{i<j} overlap(S_i, \widetilde{S}_j) \tag{3.18}$$

where $overlap(S_a, S_b) \equiv o(\delta(S_a, S_b))$ penalizes collisions between shapes S_a and S_b using the overlap cost $o(\cdot)$ and signed distance $\delta(\cdot)$.

The bounding penalty, J_b, forces component-type parts to remain inside the enclosure:

$$J_b \equiv \sum_{c} overlap(S_e, S_c) \tag{3.19}$$

The bounding penalty only considers component-type parts in order to save some computational cost – by construction, both mounts and fasteners will lie within the enclosure if their associated components are within.

Note that the enclosure shape used in the collision penalty is the shape without lids (as used in assembly validity), while the enclosure shape used in the bounding penalty is the shape with lids (as used in layout validity). While our cost function $J(\mathbb{D})$ only accounts for collisions within a device, other cost terms that capture desired properties of the device design can be easily added to $J(\mathbb{D})$. For example, in order to achieve material minimzation during 3D printing, one could include a cost term corresponding to volume of mounts to $J(\mathbb{D})$, thereby reducing the size of resultant mounts.

3.7.2 Numerical optimization

To optimize the cost function $J(\mathbb{D})$ as defined in eq. (3.15), we develop an efficient algorithm that combines heuristics inspired by the CAD design community, and powerful optimization strategies.

Heuristics

We interviewed an expert with five years of CAD experience in designing mechanical assemblies to understand the design practices in the community (see sec. 3.9 for more details). The expert supported simultaneous reasoning for assembly and layout of components during design. However, the expert highlighted that the expert would approach such concurrent assembly-layout design in an incremental manner. Instead of adding all components in a device at once, the expert would add one component at a time and focus on finding valid layout and assembly for this latest addition, before adding any more components. Similar incremental approaches have also been applied for automatic computer-aided design of VLSI [49, 52], architectural floor plans [53], specifically to deal with high design complexity, and to improve algorithm run times. Inspired by these, we adopt an incremental approach to ensure interactivity during design. Instead of searching for valid configurations (layout and assembly) of all components at once, we incrementally create partial device designs by adding and properly configuring one component at a time to the device. Incrementally adding components to the device optimization ensures that the search space complexity increases gradually, aiding interactivity. Finally, an additional component may be accommodated in a partial design with small reconfigurations of its existing components, if the component makes the best use of available empty space. We therefore reward the use of empty space during our incremental optimization.

Choice of optimization strategy

Our cost function $J(\mathbb{D})$ as defined in eq. (3.15), is highly multimodal with a null space. Further it has a mixture of discrete and continuous optimization variables. Determining the assembly ordering is a combinatorial problem while layout optimization is continuous. Markov Chain Monte Carlo (MCMC) based stochastic optimization methods have been successfully used in the past for combinatorial problems [101, 110]. However, standard MCMC methods tend to get stuck in a single mode while sampling from a multimodal probability distribution. Approaches based on multiple markov chains such as Parallel Tempering have been proposed to overcome this issue [17, 83]. These approaches however, do not offer a mechanism to exploit the availability of gradient information for continuous optimization variables. Recent approaches have shown the benefit of combining gradient based optimization with sampling for both continuous and mixed optimization problems [29, 124]. Using gradient information increases the efficiency of sampling by ensuring less-random walks of the markov chains in the parameter space. Inspired by these approaches, we combine gradient-based methods with Parallel Tempering for our problem.

MCMC algorithms

Typical MCMC methods perform a memoryless, random walk in the space of parameters ϕ by simulating a markov chain that generates samples from a function $f(\phi)$. These samples can be generated using a Boltzmann-like probability distribution such as:

$$P(\phi) = \frac{1}{Z} e^{-f(\phi)\beta}\,, \tag{3.20}$$

where Z normalizes the distribution, and $\beta \leq 1$ is known as an inverse-temperature constant. Even though computing Z is generally intractable, the Metropolis-Hastings (MH) MCMC algorithm [93, 142] allows exploring distributions without computing Z, in the following manner. Starting with a random configuration in the parameter space ϕ_1, a sample ϕ' is proposed at each time step, from an easy to sample proposal distribution $Q(\phi \mid \phi_t)$. ϕ' is accepted in the chain with a probability:

$$\alpha(\phi') = \min\left(1, \frac{P(\phi')}{P(\phi_t)} \frac{Q(\phi_t \mid \phi')}{Q(\phi' \mid \phi_t)}\right)\,, \tag{3.21}$$

where α is called the MH acceptance probability. If ϕ' is accepted, $\phi_{t+1} = \phi'$. We refer to this as the MH-update step.

Parallel Tempering (PT)

The amount of exploration is a function of a temperature parameter β. At higher temperatures (lower β values), the distribution P (in eq. (3.20)) is “flattened”. This increases the acceptance of samples ϕ' with $f(\phi') > f(\phi_t)$, thereby ensuring exploration. In PT, independent markov chains are run in parallel on a set of N distributions with inverse temperatures defined as $0 \leq \beta_N < \beta_{N-1} < \cdots < \beta_1 < \beta_0 = 1$. Periodically, the configurations of these chains are swapped probabilistically. This allows chains at higher temperature that tend to explore more, to pass information about better configurations to exploitative chains at lower temperatures, thereby allowing colder chains to escape local minima. A proposed swap at time step t, between chains k_1 and k_2 is accepted with a swap acceptance probability:

$$\alpha_{swap}(k_1 \leftrightarrow k_2) = \min\left(1, \frac{P_{k_1}(\phi_t^{k_2})}{P_{k_1}(\phi_t^{k_1})} \frac{P_{k_2}(\phi_t^{k_1})}{P_{k_2}(\phi_t^{k_2})}\right)\,, \tag{3.22}$$

where $P_{k_1}(\cdot)$, $P_{k_2}(\cdot)$ are distributions, and $\phi_t^{k_1}$, $\phi_t^{k_2}$ are the t^{th} time-step configurations corresponding to chains k_1 and k_2 respectively. The swap probability $\alpha_{swap}(k_1 \leftrightarrow k_2)$ between chains k_1 and k_2 is a function of overlap between their distributions. Therefore we swap configurations only between a pair of neighboring chains that is chosen randomly. Such a swap proposal is done after each iteration, where in each of N chains undergoes MH-update using eq. (3.21).

The performance of PT is dependent on the proposal distributions Q of the chains and sequence of their inverse-temperatures. We next describe our proposal distributions, and rationale behind selection of chain temperatures.

Proposal distribution Q

For exploring the space of possible layouts and assembly plans effectively, we define proposal distributions that generate proposed samples by perturbing the layout and assembly parameters. These perturbations allow local adjustments around the current values of these parameters as well as create global design changes. For each MH-update of a chain (eq. 3.21), one of these perturbation is executed.

Layout perturbation: The layout parameters p_i^l of components $c_i \in \mathbb{D}$ (as defined in eq. (3.1)) consisting of 3D position x_i and orientation R_i of c are perturbed in 4 ways:

- x_i of each component c_i is perturbed by adding a Gaussian term $\mathcal{N}(0, \sigma_x)$ to each coordinate.
- R_i of each component c_i is uniformly sampled from a set of valid orientations. We found this to work better empirically than perturbing R_i with a Gaussian term $\mathcal{N}(0, \sigma_R)$. This also results in more feasible designs since arbitrary orientations may result in unstable and hard to assemble configurations.
- Swap positions of 2 randomly selected components.
- Swap orientations of 2 randomly selected components.

The first two perturbations are 'local', while the last two perturbations allow the markov chains to jump to different parts of the parameter space. We employ rejection sampling from $\mathcal{N}(0, \sigma_x)$ to ensure that the resultant component configuration is within the bounds of the enclosure. σ_x is auto-tuned to achieve 23% acceptance rate during MH-updates of each chain (eq. (3.21)). This is based on the theoretical evidence that suggests this rate to be a good general setting [175].

Assembly plan perturbation: Based on our early experiments, we develop a set of heuristics to perturb the assembly parameters p_i^a corresponding to the assembly path, and the assembly order i (defined in eq. (3.14)).

- Instead of sampling the assembly path parameters p_i^a according to a Gaussian distribution, we uniformly sample these parameters for each component $c_i \in \mathbb{D}$ around a set of main directions that correspond to removable panels (lids) of the enclosure. Such biased sampling of assembly paths allows us to filter out paths that are blocked by enclosure walls, and speed up computation considerably.
- To perturb assembly order, we adopt a greedy strategy that allows for occasional exploration. We swap assembly order i (index in device tuple in eq. 3.1) of two randomly selected components with a small probability, or generate a heuristic ordering otherwise. This strategy is based on the observation that out of $n!$ assembly orderings for a set of n components, many orderings have the same outcome. For example, when a group of small components is blocked by a larger component, swapping the order between components in this group is counter-productive. Therefore, instead of resorting to un-informed sampling

in the assembly order space, our heuristic ordering is generated by considering the layout parameters. It is decided based on the distance of components from the main opening of the enclosure, with the farthest component getting assembled first. This approach of sampling a parameter given other parameters is similar to Gibbs sampling [37].

Note that our proposal distributions are symmetric ($Q(\theta_t \mid \theta') = Q(\theta' \mid \theta_t)$), which further simplifies eq. 3.21.

Tuning PT sampler

The influence of hot chains on the colder chains, can be maximized by increasing the speed with which the chain values move along in the inverse-temperature domain [17]. To enable this, previous research suggests selecting the temperature ladder so as to achieve 23% swap acceptance rate (eq. (3.22)) [17, 113]. Inspired by [187], we adapt the temperature of chains during sampling to achieve this swap rate of 23% amongst each adjacent pair of chains. For this adaptation, we first initialize inverse-temperatures with a geometric temperature sequence: $\beta_{j+i} = \rho\beta_j$, where ρ is a constant. ρ can be easily determined given the number of chains N and maximum chain temperature β_N. We use $N = 10$, and $\beta_N = 0.001$ for all our experiments (empirically determined).

Interleaving gradient optimization with PT

Since the cost function $J(\mathbb{D})$ is multimodal, we want the chains to quickly explore and find the modes of $J(\mathbb{D})$. To drive the random walk of these chains towards regions of high probability (modes) in the manner of a gradient flow, we utilize gradient information for the continuous parameters in θ_t^k. Keeping the discrete parameters fixed (assembly ordering), θ_t^k is updated using single step of gradient descent in each iteration t (line 13 in Algorithm 1).

$$\theta_t^k = \theta_t^k - \gamma \frac{\partial J(\mathbb{D})}{\partial \theta}\bigg|_{\theta=(p^l,p^a)\in\theta_t^k}, \tag{3.23}$$

where $\frac{\partial J(\mathbb{D})}{\partial \theta}$ is a numerically computed gradient, and γ is the gradient step-size determined by line search. p^l and p^a are the continuous layout and assembly DOF respectively (eq. 3.13, 3.14).

Incremental interleaved optimization

Since our approach uses incremental design heuristics and interleaves gradient optimization with PT, we call it an *incremental interleaved optimization*. Partial designs are created incrementally by adding one component at a time based on their sizes, starting with larger components first. Each partial design is then optimized with interleaved gradient-PT optimization, before updating the partial design by adding the next set of components. In order to make better use of available empty space while adding a component to a partial design, and to utilize previously found valid design, we formulate a new initialization procedure for PT chains, as described next.

Starting with a partial design $\mathbb{D}_{partial}$, and the current component to add (c_{add}), the initialization algorithm sets up N chains for our interleaved optimization. Half of the chains (at lower

Algorithm 1: Incremental interleaved optimization for layout and assembly design of electromechanical devices

input : $\mathbb{D}_{partial}$, c_{add}, $threshold$ for convergence
output: $\mathbb{D}_{partial} \cup c_{add} \mid J(\mathbb{D}_{partial} \cup c_{add}) < threshold$

Initialize N chains with $\beta_1, \ldots, \beta_N$
for *chain* k **do**
    if k ¡ $\frac{N}{2}$ **then**
        $\theta_0' \leftarrow$ Sample 50 configurations for c_{add}, keeping $\mathbb{D}_{partial}$ fixed
        $\theta_0 * \leftarrow$ Pick the best configuration out of θ_0'
        Intialize chain k at $\theta_0 *$
    else
        Intialize chain k at random
    end
end

while *not converged* **do**
    foreach *chain* k **do**
        gradient step for continuous variables (eq. (3.23))
        Metropolis-Hastings (MH) update of chain k (eq. 3.21)
        if $J(\mathbb{D})$ *at* $\theta_t^k < threshold$ **then return** converged
    end
    swap a random pair of adjacent chains
    update chain's β_k periodically
end

end of temperature spectrum) are initialized to exploit the configuration of c_{add} around the previously configured components in $\mathbb{D}_{partial}$. In order to increase the probability of adding c_{add} in the available empty space, we sample 50 configurations (empirically determined) of c_{add} without changing previously configured components in $\mathbb{D}_{partial}$, and pick the best one. While there is no guarantee that c_{add}'s configurations falls in an empty space in 50 samples (and this probability goes down with higher number of components and fill ratio), in practice, just being close to an empty space is helpful enough. This is because if c_{add} is initialized near an empty space, the perturbations during the interleaved optimization will end up pushing it in the empty space. Such an initialization serves as a hypothesis for possible configurations of c_{add}, and the corresponding chain refines it further as the optimization proceeds. On the other hand, the chains at higher temperatures are initialized randomly. They search for completely different configurations for all components in $\mathbb{D}_{partial}$ including that of c_{add}. They are meant to handle situations that require major re-configurations of the previous design to accommodate c_{add}.

The incremental interleaved optimization algorithm is outlined in Algorithm 1 and is used for adding c_{add} to $D_{partial}$ at each stage in the design. We begin the PT sampling process at $t = 0$, with N chains initialized at $\beta_1, \ldots, \beta_N$ temperatures. The initial configurations of chains in the parameter space are obtained using our initialization procedure (line 2-10). Each sample θ_t^k of chain k at time t, consists of $\{L(\mathbb{D}), A(\mathbb{D})\}$. Lines 11-19 correspond to our interleaved gradient-PT approach. In order to maximize the influence of hot chains on the colder chains for faster convergence, we also update the chain temperatures periodically (line 18) using a procedure described above for tuning PT sampler. The algorithm converges when any chain's sample values correspond to a design with $J(\mathbb{D}_{partial} \cup c_{add}) < threshold$. This threshold is set so as to ensure a collision-free design.

Such incremental design enables the overall optimization process to be much more effective. This is because partial designs have fewer parameters to optimize (smaller design space), and a less constrained volume available for layout. Further, when the optimization is re-run with the next set of components, partially optimized designs result in more favorable initial conditions. Interleaved optimization without our heuristics (incremental design, and use of empty space) lead to much longer optimization times. For instance, the average design time for one of our test devices – Clumsy (Fig. 3.9) comes out to be 706.7s over 10 runs using only interleaved optimization, instead of 214.14s with incremental interleaved optimization.

3.7.3 Role of users in design optimization

When the optimization freezes for more than 5 min (empirically decided), our system asks the users to make small modifications and rerun. Problematic components are highlighted in red, and users tend to modify those, helping the escape from minimas.

3.8 Fabricated examples

We designed and fabricated three devices with very different functionalities to demonstrate the utility of our system – a four wheeled robot called *Crusher*, a two wheeled balancing robot - *Clumsy*, and a smart crib monitoring toy owl – *Chirpy* (shown in Fig. 3.1, 3.9). Crusher is a

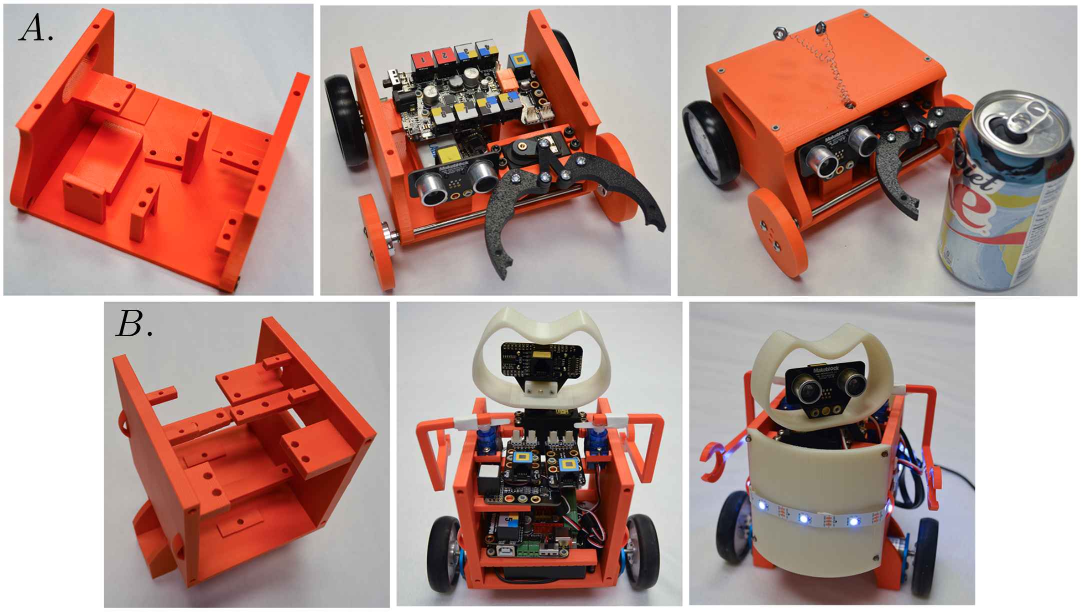


Figure 3.9: Fabricated devices – (A) Crusher, and (B) Clumsy are shown with their 3D printed enclosures, and their final assembled design. Each enclosure has custom mounts and fastener geometries created by our system for their components, based on the optimized layout.

bluetooth controlled recycling robot that can detect and grab soda cans with its gripper arm. Clumsy freaks out when it detects obstacles and tries to ask for help by waving its hands. Chirpy can detect and soothe a crying baby. Chirpy also alerts the baby's parents by sending a message when the baby starts crying. Our video[2] shows these robots in action. Each device has a variety of components and unique enclosures. While our system is not restricted to any particular type of kit or modules, we use electromechanical components from Makeblock kits for our examples [130]. Relevant information about fasteners and mounts for each component is pre-processed manually, but one can envision scanning a catalog to gather this information.

To endow devices with a desired functionality, we selected a set of components for each device. The configurations of certain components may be limited within an enclosure owing to the functional and aesthetic requirements of the device. For example, Crusher's motors need to be configured so as to connect to its wheels, while the LEDs for Chirpy should be placed near its eyes. We therefore pre-specify the configurations of such components before generating the assemblable layout of other components using our interleaved optimization (Sec. 3.7.2). The optimization process maintains the layout of pre-specified components, and optimizes for their assembly while concurrently optimizing the layout and assembly parameters for all other components. For each layout, our system also generated mounting structures. These are integrated into the original 3D model of the enclosure using Constructive Solid Geometry (CSG) operations. The result is a 3D printable enclosure custom-designed for specific functionality. We fabricated our designs using a Stratasys uPrint SE Plus, a filament based 3D printer using ABSP430 plastic as the model material and a dissolvable support material. The fabrication time varied from a few

[2]Available at – `https://youtu.be/DrIXD5Fpg0I`

hours to a day, depending on the geometric size of the enclosure. In contrast, the average time to design a device with our system ranged from 2-15 minutes (more details in Fig. 3.11), and the assembly time of each device was less than 15 minutes. Each enclosure also had single or multiple lids that were printed separately and attached to the enclosure after assembly, through rivets.

While each of the three devices that we fabricate look simple due to small number of components, they are representative of the types of electromechanical devices that are found in community-driven online repositories made by DIY enthusiasts, and hobbyists (our target audience). We base this observation on an informal survey of 3D printable devices that we conducted on two such popular platforms – Instructables and Thingiverse [99, 215]. We found that these designs used on an average 7 components (average over 70 designs of robots, IoT devices etc). While these designs may not be from casual makers, they reflect what the maker community is interested in building. Further, based on the findings of our user-study, devices with 8 components are already quite difficult, and time-consuming for non-experts to design manually.

3.9 User-studies

We conducted two types of user-studies with different goals in mind – (a) an exploratory interview cum observation-based user-study with an expert to gauge the challenges of the design task, as well as to gain insights about design strategies, and (b) a detailed user-study with novices to understand the utility of our system.

3.9.1 Exploratory study with an expert

Solidworks is a CAD tool of choice for creating 3D designs, and assemblies of complex artifacts such as the ones our system helps to create [196]. To understand how experts design 3D devices using available tools, we sought help from an expert user with five years of experience with Solidworks. We not only interviewed the expert about the design process to get an idea about the expert's workflow, but also asked the expert to design Crusher and Clumsy using Solidworks, starting with 3D models of the enclosure and the necessary components. The expert reasoned about assembly while doing component layout during design, thereby supporting our concurrent design approach. As discussed in our optimization section (Sec. 3.7.2), we also found that the expert executed such simultaneous design for components, one at a time. In other words, the expert dealt designed in an incremental manner, inspiring us to integrate similar heuristics within our automated approach.

The expert also noted that the hardest part of the design was deciding the layout of certain components. For instance, the expert specified that the hardest part for Clumsy's design was deciding the placement of two of its biggest components. Consequently, the expert explored multiple layout configurations before finalizing the layout design. Apart from layout configurations, the expert spent a major portion of the design time in designing component mounting structures, and re-designing them during layout re-configurations. This is because appropriate mount and fastener geometry design need to done manually in Solidworks. Further, these geometries also need to be updated manually when components are re-configured during layout design.

As a result, the expert took 4320 seconds (70 minutes) to design Crusher, and 5400 seconds (90 minutes) to design Clumsy. These design times are even longer than the times taken by casual users to design these devices manually with our system (see Fig. 3.11).

The expert's design approach however, also highlighted certain limitations of our system owing to the assumptions that we impose. Firstly, the expert designed *shared* or *common* mounting structures that could support multiple components at once. This allowed more flexibility for component layout. Secondly, the expert also reasoned about space for wires, as well as space needed for hands during manual assembly, which we currently do not account for.

3.9.2 User-study with casual users

For further validating the need and usefulness of our computational framework, we conducted a user-study with 24 paid novice participants. The user-study had two goals. First, we wanted to understand and quantify the difficulty that novices face while manually creating assemblable layouts for electromechanical devices. Secondly, we wanted to determine if our system reduces the entry barrier they face in creating such devices.

Participants

All participants were undergraduate or 1st year CS graduate students (7F, 17M). We define a casual user/novice as someone who may be interested in building devices but does not know how to use a CAD tool, and is unaware of the assembly procedures (e.g., accounting for fasteners) required for creating a feasible design. To ensure that our participants belonged to our target user group, we asked the participants 2 questions about their background and interest in building devices in the user-study survey – 1. Are you interested in building/making things? (Answers: Yes/No/Maybe), 2. What is your expertise with CAD tools for 3D design? (Answers: 5-pt Likert scale with score 1 = no expertise). Only 1/24 participant reported about not being interested in making things, with 70% replying with a definite yes. All 24 reported none or slight CAD expertise (average: 1.5 likert score).

Study structure

Each user-study session lasted 75 minutes, and consisted of an introduction and training session (25 minutes), followed by a design task (45 minutes), and concluded with a survey (5 minutes). Each participant was asked to design assemblable component layouts for one of either Crusher, Clumsy or Chirpy within 45 minutes in the design task. The introduction and training session were responsible for familiarizing the participants with the user interface and the overall task. In order to boot-strap the participants into thinking about the constraints of the design task, the experimenter and the participant co-designed layout of a set of components within a box enclosure, during the training session. For the design task, the participants were explained the functionality of the device they were creating, and were provided with a list of components to use according to the device's functionality. We also provided them the device enclosure and the configurations of the components that need to be pre-fixed (such as motors and LEDs). In other words, the participants were given the same input as taken by our optimization. We recorded whether the participants succeeded in creating a valid design in their alloted time, and the total time taken by the participants to create such a design in case they succeeded. The survey then evaluated their perception of the task difficulty, and the capabilities of the system by asking them to rate various aspects of the design process using a Likert scale. Their feedback on the survey provided us with a qualitative understanding of their design experience.

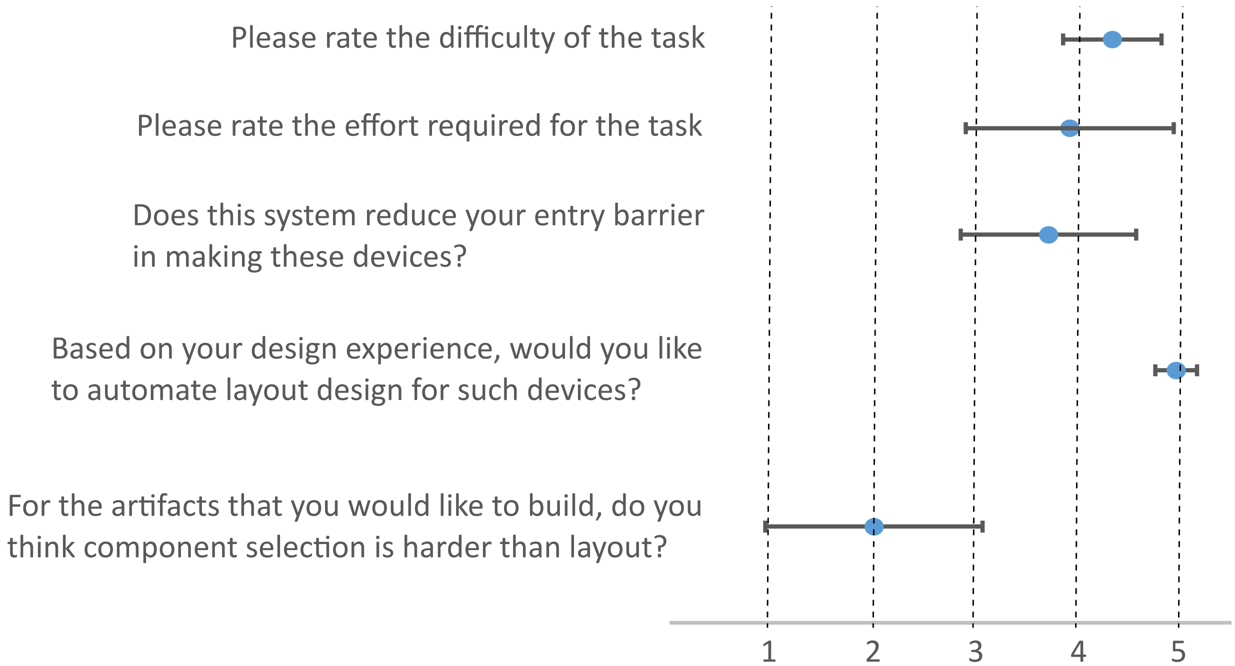


Figure 3.10: User design experience – Users feedback about the design task, and our system are highlighted by their responses to our survey. 1 - strongly disagree/very low, 3 - neutral, 5 - strongly agree/very high. Error bars indicate standard deviation.

3.9.3 Qualitative analysis

Participants found our highlighted guidance, automatic mount creation, and animation features highly useful (Fig. 3.4). For instance, a participant P7 reported that *"assembly animations were very useful, without them layout would be hard."*. Aided by these features majority of participants succeeded in creating device designs (Fig. 3.11). Inspite of this, all 24 participants rated the design task to be difficult or very difficult, and supported the utility of automatic design mode. Participants reported that they would either do physical mockups with iterations (4/24), use pen-paper (2/24), attempt to learn CAD (9/24), or wouldnt know what to do (9/24); for designing such devices if not for our tool. The use of incremental design strategy was also seen in some participants. While all participants reasoned well about the space in the device for layout, most of them struggled to make assembly considerations (e.g., P1: *"at first I was only focusing on layout and only then I realized that I should think about [assembly] order."*). Fig. 3.10 shows user responses to our survey questions on Likert scale and provides more insights into the user experience with our system as well as into their perception of the task complexity.

3.9.4 Quantitative analysis

Figure 3.11 shows the design time and success rate measured during the user-study for each of the devices (orange bars). These statistics emphasize the difficulty of manual design process, inspite of our system features. Even for devices with 8-10 components, manual layout is challenging, especially when assembly considerations are taken into account. The average design time increases, and the success rate decreases as the complexity of devices increased. Chirpy and Clumsy were on the lowest and highest end of the perceived complexity spectrum respectively. In particular, only 3 out of 8 participants found a valid design for Clumsy in the allotted time. In order to compare these statistics with that of the automated design process, we ran our optimization 10 times for each design (resultant statistics shown with gray bars). Each such experiment was run for 1000 seconds or until the threshold cost of a valid collision-free design was achieved (see algo. 1). We ran all our experiments on a standard desktop with a 3.6 GHz i7 CPU and 16 GB RAM. Similar to the user-study, the success rate for the optimization indicates whether a collision-free layout and assembly plan was found in the allotted time. For the cases where it failed to find a valid design in 1000s, the optimization process becomes trapped in local minima. We found our automated approach to be much faster and successful than the manual design.

3.10 A discussion on device complexity

Even though Clumsy has the same number of components as Crusher, there is a significant difference in their complexity, as evident in the design time and success rate statistics. We therefore attempt to quantify the approximate complexity of these devices using a set of features (table 3.1). In particular, we use – a) the number of parameters, and b) the fill ratio which is defined as the relative volume occupied by all components and mounting structures. The fill ratio is computed for each valid design. Note that the number of parameters and the fill ratio capture different aspects of complexity.

Device	$\mid C \mid$	$\mid F \mid$	Avg. fill	# p (discontinuous)	# p (continuous)
Crusher	8	18	30.5%	8	34
Chirpy	10	20	21.9%	10	47
Clumsy	8	19	45.2%	8	33

Table 3.1: Quantifying design complexity of 3D printable devices with embedded electromechanical components - The number of components ($\mid C \mid$), and fasteners ($\mid F \mid$), fill ratio, and number of discrete and continuous parameters to optimize (# p) indicate approximate complexity of each design. The complexity also depends on the number and configuration of fixed components since they pose extra constraints.

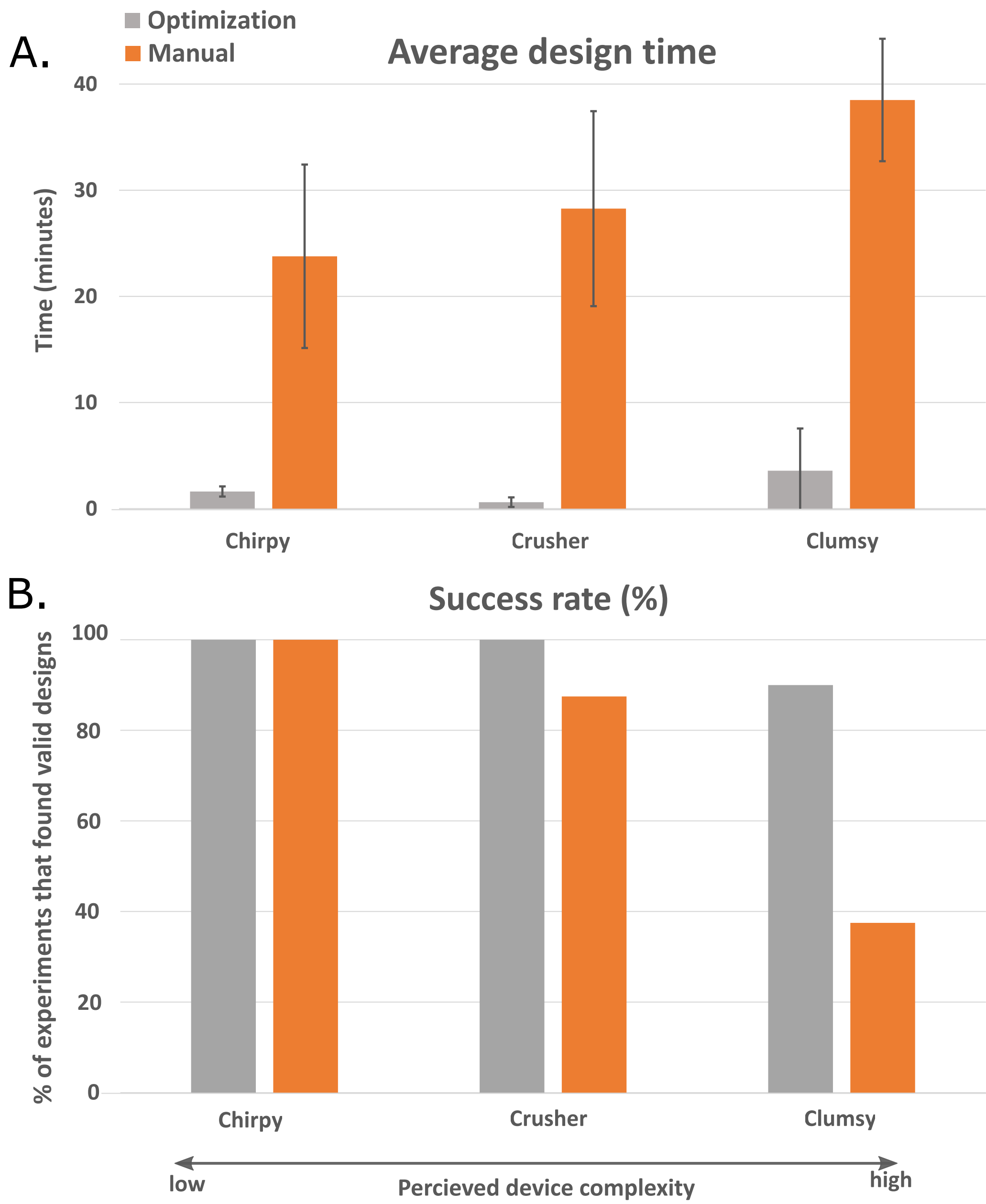


Figure 3.11: User-study statistics – A) The average design time in minutes, and (B) the success rate for each of our 3 devices are shown. The orange bars represent these statistics averaged over 8 participants per device during our study, while the gray bars correspond to those averaged over 10 runs of incremental interleaved optimization. Error bars indicate standard deviation.

While the features in table 3.1 provide an approximate idea of device complexity, it is hard to precisely quantify the difficulty of finding an assemblable layouts. This is because the complexity depends upon not just the number of components, volume filled, and parameter count, but is also a function of many other factors. In particular, the shape of the enclosure dictates possible layouts and assembly paths for the components. However, the enclosure shape is not characterized by its volume. Out of two enclosures with the same volume, the enclosure with a shape that provides more surface area to mount the components may be more amenable for layout. Further, when a component is added to an enclosure, the component and its mounts partition the space available for other components in a non-trivial manner. Fasteners, and configurations of locked components such as LED further shape the remaining available space. The role of these factors becomes apparent in the design of Clumsy. Even though Clumsy has only 8 components, 2 of its components – a battery pack, and a controller board are very large and can be arranged in only certain configurations so as to fit within the enclosure. If their configurations are badly initialized, the smaller components may block them from achieving these valid configurations (as reflected in the less than 100% success rate of our optimization (gray) in Fig. 3.11).

3.11 Validation

We validate our framework in three ways. First, we show examples of device designs with arbitrary electronic components and enclosure shapes to demonstrate the versatility of our system. Next, we validate the advantage of interleaving gradient-based optimization with PT (Sec. 3.7.2) for our problem, by comparing with other standard sampling and gradient based methods. Finally, we describe a benchmarking experiment that shows the scalability of our framework with increasing number of components, and fill ratio.

3.11.1 Virtual device examples

Chirpy, Crusher and Clumsy, have cuboidal enclosures. However our system works for any convex-shaped enclosure. Fig. 3.12 shows devices with a polygonal enclosure, a trapezoidal enclosure with slanted walls, and a bunny-shaped enclosure. Each device contains arbitrary electronic components of varied sizes selected randomly. The trapezoidal enclosure has openings on the top and the bottom, while the polygonal and bunny-shaped enclosures have one opening on the top, and one on the sides. While our current implementation only supports convex-enclosures, our framework will work for concave enclosures as long as we have a way to compute distances from the enclosure. Note that manual layout design for devices such as our trapezoidal and polygonal examples is difficult, owing to the large number of components, and corresponding constraints. This is one of the reasons why the number of components in devices created by non-experts ranged from 2-20 in our informal survey of such designs.

3.11.2 Comparison with other optimization approaches

In order to validate the advantage of interleaving gradient optimization with Parallel Tempering (PT), we compare against other possible variants – ‘PT only’, ‘PT followed by gradient

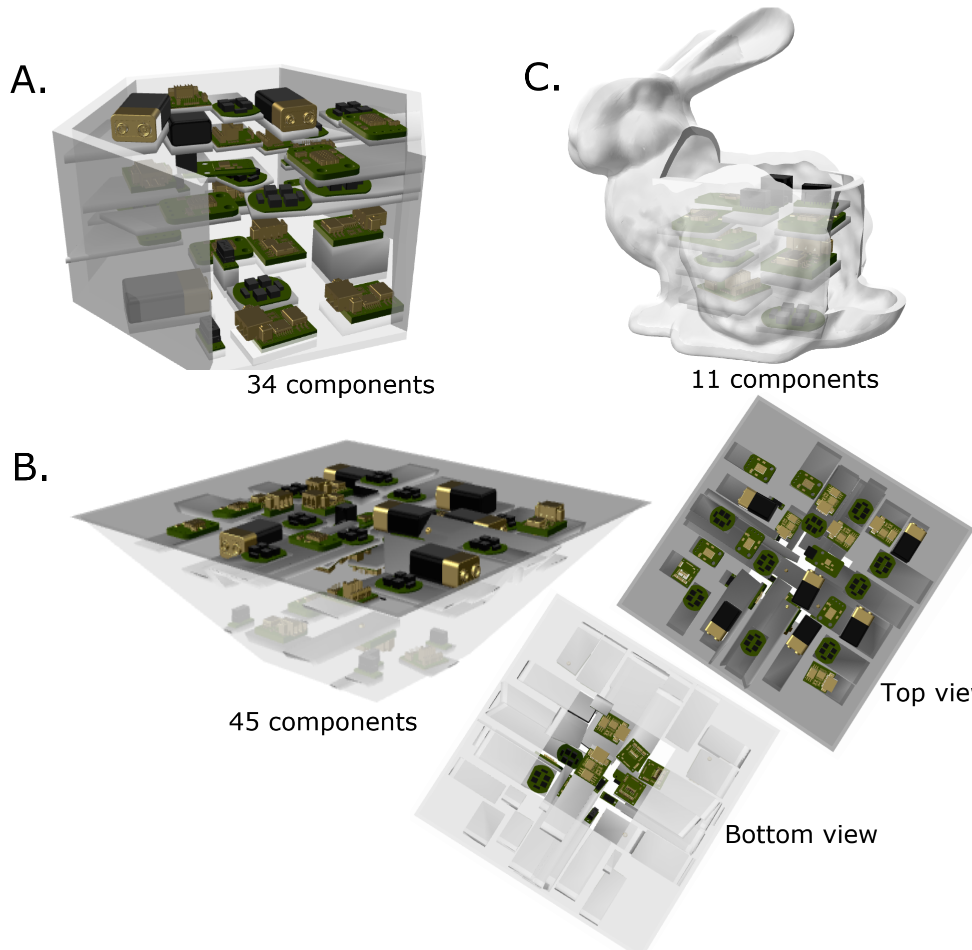


Figure 3.12: Devices with differently-shaped enclosures and components – (A) A polygonal device, (B) a trapezoidal device, and (C) a bunny-shaped device, each with arbitrary electronic components are shown.

optimization', and 'gradient optimization only'. We use the BFGS quasi-Newton method for gradient optimization in the last two test conditions [151]. 'PT only' has been successfully used for layout problems in the past such as for furniture layout [141], while BFGS is widely used for continuous optimization problems [151]. 'PT followed by gradient optimization' combines stochastic and gradient based optimization in a naive manner. Table 3.2 shows the comparison of our interleaved optimization against these variants for the design of Crusher averaged over 10 runs. Considering that the interleaved optimization for Crusher design found a valid solution in 154s on average, we ran PT for 500s or until the cost threshold of collision-free design was reached, for 'PT only', and 'PT followed by gradient optimization' test conditions. This was followed by 100s of gradient optimization for the latter. In order to replicate the random initializations and N parallel chains of PT, we execute $N = 10$ parallel gradient optimizations, each starting with a random configuration for the last test condition. Since the gradient optimization cannot be used to find a discrete assembly ordering, we set the ordering based on the initial configuration in a heuristic manner (as described in Sec. 3.7.2).

Optimization	Avg. time (s)	Success rate
Interleaved optimization (ours)	154.7	100%
PT only	-	0%
PT followed by BFGS	539.6	80%
BFGS only	493.7	40%

Table 3.2: Comparing optimization techniques for computing 3D printable device designs - We base the comparisons on how often the techniques find valid designs, and the time they take to find them. We ran all our experiments on a standard desktop with a 3.6 GHz i7 CPU and 16 GB RAM.

The success rate of this experiment is an indicator of the probability of finding an assemblable layout given a fixed ordering. It reaffirms the need to search for an assembly ordering and layout concurrently. The interleaved optimization strategy finds valid designs in lesser time with higher success rate compared to other methods. 'PT only' does not manage to find acceptable designs in 500s for any runs.

3.11.3 Scalability experiments

In Sec. 3.9.4, we described how quantifying complexity of 3D printable electro-mechanical devices is non-trivial. Nevertheless, to study the ability of our proposed algorithm to scale to complex examples, we focus on two features – fill ratio, and number of components. We select these features because they are easy to quantify, and control in an experimental set-up.

For each experiment, a virtual device is created with fill ratio m, and n cuboidal components of arbitrary sizes. To reduce the affect of enclosure and component shapes on the device complexity, we use simple cuboidal enclosure and cuboidal components, for all our experiments (Fig. 3.13(A)). Further, we assume that the enclosure's size and thereby volume can be appropriately scaled as needed. Scaling the enclosure in this manner allows us to easily control a device's fill ratio for our test scenario.

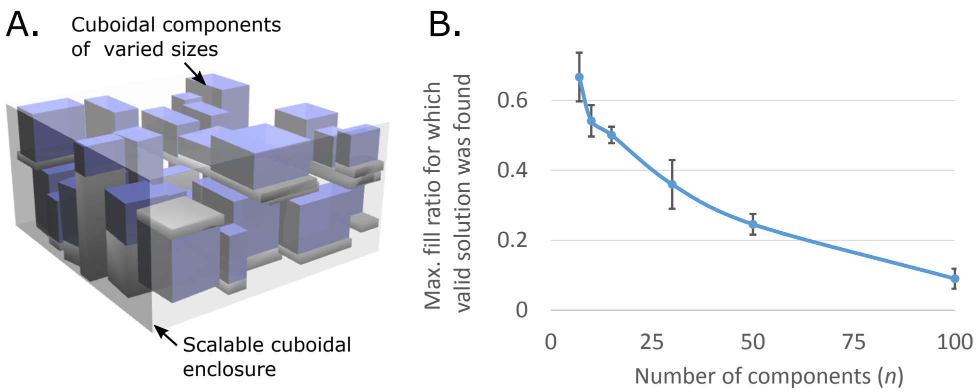


Figure 3.13: We perform scalability experiments that measure the performance of the algorithm as the number of components, and fill ratio increases. (A) shows an example virtual device with arbitrary sized cuboidal components that we create for these experiments. A scalable cuboidal enclosure is used to easily increase the fill ratio during the experiments. (B) gives an intuition about the maximum fill ratio of devices with n components for which the algorithm was able to find a solution in the allotted time.

In order to test the scalability of our framework with increasing fill ratio, we keep the number of components n in a device constant as we change the device's fill ratio m. Similarly, to test how the framework scales with increasing number of components in a device, we keep the fill ratio fixed as we increase the number of components in the device. During the course of first experiment, we gradually scale down the enclosure volume to increase the fill ratio for a device with n components. We run incremental design optimization to find valid assemblable layout designs, for an hour for each fill ratio. We continue increasing the fill ratio till the optimization fails to find a valid device design in an hour, and record the maximum fill ratio for which the optimization succeeded. For testing scalability with number of components, we repeat the above experiment for different values of n (number of components). Fig. 3.13(B) shows a plot of maximum fill ratio for which the optimization found a valid design in the allotted time vs. number of components for our virtual cuboidal device, calculated over set of 3 experiments. As the number of components or fill ratio increases, the optimization needs more time, and hence finds fewer valid designs in the allotted time.

3.12 Frequently asked questions (FAQ)

1. **What exactly is the target audience for this system?**
   Our target audience is school students, artists, DIY enthusiasts, and novices. We define a casual user/novice as someone who may be interested in building devices but does not know how to use a CAD tool, and is unaware of the assembly procedures (e.g., accounting for fasteners) required for creating a feasible design.
2. **Do novices really need/want such a design system?**
   During our user-study, we asked the participants about various possible design alternatives that they would use for designing such devices. Participants reported that they would

either do physical mockups with iterations (4/24), use pen-paper (2/24), attempt to learn CAD (9/24), or wouldnt know what to do (9/24); for designing such devices, if not for our system.

3. **Can the system handle concave enclosures?**
Our implementation relies on collision detection that operates on convex primitives, but our framework will work with concave objects if collision computation is available for the same.

4. **Why do we need to create rigid mounts for components?**
While one can make-do with loose mounts for some components such as controller or battery, majority of components – accelerometers, light and range sensors, sliders, panels, buttons, LEDs, motors work better when mounted properly. Towards this, our goal is to leverage customization capabilities of 3D printing for creating better devices with rigid mounts for components.

5. **Designing physical devices completely virtually sounds unintuitive. Why not use physical mockups for designing, instead of this system?**
Physical mockups may provide good intuition to the users, but assembly and fastener integration could still be hard to visualize as evident from participant quotes on assembly animations that were extensively used in manual design. Further, experts also design things virtually in CAD before making mockups. There is however an interesting opportunity of assembling digital mockups in virtual reality (VR) for better spatial intuition as a pre-design step, before creating final fabricable designs using our system.

6. **How is this system's computational approach different than existing layout design approaches?**
Unlike existing layout approaches [78, 141, 237], we are jointly optimizing both layout and assembly process. The corresponding spatio-temporal model demands a different computational approach than problems that only consider layout. Specifically, in addition to the need to model the temporal aspects of the design, most layout problems are continuous, unlike ours, which has both discrete and continuous parameters. Inclusion of fasteners and mounts in the optimization also makes it more challenging than standard layout problems. As a result, we had to leverage both design heuristics based on expert design strategies as well as efficient numerical techniques such as combined stochastic sampling and gradient-based optimization for our design problem.

7. **What sort of components does the system support?**
We are currently targeting hobby-grade components, because of their ease of use, affordability, and accessibility.

8. **Isn't providing components for a desired device functionality as input, hard for novices to do? How is this accessible?**
Choosing the right set of components for a target functionality does require domain knowledge. This challenge is however complementary to the problem we address, and it is actively being investigated in parallel [168]. Further, as the users indicated in our study, component selection is perceived to be easier than designing with these components by our target audience (see Fig. 3.10).

9. **Which assembly tools are supported by the system?**
   Our system currently accounts for assembly with tools such as screwdrivers and hex keys.
10. **Can the system account for assembly order preferences, such as keep the battery to be most accessible during device assembly?**
    While we currently enforce no constraints in the assembly ordering of components, our formulation can easily account for such assembly order constraints by simply discarding orderings that do not satisfy such constraints during sampling.
11. **Does the system account for structural strength during device design?**
    We currently do not account for structural strengths of mounts generated. Most electronic components are lightweight. Currently many novice prototypes on Thingiverse and Instructables have loosely connected components assembled with hot glue or tape, suggesting limited strength requirements. Thus, mounts with a certain thickness (empirically determined using the yield strength of PLA and the maximum weight of components) spanning components fastener holes, as in our system, are sufficient. However, it will be important to perform structural optimization of mounts for making robust designs in the future.
12. **Why not optimize for enclosure design itself instead of optimizing component layout design for creating such devices?**
    Approaches that optimize/snap enclosure walls to bound the given components have been developed in the past specifically for laser-cutting based fabrication methods [230]. Our approach complements these approaches. In particular, we are interested in design scenarios where the enclosure shape is fixed due to aesthetic or fabrication constraints such as available volume of 3D printer, which may limit scaling or modification of the enclosure.
13. **Are the system generated designs easy to assemble in reality?**
    The DFA guidelines (Sec. 3.4.2) ensured that the resultant device designs were amenable to manual assembly. However, since we did not model space for hands/fingers during assembly, some of our fabricated prototypes were slightly cumbersome to assemble in our experiments. This could be dealt in a conservative manner by increasing the minimum threshold distance (ϵ in eq. 3.17) between the components during optimization. A secondary user-study that purely evaluates assemblability of generated device designs by asking a group of novices to assemble these devices might also be helpful for qualitatively measuring device assemblability in the future.

3.13 Limitations

Limiting assumptions: Our system is currently focused on non-articulated devices without transmission, multiple joints or moving parts. Extending our framework to account for moving parts that take a range of configurations once assembled will enable the design of more diverse set of devices. Including other higher level design requirements (such as a desired center of mass for Clumsy) while optimizing for component layout may further increase the space of designs. Currently, we also assume separate mounting structure for each component. As shown by our expert study (Sec. 3.9.1), enabling the design of shared mounting structures between components may lead to better use of space, and may be worth exploring for devices with higher complexity (large number of components or higher fill ratio).

Although we follow DFA guidelines, we do not consider wire routing and insertion or the space needed for hands during the assembly process. As a result, certain parts of our devices were a bit cumbersome to assemble. In particular, we found that experts do account for some of these practical design considerations while designing assemblable devices (see Sec. 3.9.1). In the future, accounting for these aspects would therefore be essential for enabling novices in creating even better interactive artifacts.

Finally, our tool currently only provides a single valid device design to the user. Providing alternate designs might be important to support user-creativity [192]. Alternate solutions may also be necessary for scenarios wherein valid solutions exist only on a pareto-optimal front. For instance, when other auxiliary device objectives such as device weight distribution and 3D print material minimization are added to $J(\mathbb{D})$ (eq. 3.15), different solutions that trade-off 3D print material and weight distribution through different component layout may exist.

Handling more complex devices: We also found that it is challenging to quantify design complexity of such devices. As a result, guaranteeing a solution for any arbitrary device is non-trivial. Further, owing to the nature of stochastic optimization, ensuring a valid solution in stipulated time, which might be important for an interactive design system, is also currently not possible. While our experiments on incremental design were promising, our scalability experiments showed that the optimization needed much longer time to find valid designs for devices with more than 15 components, or higher than 50% fill ratio. The number of components and fill ratio in devices made by our target audience of makers and artist is well below these bounds. However, more research is necessary to enable our system to aid experts or other target audiences. To this end, a promising approach entails putting user-in-the loop during incremental design, as well as exploring better ways to incorporate user intuition during design.

3.14 Publication and dissemination

This work has been accepted for publication at ACM Symposium on User Interface Software and Technology (UIST) 2018. Once published, we will also make the files of fabricated devices publicly available. Yincheng Zhao from Tshingua University helped with the design of Clumsy as a summer intern in the lab.

An overview video about our system is available at – `https://youtu.be/DrIXD5Fpg0I`

Chapter 4

Modular structure design for articulated robots

4.1 Preamble

We now describe our second structure design tool for articulated robots consisting of series of interconnected components. It features an intuitive interface, a powerful model encapsulating necessary domain knowledge, modeling of fabrication constraints, support for design space exploration, and accessible two-way feedback between users and the system powered by a physics-based simulation. Aided by these capabilities, our tool enables the creation of a wide variety of robotic devices (Figure 4.1).

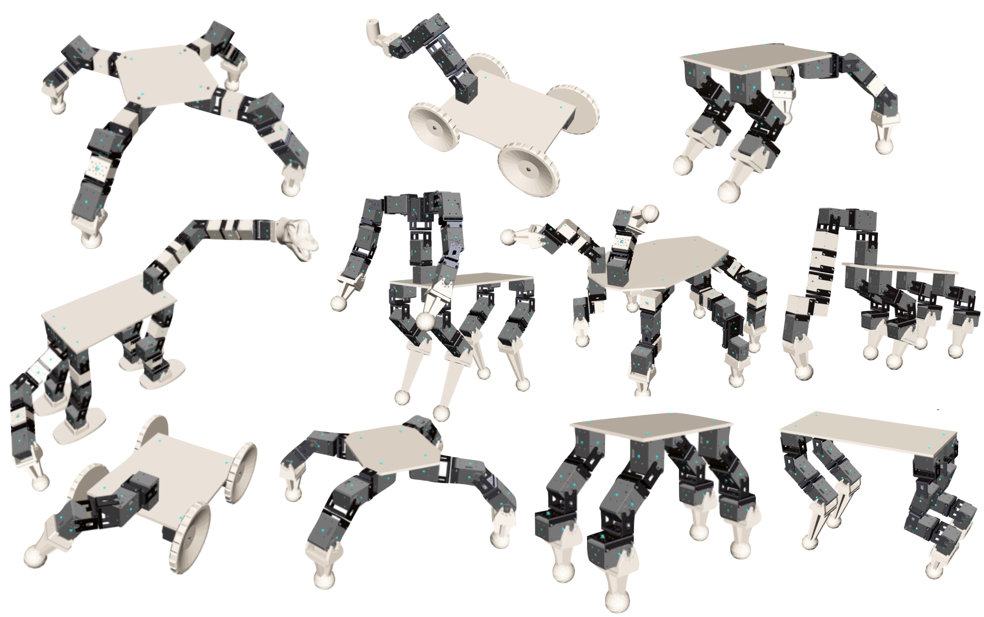


Figure 4.1: Preview of robots build with our modular structure design tool – Using a set of only nine modular components (four off-the-shelf components, and five 3D printed components), we are able to design a plethora of diverse robots.

4.2 Introduction

The task of designing an articulated robot amounts to choosing which components to use, how to combine them into a functional system, and how to control the resulting robot in order to achieve a desirable set of behaviors. Our modular structure design tool is an interactive computational system for assisting users with each of these tasks. It enables easy configuration and deployment of robots using a library of standard building blocks such as actuators, mounting brackets, and 3D printed parts.

Our system is based on a formal design abstraction that models the way in which modular building blocks can be combined to form complex robotic systems. In particular, our abstraction presents a way to represent each component in terms of geometric features and virtual pins. The virtual pins establish compatibilities between different components and define the set of possible physical connections between them.

This design abstraction is leveraged within the visual design environment to support manual and semi-automatic design modes. During the manual mode, the virtual pins are used to suggest valid placements for each new component, and thus enable an intuitive exploratory design process. For semi-automatic design, the design abstraction is used to automatically generate assemblies of structural components that connect pairs of actuators whose desired relative placement is user-specified. These intermediate designs are obtained by efficiently searching through the space of possible arrangements of modular components.

Finally, our design system provides a physical simulation environment where users' robot designs can be tested before fabrication. In particular, we employ an existing optimal control method to generate physically-valid motions for legged robots of arbitrary designs [134]. The resulting motions are tracked in simulation using Proportional-Derivative (PD) control. By providing feedback at interactive rates, users of our system can therefore iteratively adjust their design to best meet their individual needs and preferences. Figure 4.2 gives an overview of our tool, highlighting its capabilities.

To evaluate the scalability of our approach, we employ a library of modular components consisting of Dynamixel actuators [176], off-the-shelf brackets, and 3D printable connectors. We demonstrate the effectiveness of our tool by creating an assortment of legged and wheeled robotic devices. We validate the physical feasibility of our results by fabricating two of the robots generated with our design system – a wheeled robot with a manipulator arm and a quadruped robot.

4.3 Related work

We are inspired by the ease of use and flexibility of the Spore Creature Creator [95], which comes from the computer graphics literature. To date, close to 200 million customized animated characters have been created with this system by non-expert users from around the world [72]. Our vision is to make the process of creating customized robotic devices equally powerful and accessible.

Robot design tools: A plethora of commercial as well as academically developed designs tools

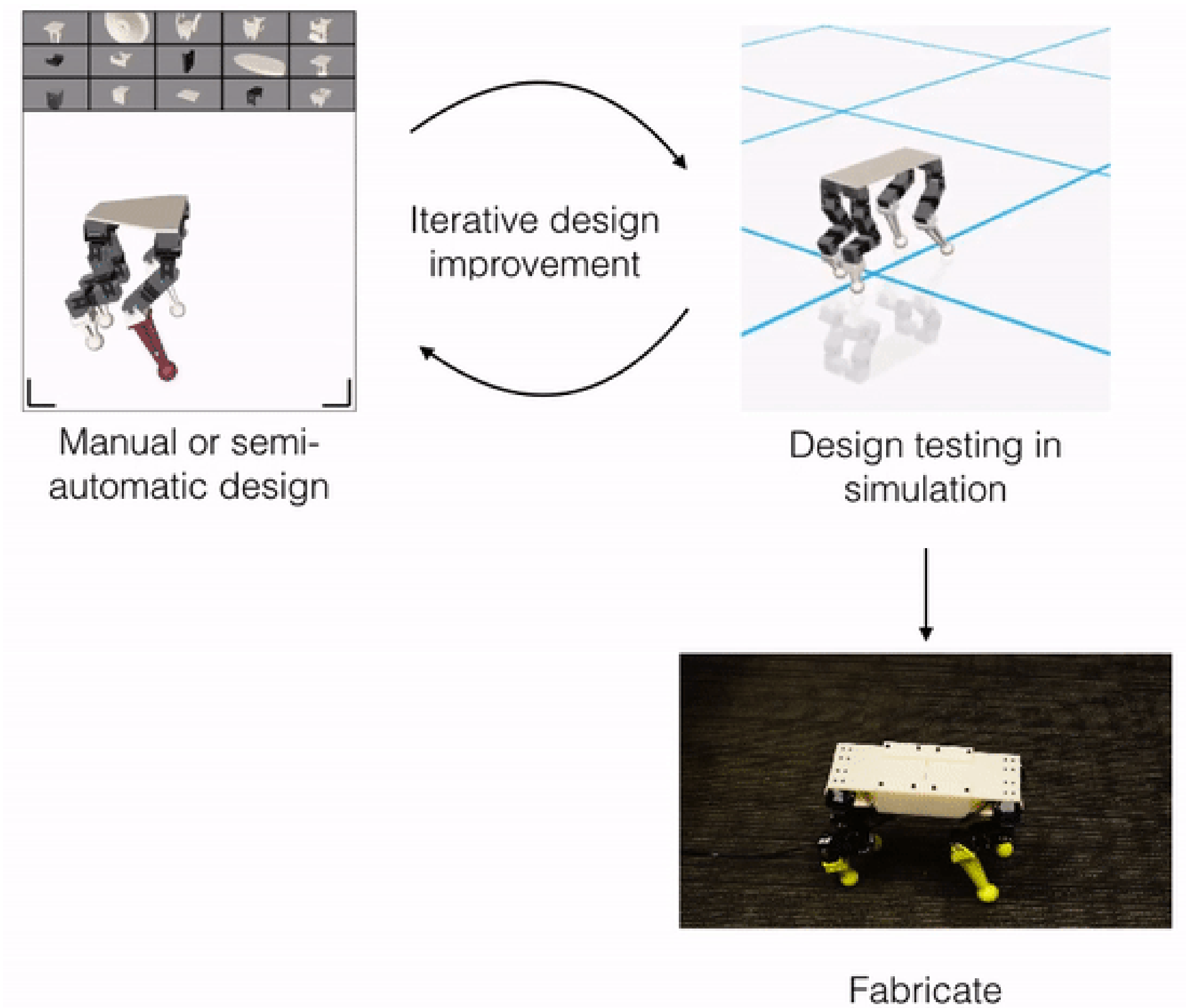


Figure 4.2: Overview of the modular structure design tool – An intuitive visual environment supports manual and semi-automatic modes for creating articulated robots. Designs can be tested in physical simulation, and iteratively improved based on simulation feedback, before fabrication.

are available for robot design (Chapter 2). Relevant to this work are the tools proposed by researchers for custom robots that can be digitally fabricated [135, 136, 137, 155, 205]. These approaches focus particularly on origami-based robots and rely on a set of expert-designed foldable building blocks that are hierarchically chained together using a custom scripting language. Although we use a similar abstraction to enable modular composition, rather than defining new robots through a scripting language, we develop an intuitive *visual design environment* that provides guiding suggestions and automatic design completion capabilities.

Specialized design systems have also been developed at commercial scale such as Tinkerplay [20], and VEX assembler/SnapCAD [225] (See chapter 2 for more examples). We take inspiration from these applications, which are developed for specific types of physical artifacts such as articulated 3d-printable figurines or LEGO robots. The goal of our design system, however, is to be more general and provide users with the ability to create robotic devices using different libraries of electromechanical components.

Simulation-based feedback: When experts design robotic systems, simulators such as Gazebo play a critical role as tools for quick and efficient testing of new concepts, strategies, and algorithms [112]. Simulation-based feedback for design has also proven to be beneficial for other design domains such as Aerospace, and Architecture [96, 160]. We wish to bring such capability within robot design tools meant for novices. To this end, we build on the work of Megaro *et al.* [134] to provide intuitive understanding of the robot behavior in real world to the users at design time, using a physical simulation environment. While [134] focused solely on walking motions for 3D-printable robots, our goal is to enable the design of a much larger variety of devices leveraging modular, off-the-shelf components.

Automatic robot synthesis: A long-standing goal in robotics is the automatic synthesis of robots based on specifications. To this end, inspired by Sims' work on evolving virtual creatures [193], a variety of evolutionary methods that aim to design a robot's structure and even control inputs at times, have been investigated [122, 126], and this effort continues today [18]. However, evolutionary methods rely on stochastic algorithms that are slow, notorious for exploiting unwanted loopholes in the problem specification, and lead to results that are often not repeatable [122]. Rather than relying on stochastic algorithms, our design system puts *the user in the loop* and assists them with suggestions and semi-automatic design completion.

4.4 Design abstraction: A formal model

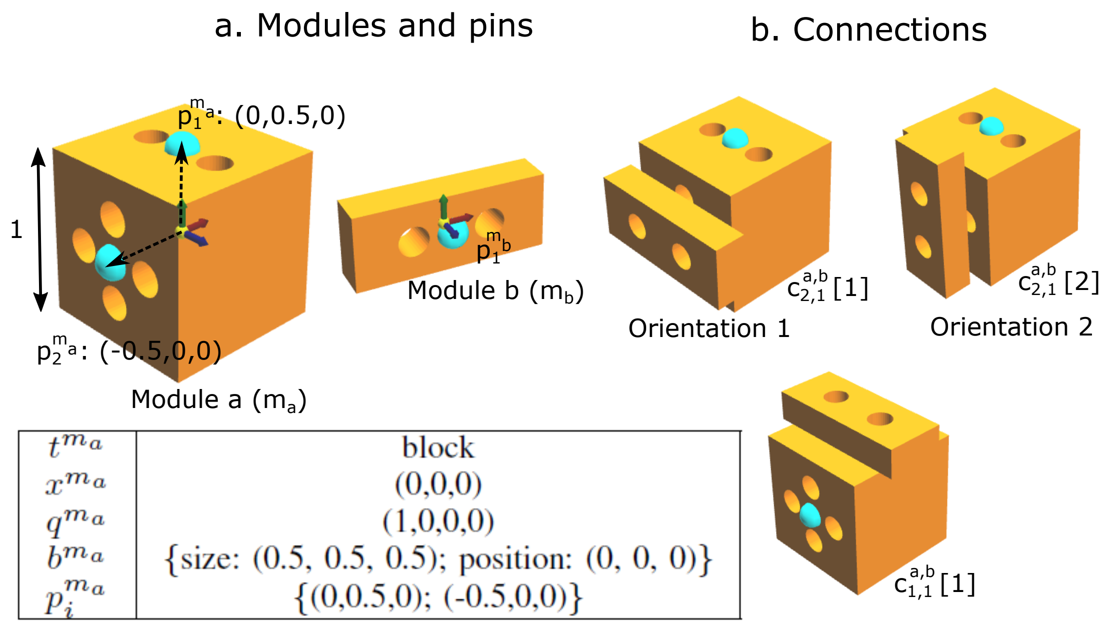
To efficiently capture the process of creating customized robots, we formalize a design abstraction. The main elements of our abstraction are the modular components or building blocks, and the way they connect to each other.

Modules and pins: Each electromechanical component (henceforth called a module) of a robot is modeled as a rigid body with 6 degrees of freedom (DOF). Each module also has an associated bounding box and a set of virtual pins (henceforth called pins). Bounding boxes are used during the design process for collision detection, while pins model physically compatible connections between modules. A module m is formally defined as:

$$m = (t^m, x^m, q^m, b^m, \{p_1^m, \dots, p_n^m\}) \; , \tag{4.1}$$

where t^m denotes the module type (e.g., motors), x^m and the quaternion q^m store the module's position and orientation in a global coordinate frame, b^m represents its axis aligned bounding box, and $\{p_i^m\}_{i=1}^n$ is a set of n pins. Each pin p_i^m represents a pre-set location on module m where another module can attach (e.g., the horn of a servomotor). We use local coordinates to represent the location of each pin. Axis aligned bounding boxes for each module are also defined in local coordinates.

Fig. 4.3(a) illustrates two hypothetical modules together with their virtual pins.



t^{m_a}	block
x^{m_a}	(0,0,0)
q^{m_a}	(1,0,0,0)
b^{m_a}	{size: (0.5, 0.5, 0.5); position: (0, 0, 0)}
$p_i^{m_a}$	{(0,0.5,0); (-0.5,0,0)}

Figure 4.3: Design abstraction example – (a) Two hypothetical modules (a and b) are shown here. Module a is a block of 1 unit, with two pins (shown in cyan) and module b is a type of connector with one pin. Their local co-ordinate frames are shown at their center with a triad. The pin positions and bounding box are defined with respect to this frame. The table enumerates the parameters that define module a. (b) All possible connections between modules a and b are shown here. Modules a and b connect with each other at $p_2^{m_a}$ and $p_1^{m_b}$ in two different orientations while they can connect in only one orientation at pins $p_1^{m_a}$ and $p_1^{m_b}$.

We associate with each module a semantic label t^m, which can take on a value in the set $\{motor, connector, body\_plate, end\_effector\}$. Modules with type $motor$ are used to represent the actuated degrees of freedom of a robot, while the $end\_effector$ type indicates that the module is a foot, wheel or manipulator. Modules labeled as $connector$ denote structural components, such as mounting brackets, and $body\_plate$ modules specify locations on the robot's main body where motors can be attached to start new kinematic chains (e.g., a new leg). The geometry of the robot's body is generated as the convex hull of all $body\_plate$ modules in a design.

Connections: Connections encapsulate the conceptual design rules that specify how individual modules can be combined to form complex robotic systems. A connection between two modules m_a and m_b is defined through a pair of compatible pins, as well as a set of physically-valid relative orientations (e.g., due to screw mounting holes):

$$c_{i,j}^{a,b} = \left(p_i^{m_a}, p_j^{m_b}, \{q_1^{m_a,m_b}, \dots, q_n^{m_a,m_b}\}\right) , \tag{4.2}$$

where $c_{i,j}^{a,b}$ denotes a connection between modules m_a and m_b using the i^{th} pin of m_a ($p_i^{m_a}$) and j^{th} pin of m_b ($p_j^{m_b}$). We use $c_{i,j}^{a,b}[k]$ to denote the k^{th} feasible relative orientation (i.e. the quaternion $q_k^{m_a,m_b}$) between the two modules. Connecting two modules through $c_{i,j}^{a,b}[k]$ amounts to positioning them relative to each other such that the world positions of pins $p_i^{m_a}$ and $p_j^{m_b}$ line up and the relative orientation between them, $q_w^{m_a-1}q_w^{m_b}$, is $q_k^{m_a,m_b}$. As shown in Fig. 4.3(b), modules can be connected to each other both using different relative orientations, and through different pairs of pins altogether. We store the library of modules, the virtual pins and all connectivity information in a configuration file, as exemplified in Fig. 4.4. We note that the configuration file does not explicitly store the position x^m and orientation q^m of the modules. These are determined automatically during the design process.

A complete testbed: To test our computational design approach, we defined a complete library of modular components consisting of Dynamixel XM430 actuators, off-the-shelf mounting brackets and 3D printable end effectors. As illustrated in Fig. 4.5(a), the library features four types of connectors: *horn bracket*, *side bracket*, *bottom bracket*, and a *custom connector* that can be attached to each type of bracket in multiple ways. The library also includes two types of end-effectors that can be used as point or area feet for legged robots, as well a wheel for mobile robotic vehicles.

We note that while we use this testbed, and its modules for all our prototypes, our abstraction is very general, and can support other modular components, or any user-designed custom parts equally well. In Chapter 5, we use this framework with completely different library of modular building blocks from industry for creating robotic arms.

4.5 Interactive and powerful visual design

Our design abstraction enables an interactive process for creating robotic devices. In addition to a manual mode that leverages a graphical user interface capable of providing meaningful suggestions, we also develop a search-based algorithm for design auto-completion. Once a design is finished, our system provides a physically-simulated environment where robots can be tested before being assembled. In particular, for legged robots, we use a previous work to efficiently generate stable full-body motions [134]. By providing feedback at interactive rates, users of our system can iteratively adjust their design until it best meets their needs.

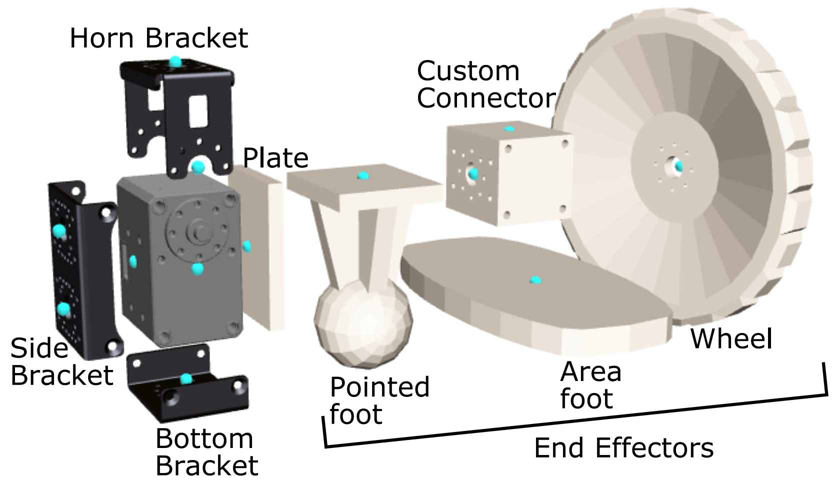
4.5.1 System-guided manual design

Using our design abstraction as the technical core, we developed a visual design system for interactive, on-demand generation of custom robotic devices. As Fig. 4.6(a) illustrates, users of

```
Module
Name a
Type Block
BoundingBox
Size 0.5 0.5 0.5
Position 0 0 0
Pin
Name a-pin1
Position 0 0.5 0
Pin
Name a-pin2
Position -0.5 0 0

Module
Name b
Type Connector
BoundingBox
Size 0.75 0.3 0.1
Position 0 0 0
Pin
Name b-pin1
Position 0 0 0.1

ConnectionMap
PinPair a-pin2 b-pin1
RelativeOrientation 0.7 0 0.7 0

PinPair a-pin2 b-pin1
RelativeOrientation 0.49 0.49 0.49 -0.49

PinPair a-pin1 b-pin1
RelativeOrientation 0.49 0.49 0.49 0.49
```

Figure 4.4: Configuration file example – Information about modules and their connection can be stored in a file for our system to use. This representation allows using any modules, as long as their geometric information and connections can be defined properly.

a.



b.

	Motor	Horn Bracket	Bottom Bracket	Side Bracket	End Effector	Plate	Custom Connector
Motor		13	9	18		40	
Horn Bracket	13	8	3	8	5		10
Bottom Bracket	9	3	3	4	5		4
Side Bracket	18	8	4	3	10		
End Effector		5	5	10			
Plate	40						
Custom Connector		10	4				

Figure 4.5: A design testbed with Dynamixel modules – (a) We use custom designed modules such as end-effectors, plates and a custom connector (shown in white) as well as commercially available Dynamixel modules such as motors. Their pins are shown in cyan. (b) Table describes connection compatibility between these modules. Each shaded entry indicates the total number of connections between two modules. Wheel can only connect to the motor in one way and hence end effectors entries refer only to the feet.

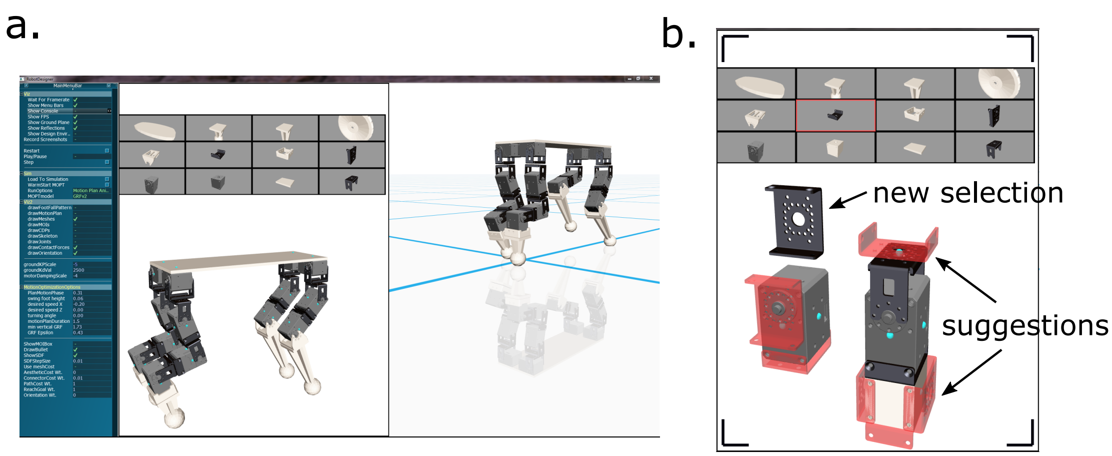


Figure 4.6: Visual design system – (a) The design interface consists of two workspaces – the left workspace allows for designing the robot while the right workspace runs a physics simulation of the robot designed by the user for its feasibility. The left workspace displays a list of various modules at the top. The leftmost menu provides various functions that allow users to define preferences for the search process, visualization as well as for physical simulation. (b) When the user selects a new module from the list, our system makes visual suggestions (shown in red) about possible connections for this module, based on the current design.

our system are presented with a design workspace (left) and a simulation workspace (right). The design workspace allows them to browse through the list of available modules, which can be dragged and dropped into the scene at any time. Once a specific module is selected, our system visualizes the ways in which it can be connected to the design the user is currently working on (see Fig. 4.6(b)). This is achieved by iterating through all unused pins in the current design and checking if they are compatible with the pins of the new module. As the user positions the module in the scene sufficiently close to any of the suggested placements, it is automatically snapped into place. The user can then cycle through the set of possible relative orientations (i.e., $c_{i,j}^{a,b}[k]$) associated with the connection that was selected. Our design system also supports other editing operations that increase productivity. For example, different parts of a design can be marked as symmetric (e.g., left and right limbs), with edits made to one part being automatically propagated to the other. Different parts of an existing design can also be copied, pasted and mixed with other designs. To further guide the design process, users can load a 3D mesh and overlay it into the workspace, as seen in Fig. 4.7. We illustrate a representative design session in this video – `https://www.youtube.com/watch?v=PGpTsQtznw4`

4.5.2 Design auto-completion

The manual mode described in the previous section affords users with full control over the design. Nevertheless, this exploratory design process can become somewhat tedious if a large number of modules is required to create a robotic device. We therefore developed a novel, semi-automatic mode that allows users to focus primarily on functional characteristics of their design. For example, if the goal is to create a robot arm with 3 actuated joints, shoulder, elbow and wrist, the

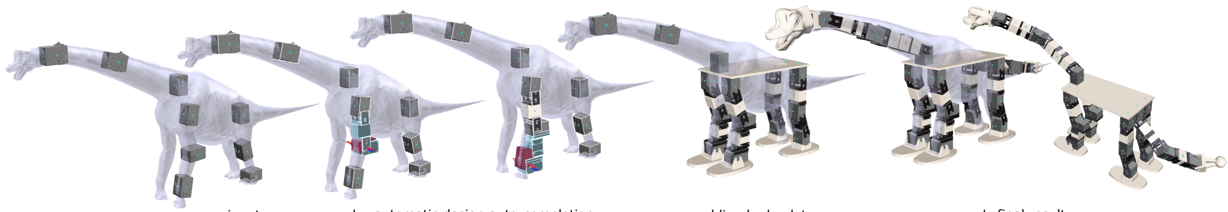


Figure 4.7: Automatic design with search – (a) Users can start with a guiding mesh for the robot they want to make. Then, they specify the positions and orientations of motors for this robot, using the drag and drop interface. (b) Our search process searches for possible designs that connect a given pair of motors in user-defined locations, according to user-defined preferences. The user can reject the solution and re-do the search with different preferences any-time. A proposed search solution connecting the root motor to the target motor (highlighted in dark red) is shown in light blue. The design is only created if the user accepts the proposed solution. The process is repeated by the user for each pair of motors. (c) Since the legs are symmetric, the users only need to use search process for two legs. Our interface allows them to create the other pair of legs by simple editing operations. Finally, the users can attach end-effectors of their choice and create a body plate to complete the robot design. (d) shows the final design (with and without the guiding mesh). The dinosaur head mesh was manually added after the design, for aesthetic appeal. To do so, a dinosaur head module with appropriate geometry and pins was added to the library. This further shows how our tool can work with any modular building blocks.

user can simply specify how the joint motors should be positioned relative to each other. Our system then employs a computational algorithm to auto-complete the design. This is achieved by searching for a sequence of modules that result in appropriate mechanical structures connecting the shoulder elbow, and wrist motors.

We use a heuristic-guided tree search algorithm to auto-complete designs. As illustrated in Fig. 4.8, starting from a *root motor*, our computational system creates a tree of possible designs in a recursive manner. Briefly, nodes correspond to modules, edges describe how compatible modules connect to each other, and the path from each node to the root corresponds to an intermediate robot design. Leaf nodes of the search tree correspond to designs that end with a *target motor*. The goal of our design auto-completion algorithm is to find the leaf node corresponding to a relative placement between the root and target motors that is as close as possible to what the user specified (e.g., placement of elbow motor relative to shoulder motor, or wrist motor relative to elbow motor). As described in the remainder of this section, we explore several heuristic functions that guide the search process in order to make it computationally efficient, and therefore suitable for interactive design.

Search tree: The search tree T is a collection of nodes and edges, $T = (N, E)$. Each node N_i represents a module m_i and is defined as:

$$N_i = (m_i, N_i^p, \{{}_1N_i^c, \ldots, {}_nN_i^c\}) \ , \tag{4.3}$$

where m_i is the module corresponding to node N_i (defined by eq. 4.1), N_i^p represents its parent node – the node it originated from, and $\{_j N_i^c\}_{j=1}^n$ represents a set of n successive nodes called children nodes. Node N_i has a child node for every way in which module m_i can connect to any other module in the library. The child nodes themselves represent the modules that are to be connected to m_i, while the way in which the two modules are attached to each other is specified by the edges of the tree. In particular, an edge connecting nodes N_a and N_b is defined as:

$$E_{i,j,k}^{a,b} = \left(N_a, N_b, c_{i,j}^{a,b}, k\right) , \tag{4.4}$$

where a is the index of the parent node, b is the index of the child node, $c_{i,j}^{a,b}$ references the connection between their modules m_a and m_b at virtual pins $p_i^{m_a}$ and $p_j^{m_b}$ respectively (defined by eq. 4.2), and k indicates the index of a relative orientation from the set of feasible configurations for $c_{i,j}^{a,b}$. With the parent node N_a's position and orientation kept fixed, the child node N_b's relative placement is automatically determined such that the locations of the connection's virtual pins (i.e. $p_i^{m_a}$ and $p_j^{m_b}$) coincide in world coordinates, and the relative orientation between the two is set according to $c_{i,j}^{a,b}[k]$. We note that nodes of the search tree are expanded as needed during the search process.

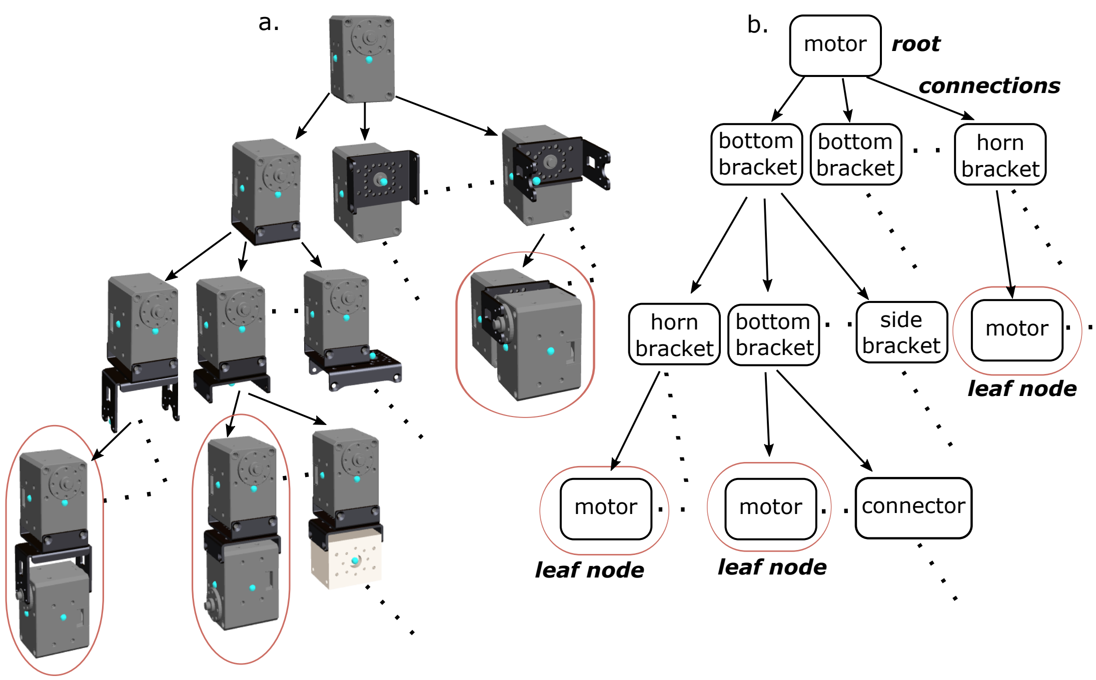


Figure 4.8: Design space tree – The search tree originates at the root motor and enumerates all possible structural designs between the root and the target motor. The visualization of physical designs represented by the search tree and its corresponding schematic representation are shown in (a) and (b) respectively. Each branch represents a different design that results in a particular target motor configuration (encoded by its leaf node, highlighted with red oval).

Informed tree search: Since the depth of the search tree is potentially unbounded, and each node has a large branching factor (see Fig. 4.3(b) and Fig. 4.5(b)), the design space is vast. As a result, a brute force traversal of the search tree would be prohibitively expensive. We therefore propose an informed search process that accounts for the desirability and feasibility of a design. We leverage existing informed search methods in graph theory, namely the A* algorithm, and adapt it for interactive, user-driven design [182].

Informed search methods use problem specific knowledge to decide which nodes to expand while traversing the search tree. Heuristic functions that embed the target information are the most common form of domain knowledge used during the search. We define a heuristic function h that measures the promise of each branch in reaching the target motor:

$$h(N_i) = |x^{m_i} - x^{m_t}| \ , \tag{4.5}$$

where N_i represents current node of a branch, module m_i represents N_i's module, m_i's position $x_w^{m_i}$ is defined in eq. 4.1, and m_t is the target motor. h refers to the euclidean or straight-line distance between m_i and m_t. Note that h is an admissible heuristic because the shortest path between any two points is a straight line, so it cannot be an overestimate [182]. Apart from heuristics, other cost functions can be used to bias the search in finding designs with desired properties.

Desirability cost: We define a desirability cost that encodes user preferences regarding aesthetics and resource economy. The desirability of a design is measured in terms of the total number of modules used and the compactness of the resulting structure. A design that uses a lower number of modules may be more economical (and it is trivial to associate different costs to different types of modules). Similarly, a design that connects the root motor and target motor using an approximately straight structure might be more compact, and therefore more aesthetically pleasing. The desirability cost D of a branch in the search tree measures these two characteristics of its design, at its current node N_i.

$$D(N_i) = w_c * \delta(N_i) + \frac{\sum_{j=1}^{\delta(N_i)} dist(N_j, \overline{RT})}{\delta(N_i)} \ , \tag{4.6}$$

where w_c is a scaling weight and $\delta(N_i)$ is the depth of node N_i in the branch (with root node at depth 0). The first term therefore represents the total number of nodes (modules) between the root and the node. $\overline{RT}$ represents a line segment in 3D joining the root and the target nodes, and $dist(N_j, \overline{RT})$ measures the distance between the node N_j and $\overline{RT}$. This distance is zero if the position of N_j's module ($x_w^{m_i}$) lies on $\overline{RT}$, thereby implying a straight structure. The second term thus measures the average deviation of branch's module positions from $\overline{RT}$. The nodes with lower D are more desirable. D's role in the search is comparable to that of path cost g in A* search [182]. However, defining D allows us to present the users with intuitive handles to control the search output.

The total cost f of a node N_i is then determined as:

$$f(N_i) = h(N_i) + w_d D(N_i) \, , \tag{4.7}$$

where w_d is a user-defined weight, $h(N_i)$ is defined by eq. 4.5, and $D(N_i)$ is defined by eq. 4.6. h, D and f are calculated and stored for each node during the expansion process. The definition of node N_i is extended to store them.

$$N_i = (m_i, N_i^p, \{{}_1N_i^c, \ldots, {}_nN_i^c\}, h, D, f)\ , \tag{4.8}$$

where m_i, N_i^p, $\{{}_jN_i^c\}$ are defined as in eq. 4.3. h, D and f are defined as in eq. 4.5, 4.6, 4.7 respectively. Algorithm 24 describes our search process that uses f to determine which node to expand. The node with lowest f is expanded first. The $openset$ keeps track of all the nodes yet to be expanded. Upon expansion, only the nodes free of collision are added to the $openset$, to ensure physical validity of the design. We use bullet collision engine [1] to compute collisions between modules. Expanding nodes with lowest f ensures that the branches whose structures extend towards the target motor position are expanded and traversed first in the tree. However, the function of an articulated link with multiple motors depends not only on the position of the motors, but also on their rotation axis. Hence, not all the designs that extend to the target motor position might function as desired. We therefore define an additional functionality cost.

Functionality cost: The functionality of each branch in the search tree is measured using the error between the position and orientation of the motor's axis at its leaf node and the user-specified target values. This error term determines whether the proposed design is acceptable. F is comparable to a termination criteria of conventional search methods.

$$F(N_i) = |x^{m_i} - x^{m_t}| + (1 - |\overrightarrow{a}^{m_i} \cdot \overrightarrow{a}^{m_t}|)\ , \tag{4.9}$$

where N_i is a leaf node, m_i represents the motor module associated with N_i, and m_t is the target motor which is placed in the design by the user. x^{m_i} and x^{m_t} denote positions of m_i and m_t respectively (eq. 4.1). $\overrightarrow{a}$ is a vector denoting the motor's axis of rotation. For certain applications, users might also care about the rotation of the target motor about it's rotation axis. This degree of freedom does not interfere with the functionality of a design, but it may affect its form factor. In such cases, F can be extended to account for this error in the orientation of the target motor:

$$F(N_i) = |x^{m_i} - x^{m_t}| + (1 - |\overrightarrow{a}^{m_i} \cdot \overrightarrow{a}^{m_t}|) + \Delta(q^{m_i}, q^{m_t})\,, \tag{4.10}$$

where q^{m_i} and q^{m_t} are quaternions representing orientations of modules m_i and target motor m_t respectively (eq. 4.1). Their orientation difference $\Delta(q^{m_i}, q^{m_t}) = \cos^{-1}(scalar(q_d))$, where $q_d = (q^{m_i})^* q^{m_t}$ and q^* represents quaternion conjugate. F is defined by eq. 4.9 by default. In algorithm 1, as soon as a motor module node is encountered in the expansion process, instead of adding it to the $openset$, it is compared to the target motor using F to determine whether its branch represents a valid functional design. If the design is valid, the search displays it to the user and discards it otherwise.

Comparison to conventional search: When $w_d > 0$ in eq. 4.7, our search process (algorithm 1) becomes an A* algorithm that determines which nodes should be expanded by summing up the heuristic cost-to-goal estimate and the cost of intermediate designs [182]. Setting $w_d = 0$ converts the search into a greedy best-first search that only uses the heuristics for node expansion. Throughout the search process, we use our functionality metric F to keep track of the current best node. Instead of waiting for the search to find an optimal mechanical assembly, the design corresponding to the current best node is displayed and updated as the search progresses. This strategy produces various alternative designs for users to choose from as they are found, and it leads to lowers wait times that promote interactive design. Users can accept any of the designs produced by the system at any point in the search process. Keeping users in the loop in this manner allows us to also account for aesthetic preferences that may not be captured by the desirability cost. If users do not choose any design, we exit the search when the number nodes in $openset$ exceeds a threshold, returning the design with minimum F. Inspired by the various alternative designs proposed during the search, users are free to change the placement of the root or target motors at anytime. The search process will account for this change and will be updated immediately. Users can further influence the search process by specifying different weights for the orientation or desirability objectives (eq. 4.7,4.10).

4.6 Evaluation

Our design system allows casual and expert users alike to easily design customized robotic devices that range from wheeled vehicles to legged robots (Figures 4.9, 4.10 and 4.12). In this section, we evaluate our system by analyzing the abilities of the manual and automatic design modes. Further, we verify the feasibility of the designs made with our system by fabricating two very different robots – a wheeled robot with a manipulator arm that draws called "robo-calligrapher"', and a quadruped robot which we call "puppy".

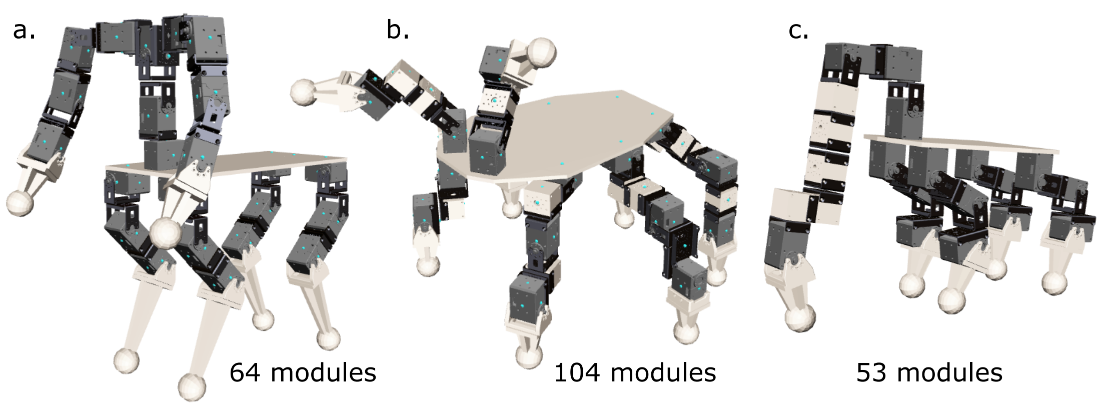


Figure 4.9: Various robots designed with our system – (a) a centaur, (b) a hexapod with arms, and (c) a pentapod. (a) was designed in the manual mode, (b) was designed using the search, and (c) was designed using both modes. Its short legs were designed in manual mode while the longer limb was designed with the search process. Each design is composed of off-the-shelf modules and custom 3D printed modules (shown in white). The total number of modules used in each design (mentioned below) indicate their complexities.

Guided by visual suggestions provided by our system, the manual mode we developed is a powerful tool that allows users to explore the design space. Fig. 4.10 shows several different designs for two types of quadrupeds. Each of these designs took a matter of minutes to create. When considering the number of motors that can be used for each limb, their placement and

Algorithm 2: Interactive search for modular robot structure

```
input  : Root motor ($r$), target motor ($t$), $max_N$, $w_d$
output: Designs connecting $r$ and $t$.

// start with root node
calculateCost ($N_r$); $openset \leftarrow N_r$
// keep track of the best design
$N_{best} = N_r$
while $0 < size(openset) < max_N$ do
    // select node for expansion
    $N_{current} = N$ in $openset$ with lowest $f$
    // node expansion
    $N_{new} \leftarrow \{{}_j N^c_{current}\}^z_{j=1}$
    for $i \leftarrow 1$ to $z$ do
        // check feasibility
        if $N_{new}(i)$ is free of collision then
            calculateCost ($N_{new}(i)$)
            // compare leaf node to target
            if $N_{new}(i) \rightarrow m = motor$ then
                if $F(N_{new}(i)) <= F(N_{best})$ then
                    $N_{best} = N_{new}(i)$
                end
            end
            // add to openset
            else
                $openset \leftarrow N_{new}(i)$
            end
        end
    end
end
// output best node
return $N_{best}$
Function calculateCost($N$)
    calculate $N \rightarrow h$, $N \rightarrow D$, $N \rightarrow f$
end
```

direction of rotation axes, configurations of the legs and how they attach to the body, the design space is very rich. The ability to quickly create or alter designs is therefore very important in exploring the relationship between the form and function of the robot. To further speed up design processes, our computational system allows users to copy-and-paste different parts of a design in order to mix and match features from different robots. This functionality is demonstrated with the top-left design in Fig. 4.10, where the robot employs two legs from the spider design, and two from the mammal design. Upper bodies with an arbitrary arrangement of manipulators can also be easily designed, as demonstrated by the robots presented in Fig. 4.9.

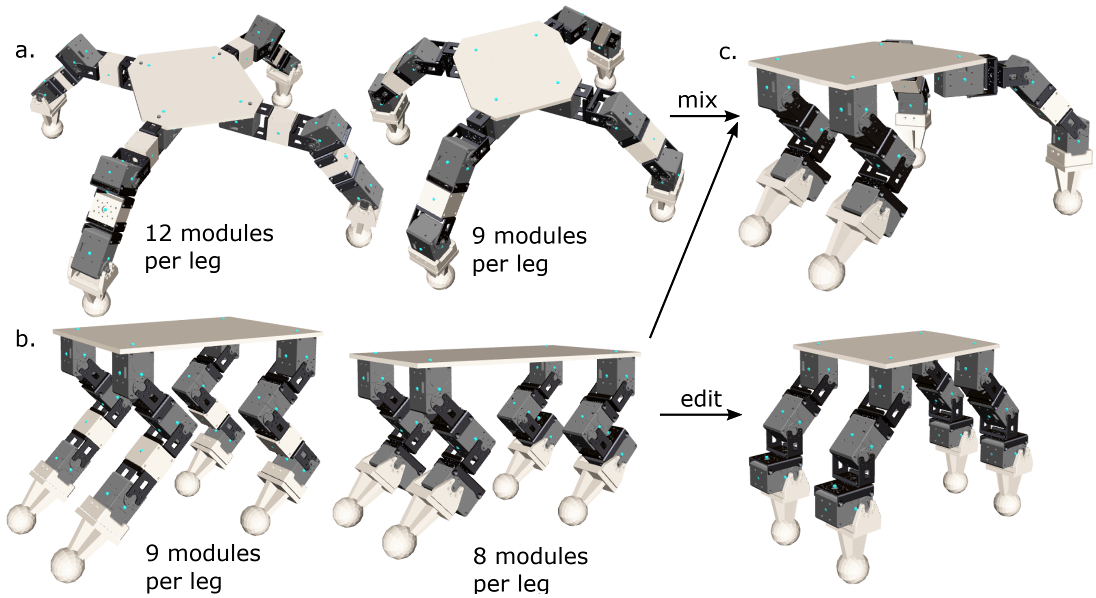


Figure 4.10: Design editing and mixing – Two types of quadrupeds inspired by spider-like legs and mammal-like legs are shown in (a) and (b) respectively. Each design uses 3 motors per leg, but uses different number of modules per leg (mentioned below each design). These modules are also used in different configurations resulting in diverse looks and motion behaviors. Designs in (c) were made by editing and mixing designs from (a) and (b).

Fig. 4.7 illustrates the process of creating a dinosaur robot using our design auto-completion mode. For designs such as this, the large number of required components can make the manual mode too tedious and time consuming. Our semi-automatic mode proposes designs that attach motors to each other through a series of modular connectors. Each design is generated to ensure that the final placement and orientation of the motors is as close as possible to user-specified configurations. Fig. 4.11(a), for example, shows various design alternatives suggested by our search algorithm. By specifying preferences regarding the number of modules, aesthetics (eq. 4.7), and motor orientations about their rotation axes (eq. 4.10), users can intuitively influence the result of the search process.

Fig. 4.12 shows two fabricated prototypes of robots designed with our system. The body plates, wheels and feet were 3D-printed, while all connecting brackets are off-the-shelf, aluminum parts. The robo-calligrapher robot was designed to take high-level commands from a blue-tooth device. By controlling the velocity of the motors that are attached to the wheels, it can

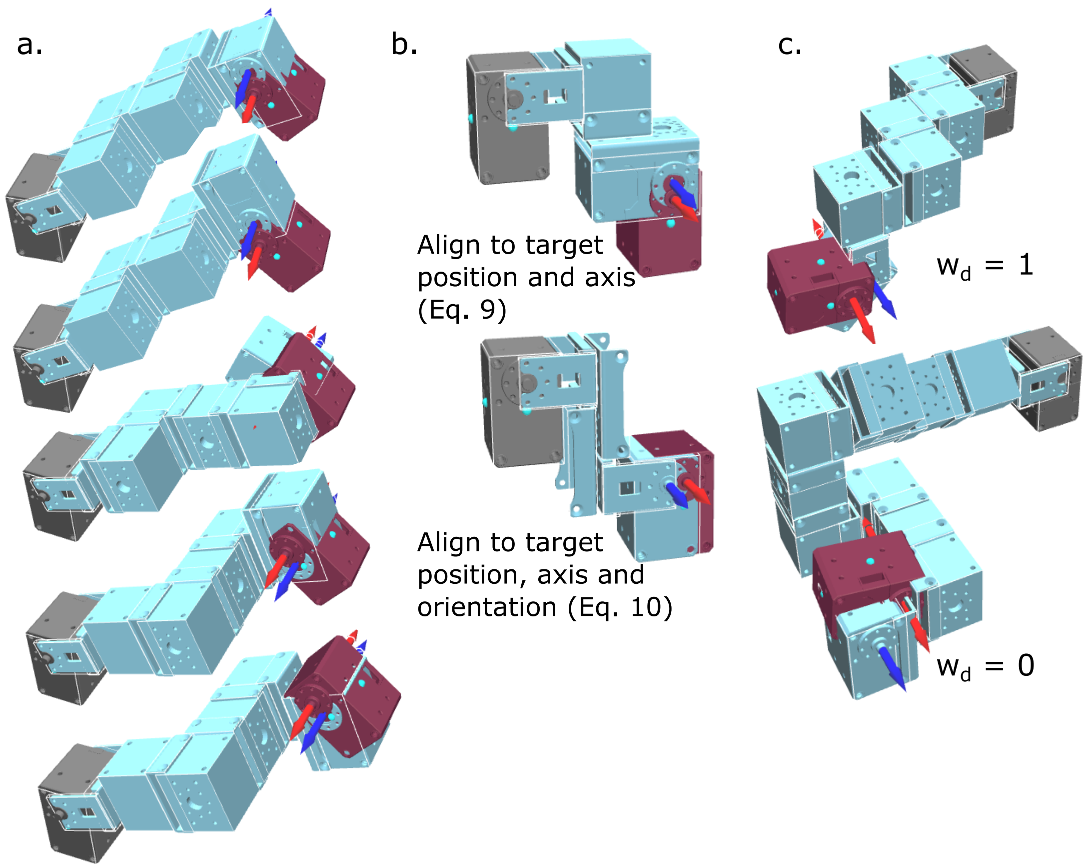


Figure 4.11: Tree-search analysis – (a) The search proposes various design alternatives (shown in light blue) that connect the root motor (in gray) to the target motor (in dark red) in desired configuration as closely as possible. The motor axis alignment for the target motor (red axis) and the proposed structure (blue axis) is also illustrated. (b) The resultant designs vary based on F cost used for comparing the resultant motor configurations of the proposed designs to the target motor. (c) The importance of desirability cost is shown here. When w_d in eq. 4.7 is set to zero, the resultant designs end up using more number of modules and are aesthetically less appealing.

easily be commanded to move forward, backward and to turn. We further developed an application that translates a user sketch to a sequence of motor commands for its arm. We used inverse kinematics for this purpose. Our design system allowed us to experiment with the number of actuators used in the arm. We found that with 3 motors and off-the-shelf mounting brackets, its range of motion was too limited, and therefore it was not able to reproduce a significant number of sketches we provided to it. Based on this diagnostic, we decided to add two additional motors to the design.

The puppy robot was designed to walk forward and sideways. For this robot, the feedback obtained from the simulation environment and the efficient iterative design process enabled by our system were particularly useful. As shown in the video[1], our system makes it easy to experiment with different body proportions, types of end effectors and motor configurations. In addition to affecting the perceived motion style, these choices also affect the robot's ability to perform different motor tasks. For example, before converging to our final design, we created a few robots that were only able to walk forward, and not sideways (e.g., bottom left design in Fig. 4.10). As our video shows, the motions of the physical prototype match well the simulated results. We therefore find the outcome of this experiment very encouraging, since performing an equivalent exploration of the design space directly in hardware is tedious and much more time consuming.

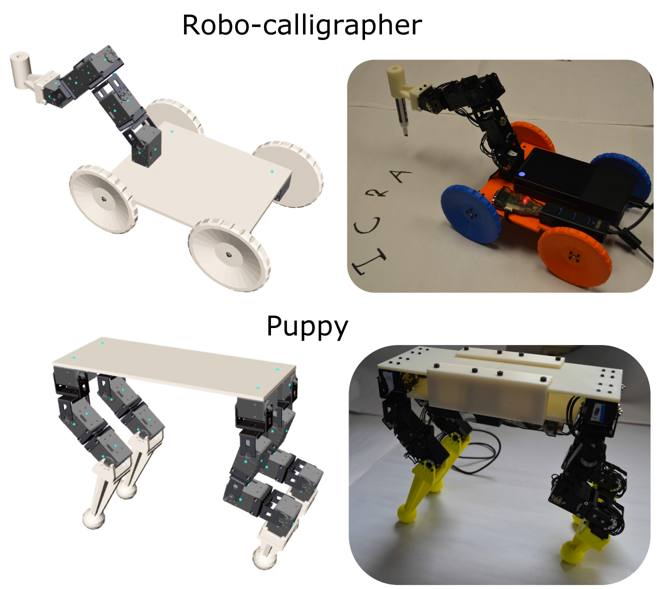


Figure 4.12: Fabricated prototypes – The "robo-calligrapher" and "puppy" robot are shown here with their fabricated counterparts. We designed a special purpose end effector that served as a pen-holder for the robo-calligrapher. Our video shows these robots in action.

[1]Available at – `https://www.youtube.com/watch?v=PGpTsQtznw4`

4.7 Frequently asked questions (FAQ)

1. **How does the search ensure that the resultant structure does not interfere with the movement of motors?**
   The required motion of the motor only depends on motor axis orientation and its position. Since the search uses this as target in the heuristics (eq. 4.9), the resultant designs will have the user-specified motor axis position and orientation. This will in turn ensure that the motor movements are as desired. We also ensure that the resultant structures are not self-colliding with the help of a Bullet physics-powered collision check during the search (line 7 in Algorithm 24).
2. **Isn't it difficult for the users to specify number of motors and their configurations for achieving a desired robot function?**
   It can indeed be hard for novice users to specify the right number of motors and their configurations to achieve desired robot functionality. We currently allow dealing with this in an iterative manner through feedback provided from the simulation. In Chapter 6, we will present an automatic design improvement approach that corrects user provided motor configurations to achieve a desired task.
3. **Aren't the alternatives provided during the search sub-optimal, since the search hasn't converged yet? Why should the users be allowed to select them then?**
   The design alternatives provided to the users as the search progresses are indeed not optimal with respect to achieving the desired motor configurations and thereby motions. However, most times user-specified configurations are approximate and users typically iterate over their designs. Further, we have found that many times the users perform a trade-off between their design performance and design aesthetics. Since we do not know whether the users prefer a straight line, curved, or zig-zag structure for their designs, the provided alternatives allow them to perform this trade off implicitly. This eliminates the challenging task of explicitly modeling such subjective user preferences about aesthetics.

4.8 Limitations

Limited automatic design support: Our system allows users to efficiently create customized robots through design space exploration and simulation-based feedback, both of which result in faster design iterations. User design is supported through a manual mode that allows forward design, and an auto-completion mode that further speeds up the design process. However, our auto-completion approach does not currently account for dynamics, external loads, or desired motion profiles that may be important for a broader class of articulated robots.

Open loop forward design: Our tool relies completely on the users to modify their designs in case of failures or undesirable design outcomes, highlighted in simulation. While such simulation-based feedback immensely helps in iteratively improving the design, translating a negative outcome observed in simulation to an appropriate change in the design may be difficult for novices. To aid casual users in improving their inadequate designs automatically, we explore inverse design approaches in our co-design tools (Chapter 6).

4.9 Publication and dissemination

This work was undertaken along with a fellow masters student Ye Yuan, and was published in International Conference on Robotics and Automation (ICRA) 2017 [60]. The tool and corresponding code, which is deployable on Windows, can be downloaded from here – `www.cs.cmu.edu/~rutad/files/articulated_robot_design.zip`. Fabrication information and relevant part files for Robo-calligrapher and Puppy robots are hosted publicly on MyMiniFactory [4]. Accessible at:

- Robo-calligrapher –

  `https://www.myminifactory.com/object/robo-calligrapher-45664`

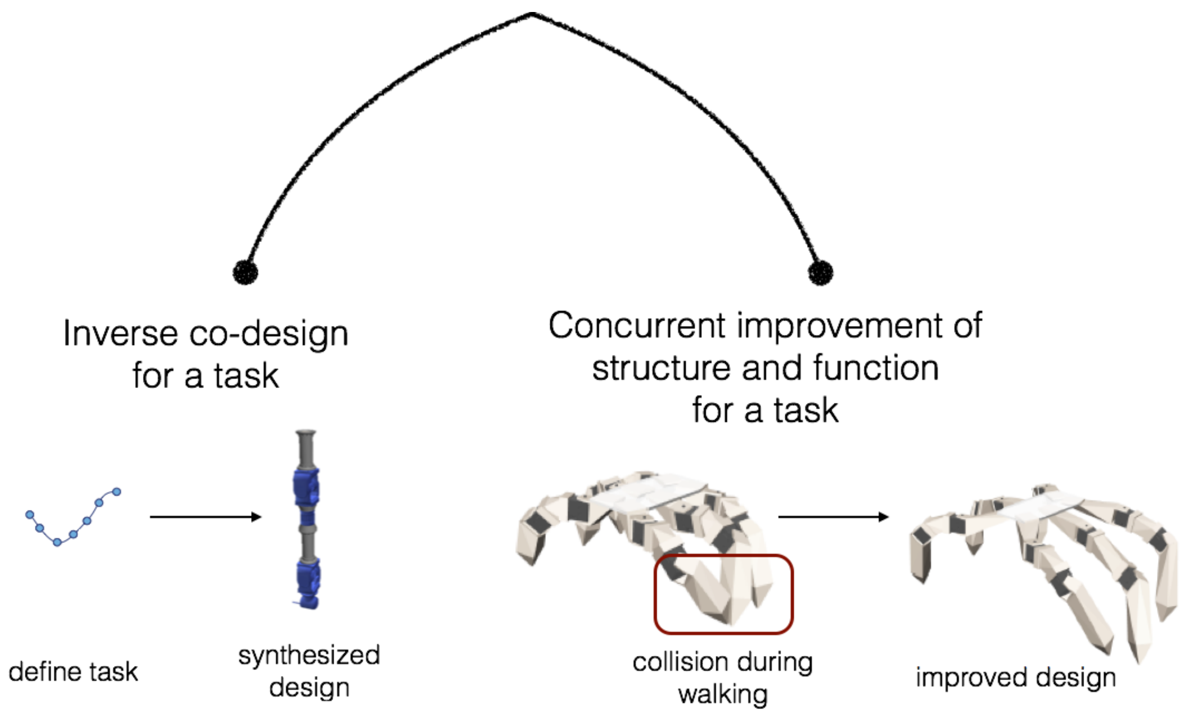
- Puppy –

  `https://www.myminifactory.com/object/puppy-45663`

An overview video about the tool is also available – `https://www.youtube.com/watch?v=PGpTsQtznw4`

Part II

Tools for structure and function co-design



Chapter 5

Automatic design of task-specific robotic arms

5.1 Preamble

With our modular structure design tool (Chapter 4) users design a custom robot, test it within a physical simulation, and iterate between design and testing till a desired robot that functions as needed is created. Such a paradigm is called *forward design*. While forward design helps when users are exploring the design space and are unclear about what they exactly want, a complementary approach of starting with the task in mind may be more efficient in other scenarios (Figure 5.1). We call this an *inverse design* approach. Such an approach is also essential for applications wherein target casual users may not be able to provide the system with low-level inputs necessary for the design such as degrees of freedom, thereby leading to inaccessibility.

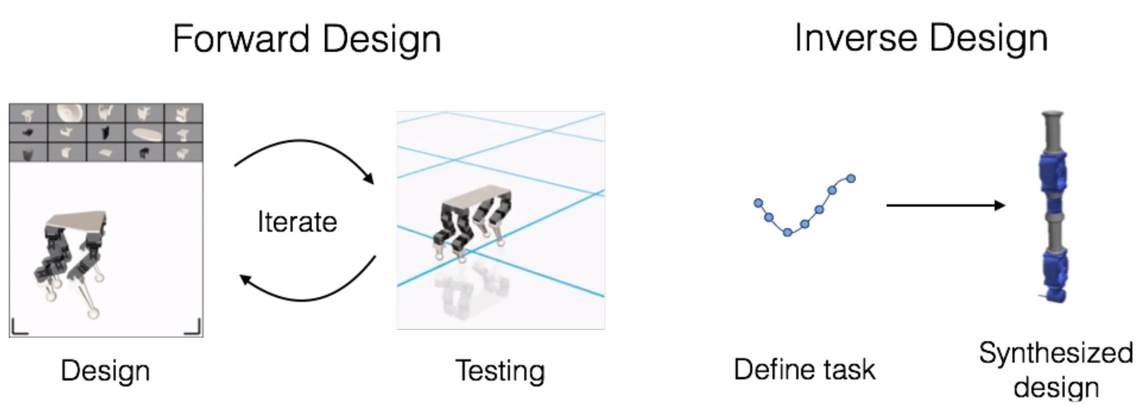


Figure 5.1: Forward vs. inverse design – Inverse design enables users to start with the task requirements, instead of iteratively improving their design till requirements are met. We propose an inverse design system that allows users to obtain robotic arm designs that will execute the user-specified motions.

In this chapter, we extend our modular design abstraction and framework for inverse design, and demonstrate it for the case of robotic arm design. We enable users to define the task requirements of a robotic arm in form of a motion trajectory. Given this task requirements, our system automatically synthesizes an appropriate arm as well as its motion to achieve the task. Since we concurrently design robotic arm's structure and motion, this work falls under the category of co-design tools.

5.2 Introduction

Many complex manufacturing scenarios require manipulation in confined spaces, while avoiding obstacles. In particular, specialized domains such as airplane manufacturing involve many such intricate motions, and therefore demand substantial manual labor. Commercially available 6 degrees of freedom (DOF) robotic arm systems such as KUKA [117], Universal robots [219] etc. cater to a large variety of manipulation tasks. However, these robotic arms cannot be modified for specialized tasks, nor can they be reconfigured when the task requirements change. Instead, robotic arms designed using reconfigurable, modular parts can be adapted as per one's need. Therefore, recent commercial robotic systems such as HEBI robotics [94], Modbot [145] have started promoting the use of customizable designs by providing modular building parts. Given such a modular part library, our goal is to enable designers and engineers to quickly create valid robotic arm designs for the task at hand. Towards this goal, we present a computational approach that automatically generates valid designs given user-specified task requirements.

We formulate the automatic robot design as a search problem through the space of all possible arrangements of the modular parts, similar to our design auto-completion method in modular structure design tool (Sec. 4.5 in Chapter 4). To find valid design solutions that achieve a desired task efficiently, we encode the task requirements with a heuristics so as to guide the search towards appropriate designs. An integrated physics simulation environment is also available within the system, which can be used to test the final auto-generated designs. Although our current focus is on synthesizing robotic arms, our automatic design approach is generic, and can be applied to a variety of robotic systems.

5.3 Related work

Task-specific manipulator design based on high-level descriptions has received considerable attention in robotics community. For instance, methods to design manipulators that reach desired configurations in a specified workspace while avoiding joint singularities have been proposed [38, 107, 162]. However, most prior methods focus on optimizing continuous parameters, such as limb lengths, for a given robot arm with fixed number of joints. In contrast, our method creates robotic arms entirely from scratch using discrete modular components. To create custom designs with such discrete parts, we build upon our previous work that defines an abstraction for modeling modular components and their connections in a robotic device [60] (Chapter 4). Using this abstraction, we also previously developed a search-based design approach that auto-completed structures of user created partial robot designs [60]. In this work, we extend this framework to automatically create complete designs while accounting for high-level task requirements such as a desired motion for the robot arm's end-effector.

5.4 Graphical design system

This work was done in collaboration with National Robotics and Engineering Center (NREC) and was motivated by the needs defined by an airplane manufacturing industry partner based on their design and manufacturing challenges. The industry partner therefore provided with a library of modular parts for building robotic arms, which can be used in manufacturing setting. These library parts include a robot base, actuators, connecting links, and manufacturing specific end-effectors such as sealing gun, welding machine etc. These end-effectors allow further customization of robotic arms based on the task. The base part denotes the robot's supporting base and can be used to define the robot arm's position in the world with respect to the environment.

To enable visual design, we extend our previously developed interactive modular structure design system in two ways (Sec. 4.5). First, we enable users to define their task requirements in terms of a desired motion for the robotic arm easily – either directly within the interface through mouse, or with the help of a text file that can be dragged and dropped in the interface. Secondly, we extend our physical simulation environment so that users can define virtual obstacles within the environment to depict real-world scenarios corresponding to the task (see fig. 5.2). It is therefore referred as task designer, henceforth.

Manual editing is available if needed, and is enabled using intuitive drag-and-drop of parts from the library menu on the top. This is adapted from our previous work [60]. For automatic design, users provide a robot base position in the real world, and a desired trajectory for the arm to follow. Users can also specify a desired end-effector such as sealing gun or a welding machine, according to their task requirements. Based on these specifications, our algorithm automatically generates a valid arm design.

5.5 Automatic design using informed tree-search

Given a library of modular building parts and a user-specified robot arm end-effector motion corresponding to a task, our design synthesis method aims to generate the simplest robot that can execute this motion (see fig. 5.3(a)). Towards this, we leverage our design abstraction and tree-based search in a manner similar to our design auto-completion approach for structure design (Sec. 4.5). However, unlike before, we define a new heuristics to encode the task requirements.

Our design abstraction (Sec. 4.4), which models modular parts and their connections, allows us to map the input part library to a space of possible robot designs. Briefly, the library parts p are modeled as rigid bodies, and their compatibilities are defined using connection rules c[1]. A robot design $\mathcal{D} = \{\mathcal{P}, \mathcal{C}\}$ can now be represented by a collection of interconnected parts $\mathcal{P}$ and their connections $\mathcal{C}$. Combinatorially many such robot designs consisting of different collections of parts $\mathcal{P}$ can be constructed even with a small-sized part library. To find the simplest valid

[1]A connection rule c for two compatible parts p_1 and p_2 is defined as $c = \{p_1, p_2, {}_1\mathbf{T}_2\}$, where ${}_1\mathbf{T}_2$ represents a rigid transformation of p_2 relative to p_1 at the time of connection (see eq. 4.2)

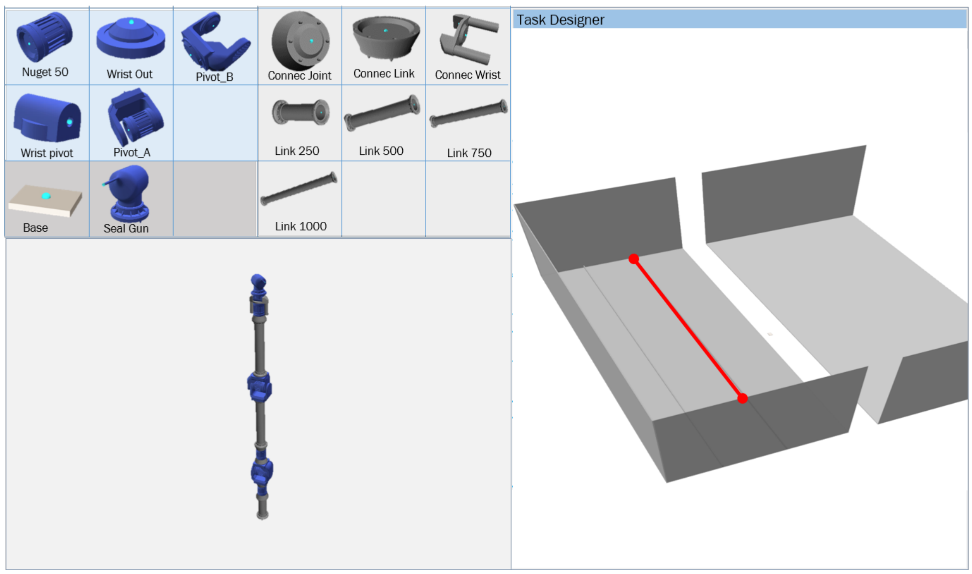


Figure 5.2: Our graphical interface shown here is based on our modular structure design system. Users can manually or automatically design robots in the design window on the left, using the modular parts displayed in a menu on the top. Actuators are shown in blue, while the connecting parts are highlighted in gray. We used the NREC Modular Actuator (Nugget) for our designs. The task designer on the right is powered by a physics simulation, and allows users to define real-world task scenarios using obstacles (gray) and target trajectories (red). Apart from enabling task-specific user inputs, the simulation also allows users to test the auto-generated designs before assembling the robotic arm hardware.

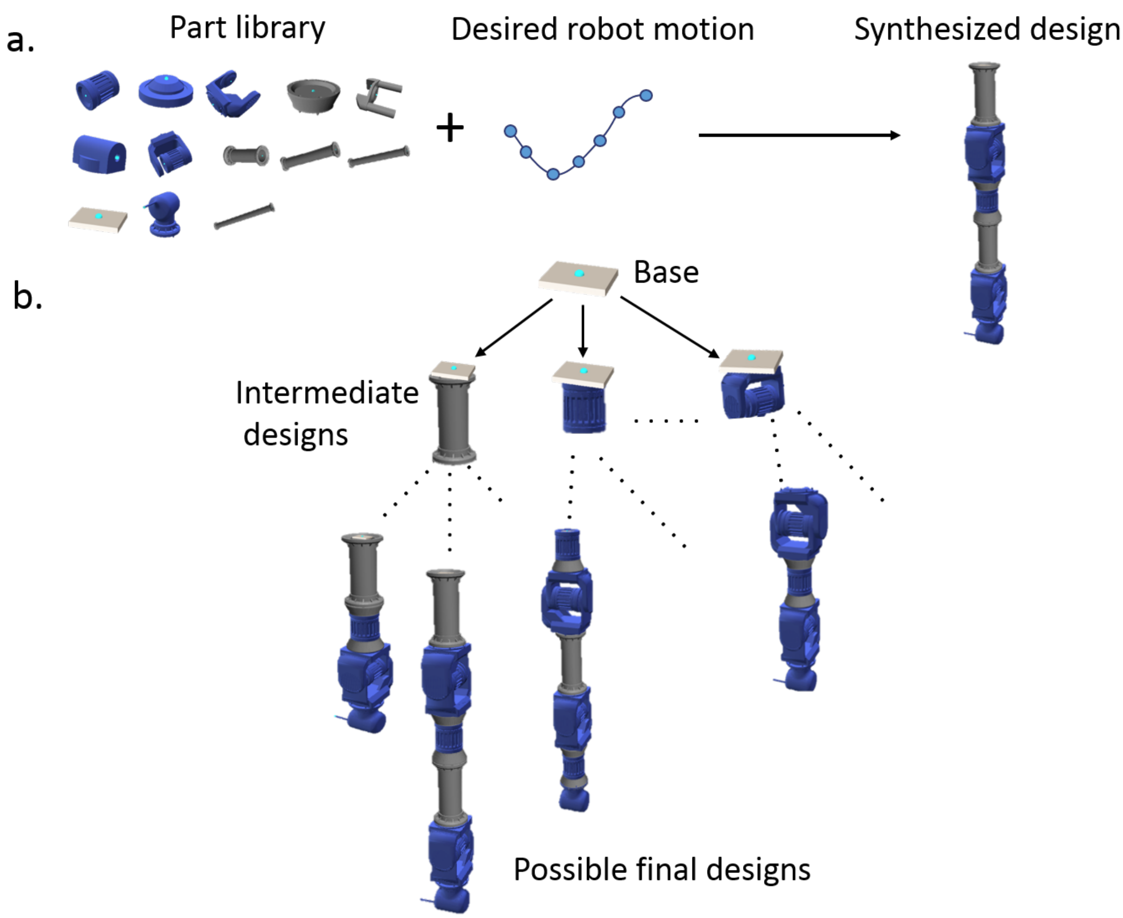


Figure 5.3: (a) Given a library of modular parts and a desired robot motion, our system automatically synthesizes valid robotic arm designs. (b) Automatic design is formulated as a search problem over recursively created tree of all possible designs.

robot design out of all such possible robot designs, we start with a robot base, and recursively construct new designs with increasing number of components till a valid design is found (see fig. 5.3(b)).

5.5.1 Tree of designs

The recursive approach for synthesizing new designs is motivated by the following observation. Consider a robot design $\mathcal{D}$ composed of part collections $\mathcal{P}$. One can create new *children* designs of $\mathcal{D}$, each with potentially different motor capabilities, by appending compatible parts from the library to $\mathcal{D}$, as defined by the connection rules. As a result, even if a design $\mathcal{D}$ is not well-suited for a user-specified motion, one of its children designs might be. This synthesis approach can be well-represented with an acyclic graph (tree), where each node in the graph corresponds to a design $\mathcal{D}$. The edge between a parent node $\mathcal{D}_1$ and a child node $\mathcal{D}_2$ describe the addition of a single part to $\mathcal{D}_1$, using corresponding connection rules, for creating $\mathcal{D}_2$. The root of the tree corresponds to a base that supports the robotic arm, and the tree terminates at *goal nodes* that correspond to valid designs, capable of executing user-specified motions (see fig. 5.3(b)). Since the design space is combinatorial, and the depth of the tree is potentially unbounded (one can keep adding more components), brute force construction and traversal of such a tree of designs is very expensive. Instead, we leverage an existing informed search method that accounts for the desirability and validity of designs during the search, called the A* algorithm [182].

5.5.2 A* search

A* is a widely used algorithm for search-based problems. It works in a best-first search manner by constructing and traversing the nodes in the tree that are most promising. Specifically, A* chooses nodes that minimize the cost function –

$$f(N) = g(N) + h(N)\,, \tag{5.1}$$

where $g(N)$ represents the current cost of the design $\mathcal{D}_N$ at node N, and $h(N)$ corresponds to a heuristics that estimates the cost of parts that are required to be added to $\mathcal{D}_N$ for generating desired designs (corresponding to goal nodes), which can execute the user-specified motion. In other words, heuristics encapsulate the deviation of $\mathcal{D}_N$ from the goal design. Since a design with fewer number of components is economical and simpler to control and fabricate, we define $g(N)$ to be proportional to the number of parts in $\mathcal{D}_N$.

$$\begin{aligned} g(N) &= \delta(\mathcal{D}_N)\,, \\ h(N) &= \mathcal{E}_{IK}(\mathcal{D}_N, \mathcal{T})\,, \end{aligned} \tag{5.2}$$

where $\delta(\mathcal{D}_N)$ computes the number of parts such as actuators and links in the design $\mathcal{D}_N$ at node N. One can customize $\delta(\mathcal{D}_N)$ to penalize parts individually by using weights corresponding to a metric of choice. For instance, by using weights corresponding to the cost of components, one can drive the search to generate designs with fewer number of costly components such as actuators. To compute the heuristics $h(N)$, we evaluate the design's ability in executing a user-specified motion trajectory $\mathcal{T} \in SE(3)$ using Inverse-Kinematics (IK). We use IK to compute the required poses of the robot design so that the robot's end-effector can follow $\mathcal{T}$ as closely as possible. A higher IK error $\mathcal{E}_{IK}$ relates to the need of adding more parts to the current design $\mathcal{D}_N$, and thereby shows the deviation of $\mathcal{D}_N$ from desired designs. Since our heuristic computation involves solving for the robot's motion, we are able to find both the robot's structure and motion for a task simultaneously.

For computing $\mathcal{E}_{IK}$, we first discretize $\mathcal{T}$ in time to obtain a set of n target frames. Let $\mathbf{q}$ represent an arbitrary pose of the robot. A robot design $\mathcal{D}$ is able to achieve the user-specified motion if there exists a set of poses $\mathbf{q}_i(1 \leq i \leq n)$ of the robot arm that enable the arm's end-effector e to achieve the target pose defined by $\mathcal{T}_i$ at i^{th} frame without collisions. This Inverse-Kinematics (IK) computation is thus solved as an optimization over robot arm's poses $\mathbf{q}$.

$$\mathcal{E}_{IK}(\mathcal{D}, \mathcal{T}) = \min_{\mathbf{q}_i} \sum_{i}^{n} \Delta(\mathcal{T}_i, e_i)^2 \tag{5.3}$$

where $\Delta(\mathcal{T}_i, e_i)$ represent the error between the pose of robot arm's end-effector e_i and the target pose specified by $\mathcal{T}_i$ at i^{th} frame. $\mathbf{q}_i$ represents the pose of the robot design $\mathcal{D}$ at i^{th} frame. $\mathcal{E}_{IK}$ is the net IK error of the design $\mathcal{D}$ while following the trajectory $\mathcal{T}$. A video showing the A* performing such a search for robot arm structures is available here[2]. Prevention of self collisions and collisions with user-specified obstacles in the environment is achieved through Bullet physics [1]. If the user-specified path passes through obstacles, the IK computation will not find a valid motion. We currently do not perform feasibility check of the trajectory $\mathcal{T}$ for avoiding such scenarios. In the future, a motion planner could be integrated within our system to create the trajectories based on even higher-level inputs from the users to prevent such scenarios.

5.5.3 Admissibility of the heuristics

Our IK error-based heuristic function (eq. 5.2, 5.3) requires careful execution with regards to two issues. First, the IK error requires each design $\mathcal{D}$ to have an end-effector attached at the rear end of the robot. End-effectors are special parts in the library such as welding machine, sealing gun etc. that are user-specified and serve a specific purpose in the task. They may not be compatible with all the other parts in the library. As a result, a design $\mathcal{D}_N$ at an intermediate node in the tree may be composed of parts that do not allow end-effector connection. Secondly, for the efficiency of A* search process, the heuristics $h(N)$ needs to be *admissible*. Without an admissible heuristic, which underestimates the actual cost of a node N, the search might traverse more nodes resulting in sub-optimal performance.

To deal with these issues, we define a *virtual* end-effector part in the library that is compatible and can attach with all other parts in the library. This not only allows computation of IK error for any intermediate designs in the tree, but also ensures admissibility. Any intermediate designs with a virtual end-effector is not feasible in the real world, and thereby its cost is an underestimate of the actual design's cost. The search termination criteria ensures that the goal nodes, which

[2]A* in action – `https://youtu.be/MOrijZu47EA`

correspond to the desired designs, consist of the user-specified end-effector instead of the virtual end-effector.

5.6 Results

We demonstrate the capability of our automatic design approach by synthesizing robotic arms with different degrees of freedom (DOF) corresponding to various trajectory following scenarios. Figure 5.4 shows some of these designs generated by our system, given a robot base position and a motion trajectory to follow in an unconstrained environment. The time taken to generate these designs depend upon the complexity of target motion trajectories (length, required DOF). Apart from complexity of the target trajectory, environmental constraints also increase the design time.

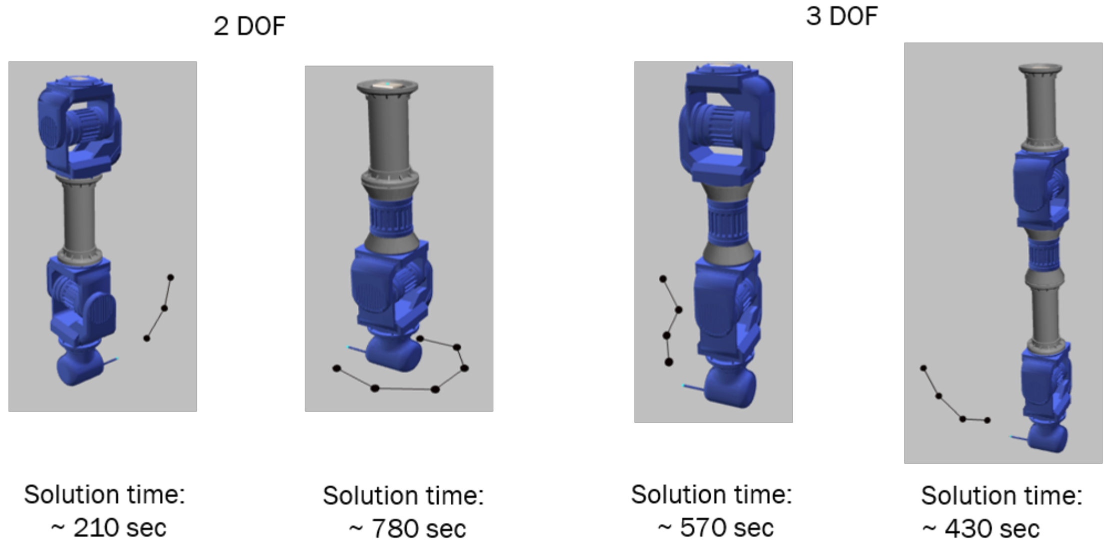


Figure 5.4: Various robotic arms with different DOF synthesized automatically with our system are shown here, along with the time taken to generate them. Corresponding user-specified target motion trajectories are shown in black. Each design is composed of modular parts such as actuators, links and end-effector. The actuators and end-effector are highlighted in blue.

In order to validate the designs generated by our system, we manually created various robotic arm designs and corresponding motion trajectories. We then used these trajectories as targets for our automatic design generation. Comparison of the automatically generated designs with the original designs further highlight the strengths of our approach. We find that our system is able to find simpler designs (with fewer DOF) as compared to the original in some cases, especially when the environment is unconstrained (see fig. 5.5(b)).

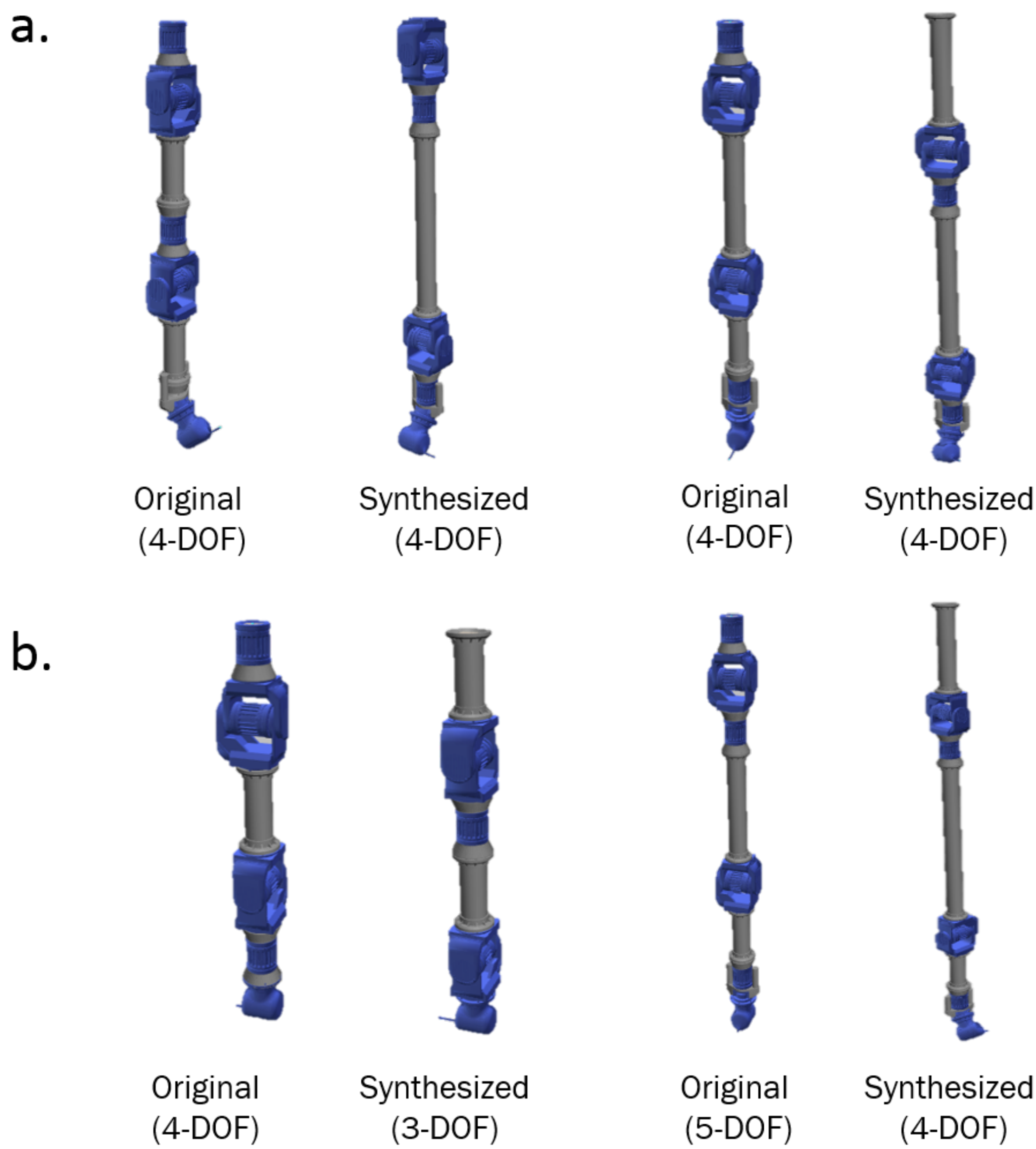


Figure 5.5: (a) To validate our approach, we synthesize arm designs to follow trajectories that correspond to manually created designs (denoted as original designs). Our approach not only generates valid designs in all cases, but also finds simpler designs (with lower DOF) to follow trajectories that were originally generated using robotic arms with higher DOF in some cases, as shown in (b). Actuators and end-effector are highlighted in blue.

5.7 Frequently asked questions (FAQ)

1. **Is it easy to specify target trajectories as input? Aren't there better alternatives to specify a robot's desired function?**
   Specifying target motion trajectories is most suitable for tasks that are purely characterized by motion. Examples include pick and place tasks, tasks involving motion along a path etc. Defining these trajectories may or may not be easy to do, depending upon the complexity of the task. Integration of a motion planner within our system will help with this in the future. We discuss this in detail in the next section (Sec. 5.8). We will also present a complementary approach that allows users to specify desired robot functionality using relevant context-specific attributes in Chapter 7.
2. **Can such an approach be extended to other types of robots?**
   Our approach can be extended for designing articulated robots beyond robotic arms, if one can define their desired robot functionality using a motion trajectory. Depending upon the type of the robot, additional physical constraints (e.g., stability for walking robots) may need to be embedded within the system. Recently, a similar approach was applied to the design of walking robots [90]. In the future, it will be essential to apply our automatic design framework to a wide class of robotic systems for testing the framework's scalability and generality.

5.8 Limitations

Limited scalability of input provided by the user: Currently, our system enables users to define the task using appropriate motion trajectories. This might be very challenging to do and might not scale in complex scenarios with large number of obstacles. One could instead obtain the necessary trajectories using a motion planner. Integration of motion planner within our framework will also eliminate other failure cases of our current framework that arise due to infeasible user-specified trajectories such as trajectories passing through obstacles. Users can then just define targets for the motion planner such as obstacles to avoid, task goals etc. Intuitively enabling users to define task-specific requirements for motion planning or general design problems is an exciting area of future research.

Limited user-in-the-loop interaction: Unlike the design auto-completion in modular structure design tool (Sec. 4.5), search for task-specific designs does not allow user-in-the-loop interactivity, owing to the higher design space complexity as well as expensive heuristics computation involving IK. Consequently, the user is unable to interact with the search for specifying less quantifiable user-preferences such as aesthetics in our current framework. Instead of synthesizing designs from scratch, one can ask users to provide an initial design along with the task description. Starting with such a design that the user prefers, one can then make only the necessary design modifications for the task. We explore such an approach in the context of legged robots in the next chapter (Chapter 6).

Limited scalability of search-based approach for complex designs: The experiments reported in this chapter are very preliminary. Our ongoing experiments with complex designs with more than 4 DOF and more than 12 modules suggest that the design time exponentially increases with design complexity, owing to the really large design space. For instance, some of our 6 DOF designs took more than 12 hours to create in these experiments. Approaches that prune the design space by injecting intuitions about the user-preferred designs therefore need to be explored in the future. Hierarchical approaches that iteratively search for designs are also worth exploring, since they could enable both user interaction and faster design times.

5.9 Publication and dissemination

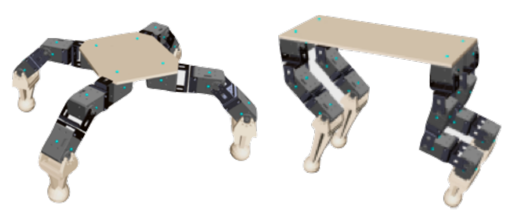
This work was undertaken in collaboration with Katharina Muelling and Margarita Safanova from National Robotics and Engineering Center (NREC). As mentioned before, it was motivated by the challenges in aerospace manufacturing. It was presented in a workshop on Autonomous Robot Design at International Conference on Robotics and Automation (ICRA) 2018. Corresponding publication is available on arXiv [61]. A video showing the automatic search-based design is available at – `https://youtu.be/MOrijZu47EA`.

Chapter 6

Co-design of structure and function for legged robots

6.1 Preamble

Our task-specific robot arm design tool described in Chapter 5 allows users to obtain designs based on high-level task requirements. In particular, with a search-based automatic design approach we enable users to find valid designs that satisfy their desired task requirements, from scratch. Unfortunately, the ability of the search-based framework to enable user-in-the-loop design or account for less quantifiable user preferences such as aesthetics is limited. For instance, given a high-level user intent of making a quadruped that walks sideways, it is unclear whether the user prefers a spider-like quadruped or a puppy-like quadruped (see inset figure). In this chapter, we therefore present a complementary task-specific co-design framework that starts with a user-defined initial design and task descriptions, and makes appropriate design modifications to achieve the user-specified task. Starting with a user preferred design (for example, a puppy-like quadruped instead of a spider-like quadruped) ensures that the resultant design matches user's intent better.



Note that, similar to modular structure design tool, the users have to specify preferred robot morphology in this framework. However, unlike modular structure design tool, users do not have to iterate over their designs themselves when their designs do not behave as desired (see Figure 6.1). Since this may be especially hard for novices to do, our approach increases accessibility of design process while accounting for complex user preferences.

6.2 Introduction

To increase the accessibility of designing complex legged robots, we present a design system that concurrently optimizes a user-specified legged robot's motion and structure to achieve a user-defined task. Since our system enables co-design based on specific task requirements, its

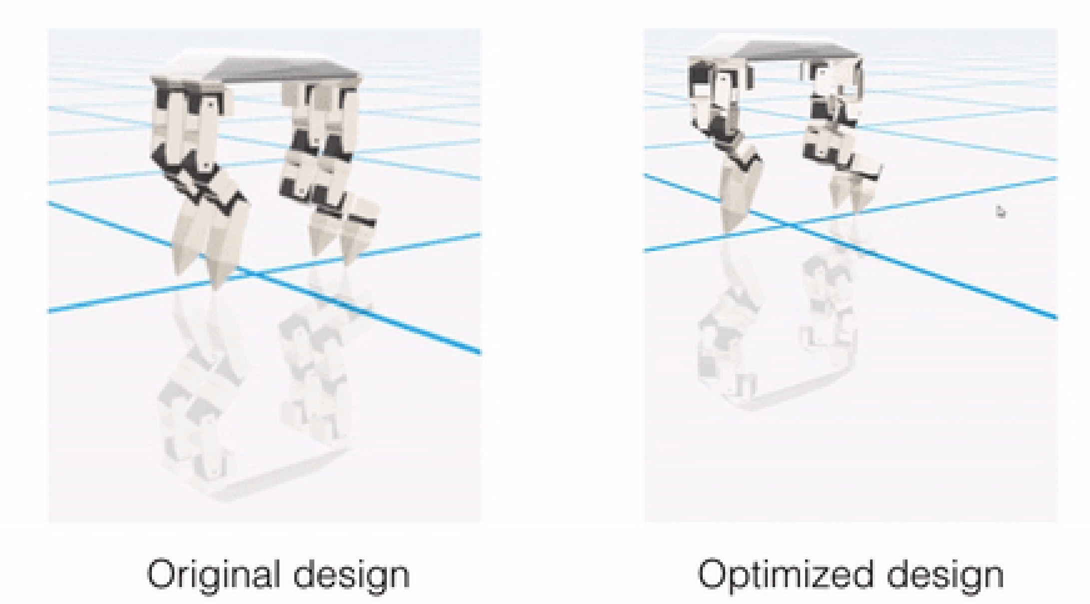


Figure 6.1: The need for co-design optimization – When user designs do not perform as expected, it might be hard for the novice users to modify their designs appropriately. We present a method for automatically updating robot design as per user-specifications. An illustrative example is shown here, wherein a robot design incapable of walking sideways is automatically updated to achieve sideways walking.

design approach can be broadly thought of as inverse design. To accomplish such design, we ask the question: can we develop mathematically-rigorous models with the predictive power to inform the design of effective legged robots?

Addressing this question, we present a mathematical model that maps the morphological parameters of a robot to its motor capabilities. Using this model as a core, we then develop a computationally efficient interactive design system that enables users to design legged robots with desired morphologies and behaviors by specifying higher level descriptions. Our framework leverages the sensitivity of robot's motion on its morphology captured by our model, to hierarchically update a robot's structure and motion to achieve a specified behavior or task performance. In particular, we focus on periodic locomotion-based tasks that are characterized by footfall patterns, walking/turning speed, and direction of motion. Our approach is therefore a departure from conventional, largely manual trial and error approaches that iteratively improve task-based robot designs. To deal with the computational complexity, and to enable user interactivity throughout the design process, we integrate the highly scalable Adjoint method [84] within our framework.

We validate our system in simulation through various task-based robot design scenarios that are challenging for casual users. We demonstrate how our system is able to aid users in dealing with a variety of such issues ranging from physical in-feasibility of the design to sub-optimality in task performance, while maintaining preferred morphology as much as possible.

6.3 Related work

Assisted robot design improvement: Recently, various approaches have been proposed to either provide feedback to the users about improving their designs [36], or for automatically changing them based on a desired outcome [89, 199]. We are highly inspired by these approaches that leverage the coupling between the robot's morphology and behavior. In particular, Canaday *et al.* and Ha *et al.* update the robot's morphology for a given behavior such that kinematic and actuator constraints are satisfied [36, 89]. On the contrary, we co-optimize the morphology and motion of a robot without assuming a predefined motion or control policy for improving the robot's performance in locomotion-based tasks, similar to [199]. However, our formulation is much more efficient than [199], making it well-suited for user-in-the-loop optimization. Unlike [199]'s approach of directly weaving in the robot's physical parameters within the robot's motion optimization framework, we establish a mapping between the robot's physical and motion parameters using the sensitivity analysis or implicit function theorem [104]. Ha *et al.* also use such a mapping between robot's form and function for optimizing robot designs that use minimal actuator forces [89]. However, they rely on the users to provide a change direction for modifying their designs. Instead, we enable our users to define their design requirements at task level using much wider variety of specifications such as desired speeds, desired directions of motion etc. We also allow the users to optimize their designs for multiple tasks, thereby automating the painstaking process of achieving optimal trade-off between possibly contradicting task requirements.

Task-specific robot co-design: Robotics community has long been interested in task-specific robot co-design from high-level descriptions. In particular, a large number of approaches based on evolutionary algorithms have been successfully used for synthesizing morphologies and controllers for a variety of robots such as virtual creatures, manipulators etc. [18, 122, 193]. Unfortunately, despite promising early results and significant increases in computing power, the field of evolutionary robotics has shown a limited ability to go beyond Sims' initial work [193] in terms of complexity and behavioral sophistication [44]. Stochastic approaches also provide limited theoretical guarantees, and are susceptible to produce designs that are often not reproducible. On the other hand, our gradient based approach is locally optimal. Furthermore, rather than synthesizing designs from scratch, which may not produce designs that appeal to the users, we enable the users to specify initial designs as well as to modify their designs at any point during the design process.

6.4 Design system overview

We build upon our previous work on modular structure design tool described in Chapter 4 that enables users to design customized articulated robots, and test them in simulation. In particular, as illustrated in Fig. 6.2(a), we adopt the GUI from our previous system because we have found it to be effective for specifying initial robot morphology, and design. The GUI consists of a design workspace (left) and a simulation workspace (right). The design workspace allows users to browse through the list of available modules such as actuators and 3D printed parts, which can be dragged and dropped into the scene, to construct and modify a robot design.

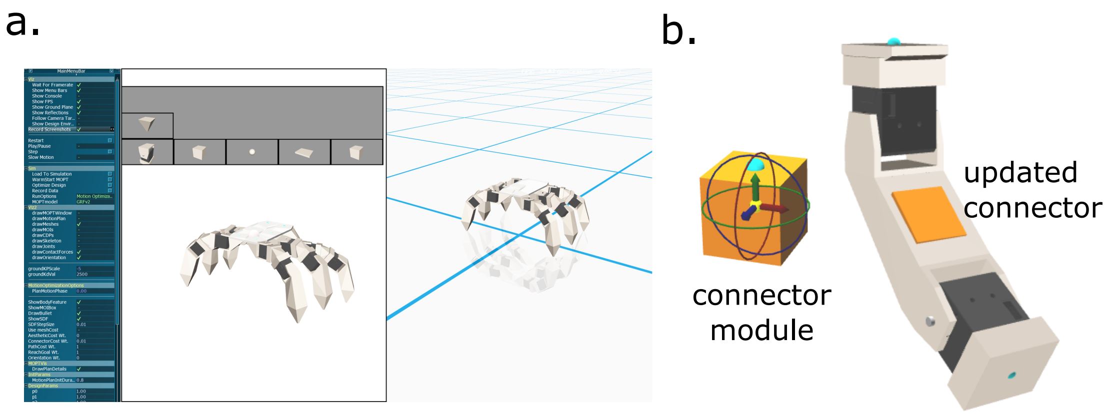


Figure 6.2: (a) Our design interface is adopted from our previous work (see Chapter 4). It allows users to design robots using modular 3D printed, and off-the-shelf parts, as well as test them using physical simulation. (b) We use a parameterized 3D printable connector module to automatically create connections between actuators. The configuration, and size of the original connector (shown in orange) gets updated based on the configuration of two actuators that it connects.

We introduce some design features within this design interface to better suit our current application. First, to enable the design of more organic looking robots that are optimized for a specific task, and are less cumbersome to assemble, we take a departure from using off-the-shelf brackets for connecting actuators. Instead, we assume that all parts of the robot's articulated structure (other than actuators for joints) are 3D printed. Second, we automatically create these 3D printable connectors between actuators. We are inspired by the robots created in [134] that create convex hull geometries between actuators. However, while [134] create these geometries as a final processing step, we enable the connector geometry update during interactive design. With every drag-and-drop user operation that changes the robot morphology, as well as during automatic design optimization, the corresponding connector geometries are updated as well.

To enable this, we define a parameterized 3D printable connector module (see fig. 6.2(b)). Parameterizing the connecting structure as a module allows us to update its position and orientation interactively with changes in the design. Each connector module is also endowed with 'virtual' attachment points on the connector's face, that get updated based on its position as well as that of the actuators it connects to. These attachment points are used to update the shape and size of the module's convex hull structure as needed. Our video[1] demonstrates how the 3D printable connecting structures of a robot's legs change as the design optimization updates the robot's design for a specific task. This allows the users to visualize the design changes as the optimization progresses. If the users disapprove of the aesthetic appearance of the robot at any point in the optimization, they can pause the optimization, and make the necessary changes, before continuing the optimization.

[1]Available at – `https://youtu.be/zrMZBgTbJho`

While the process of manually specifying the robot's initial morphology is similar to manual design in our modular structure design tool (Sec. 4.5), users no longer have to iterate over their designs to achieve desired behavior. We next present our design optimization framework, which automatically optimizes the robot's form and behavior for performing a desired task. In scenarios where the users are not satisfied with the behavior of their initial designs, they can use such an optimization to improve their designs. Fig. 6.3 illustrates this design process for creating a fast walking quadruped.

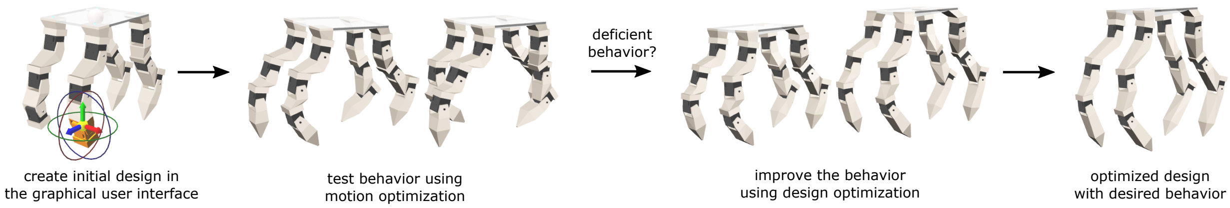


Figure 6.3: Overview of the design process – A design session with our system typically begins with users creating initial robot morphologies using our interactive GUI. The behavior of their initial designs can be tested in the physical simulation using our motion optimization framework (Sec. 6.5.5.) If the robot behavior is not as desired or deficient, instead of manually changing the design, users can use our automatic design optimization (Sec. 6.5.2) that improves their initial designs for the task at hand. In the example shown, the initial robot couldn't walk as fast as desired. The design optimization lengthens the robot limbs to enable this.

6.5 Automatic co-design optimization

The robot's structure has a huge effect on the tasks it can perform. Therefore, while designing robots for a particular task, engineers typically iterate back and forth between the robot's physical and motion design. To capture this coupling between the robot's form and function, we parameterize a robot with a set of structure parameters $\mathbf{s}$, and motion parameters $\mathbf{m}$. However, instead of treating $\mathbf{m}$ and $\mathbf{s}$ independently, our goal is to represent robot motions as a function of its structure $\mathbf{m}(\mathbf{s})$. Apart from being intuitive, such a representation allows us to solve for an optimal task-specific behavior and design hierarchically, in a computationally efficient manner. This, in turn, allows us to generate results much faster, enabling interactivity during design. We explain our formulation in detail in this section.

6.5.1 Parameterization

A larger variety of robots including manipulators, and walking robots are composed of articulated chain like structures, in particular, of serially connected and actuated links. Such robot morphologies can be well described as kinematic trees starting at the root of the robot. The design parameters $\mathbf{s}$ is used to specify the robot morphology, which is given by

$$\mathbf{s} = (l_1, \ldots, l_g, \mathbf{a}_1, \ldots, \mathbf{a}_n, b_w, b_l) \ , \tag{6.1}$$

where g is the number of links, $l_i \in \mathbb{R}$ is the length of each link, n is the number of actuators, and $\mathbf{a}_i \in \mathbb{R}^3$ is the actuator parameters. For linear actuators, $\mathbf{a}_i$ defines the 3D attachment points, while for rotary actuators, it corresponds to orientation of axis of rotation. Apart from these parameters that represent the kinematic tree morphology of the robot, we use two additional parameters b_w and b_l to represent the physical dimensions of the robot's body (width and length respectively).

Likewise, the motion parameters $\mathbf{m} = (\mathbf{P}_1, \ldots, \mathbf{P}_T)$ are defined by a time-indexed sequence of vectors $\mathbf{P}_i$, where T denotes the time for each motion cycle. $\mathbf{P}_i$ is defined as:

$$\mathbf{P}_i = \left(\mathbf{q}_i, \mathbf{x}_i, \mathbf{e}_i^1, \ldots, \mathbf{e}_i^k, \mathbf{f}_i^1, \ldots, \mathbf{f}_i^k, c_i^1, \ldots, c_i^k,\right) , \tag{6.2}$$

where $\mathbf{q}_i$ defines the pose of the robot, i.e., the position, and orientation of the root as well as joint information such as angle values, $\mathbf{x}_i \in \mathbb{R}^3$ is the position of the robot's center of mass (COM), and k is the number of end-effectors. For each end-effector j, we use $\mathbf{e}_i^j \in \mathbb{R}^3$ to represent its position and $\mathbf{f}_i^j \in \mathbb{R}^3$ to denote the ground reaction force acting on it. We also use a contact flag c_i^j to indicate whether it should be grounded ($c_i^j = 1$) or not ($c_i^j = 0$). We note that $\mathbf{s}$ remains invariant over the entire motion optimization horizon.

6.5.2 Method overview

Given an initial robot design, and a task specification, our goal is to change $\mathbf{s}$ and $\mathbf{m}$ (as defined in eq. 6.1, 6.2) to obtain a design better suited for a task. Users typically define the initial design using our graphical interface. Various task descriptions such as preferred direction of motion/action, desired movement speed, movement styles (walking, trotting, turning) etc. can also be easily specified using our interface. These task specifications can then be encoded into an evaluation criteria or cost function $F(\mathbf{s}, \mathbf{m})$. Assuming $\mathbf{p}$ to be the parameter vector containing both structure and motion parameters $\mathbf{p} = [\mathbf{s}, \mathbf{m}]$, one can search for an optimal $\mathbf{p}$ along the direction of $F(\mathbf{p})$'s gradient $\frac{\partial F}{\partial \mathbf{p}}$. However, $\mathbf{s}$ and $\mathbf{m}$ are inherently coupled. Therefore, instead of searching $\mathbf{s}$ and $\mathbf{m}$ independently, we adopt a hierarchical approach, wherein we first update $\mathbf{s}$, and then update $\mathbf{m}$ within a constrained manifold that maintains the validity and optimality of $\mathbf{m}$'s update, given $\mathbf{s}$.

By constructing a manifold of structure and motion parameters of a robot design, we can explore the sensitivity of robot's motion $\mathbf{m}$ to its structure $\mathbf{s}$. Starting with an initial design $(\mathbf{s_0}, \mathbf{m_0})$ on the manifold, one can search for $\mathbf{s}$, and corresponding $\mathbf{m}(\mathbf{s})$ on this manifold, such that $F(\mathbf{s}, \mathbf{m})$ is minimized. This dependency of $\mathbf{m}$ on $\mathbf{s}$ is captured by the Jacobian $\frac{d\mathbf{m}}{d\mathbf{s}}$ (more details in Sec. 6.5.3). This Jacobian is used to compute the search direction $\frac{dF}{d\mathbf{s}}$ for updating $\mathbf{s}$ within the manifold. However, $\frac{d\mathbf{m}}{d\mathbf{s}}$ is expensive to compute. Therefore, we further simplify this computation by using the Adjoint method (see Sec. 6.5.4). Algorithm 3 succinctly describes these steps. Note that δ_1 and δ_2 are the step sizes of the update of $\mathbf{s}$ and $\mathbf{m}$ respectively, in the desired search directions. The *update($\Delta_\mathbf{s}$)* function essentially describes the line-search procedure for δ_1 along the search direction $\Delta_\mathbf{s}$, and the corresponding update of $\mathbf{s_i}$ at each iteration i. $\Delta_\mathbf{s}$ represents the updated search direction for $\mathbf{s}$ obtained using the Limited-memory BFGS algorithm [127], given the gradient direction $\frac{dF}{d\mathbf{s}}$. N represents the number of maximum line search iterations. δ_2 is also computed using similar back-tracking line-search [151]. For each update of $\mathbf{s_i}$, multiple updates of $\mathbf{m}$ are executed to obtain the corresponding optimal $\mathbf{m_i}$. $\mathbf{m}$ is updated in the search direction defined by Newton's method wherein $\frac{\partial^2 F}{\partial \mathbf{m}^2}$, and $\frac{\partial F}{\partial \mathbf{m}}$ represent the Hessian and gradient of F with respect to $\mathbf{m}$ respectively.

Algorithm 3: Automatic motion and structure design optimization for legged robots

---

input : Initial robot design R defined with $\mathbf{s_0}$, and $\mathbf{m_0}$; Task specification $F(\mathbf{s}, \mathbf{m})$
output: $(\mathbf{s}^*, \mathbf{m}^*) \mid F(\mathbf{s}^*, \mathbf{m}^*) < threshold$
while $F(\mathbf{s_i}, \mathbf{m_i}) > threshold$ **do**
    compute $\left.\frac{dF}{d\mathbf{s}}\right|_{(\mathbf{s_i}, \mathbf{m_i})}$ using the Adjoint method
    update $\Delta_{\mathbf{s}}$ with $\frac{dF}{d\mathbf{s}}$ using L-BFGS algorithm
    update $(\mathbf{s_i}, \mathbf{m_i})$ using `update` $(\Delta_{\mathbf{s}})$
end
Function *update(*$\Delta_{\mathbf{s}}$*)*
    $\delta_1 = 1$
    for N **do**
        $\mathbf{s}' = \mathbf{s_i} - \delta_1 \Delta_{\mathbf{s}}$
        $\mathbf{m}' = \mathbf{m_i} - \delta_2 \left(\frac{\partial^2 F}{\partial \mathbf{m}^2}\right)^{-1} \frac{\partial F}{\partial \mathbf{m}}$ (Newton's method)
        if $F(\mathbf{s}', \mathbf{m}') < F(\mathbf{s_i}, \mathbf{m_i})$ **then**
            $\mathbf{m_i} = \mathbf{m}', \mathbf{s_i} = \mathbf{s}'$
            return $(\mathbf{s_i}, \mathbf{m_i})$
        else
            $\delta_1 = \frac{\delta_1}{2}$
        end
    end
    return $(\mathbf{s_i}, \mathbf{m_i})$
end

6.5.3 Coupling form and function for robot design

It is hard to analytically represent the dependency or sensitivity of robot's motion on its structure. Instead, we assume a manifold that relates robot's structure and behavior capabilities, given a specific task.

$$\mathbf{G}(\mathbf{s}, \mathbf{m}) = 0, \tag{6.3}$$

where $\mathbf{G}(\mathbf{s}, \mathbf{m}) : \mathbb{R}^{n_s} \times \mathbb{R}^{n_m} \to \mathbb{R}^{n_m}$. It allows us to understand the effect of design parameters on the motion/behavior of the robot. Such an implicit manifold between structure and function can be converted into an explicit relation between the two within a small region around a point $P_0(\mathbf{s_0}, \mathbf{m_0})$ on the manifold, using the Implicit function theorem [104]. The theorem states that when we change $\mathbf{s_0}$ and $\mathbf{m_0}$ by $\Delta\mathbf{s}$ and $\Delta\mathbf{m}$, the change in the function $\Delta\mathbf{G}$ should be zero to remain on the manifold. Using chain rule to compute $\Delta\mathbf{G}$, we obtain the following explicit relation between $\Delta\mathbf{s}$ and $\Delta\mathbf{m}$.

$$\begin{aligned}\Delta\mathbf{G} &= \frac{\partial\mathbf{G}}{\partial\mathbf{s}}\Delta\mathbf{s} + \frac{\partial\mathbf{G}}{\partial\mathbf{m}}\Delta\mathbf{m} = 0 \\ \Delta\mathbf{m} &= -\left(\frac{\partial\mathbf{G}}{\partial\mathbf{m}}\right)^{-1}\frac{\partial\mathbf{G}}{\partial\mathbf{s}}\Delta\mathbf{s}\end{aligned} \tag{6.4}$$

where $\left(\frac{\partial\mathbf{G}}{\partial\mathbf{m}}\right)$, and $\left(\frac{\partial\mathbf{G}}{\partial\mathbf{s}}\right)$ represents the Jacobian of $\mathbf{G}(\mathbf{s},\mathbf{m})$ with respect to $\mathbf{m}$, and $\mathbf{s}$ respectively.

In order to compute such a manifold, we start with a task-specific cost function $F(\mathbf{s},\mathbf{m})$. For each robot morphology defined by $\mathbf{s}$, there exists an optimal $\mathbf{m}^*$ that minimizes $F(\mathbf{s},\mathbf{m})$. Therefore, the gradient of F with respect to $\mathbf{m}$ at point $(\mathbf{s},\mathbf{m}^*)$ should be zero. One can then search for an optimal $\mathbf{s}^*$ along the manifold defined by this gradient $\mathbf{G}(\mathbf{s},\mathbf{m}) = \frac{\partial F(\mathbf{s},\mathbf{m})}{\partial\mathbf{m}}$. An optimal $\mathbf{s}^*$ on such a $\mathbf{G}(\mathbf{s},\mathbf{m})$ would automatically ensure a corresponding valid and optimal $\mathbf{m}^*$ for the task.

For searching such an optimal $\mathbf{s}^*$, we therefore need to solve the following optimization problem:

$$\begin{aligned} &\min_{s} F(\mathbf{s},\mathbf{m}) \\ s.t. \quad &\mathbf{G}(\mathbf{s},\mathbf{m}) = 0\end{aligned} \tag{6.5}$$

where $F(\mathbf{s},\mathbf{m}) : \mathbb{R}^{n_s} \times \mathbb{R}^{n_m} \to \mathbb{R}$ is the energy function; $\mathbf{G}(\mathbf{s},\mathbf{m}) : \mathbb{R}^{n_s} \times \mathbb{R}^{n_m} \to \mathbb{R}^{n_m}$ denotes the gradient of energy function with respect to motion parameters $\mathbf{m}$ and thus $\mathbf{G} = \frac{\partial F}{\partial\mathbf{m}}$. Empowered by the Jacobian $\frac{d\mathbf{m}}{d\mathbf{s}}$ that essentially encodes $\mathbf{m}(\mathbf{s})$ (defined by eq. 6.4), we can define the search direction for $\mathbf{s}$ as follows:

$$\begin{aligned}\frac{dF}{d\mathbf{s}} &= \frac{\partial F}{\partial\mathbf{m}}\frac{d\mathbf{m}}{d\mathbf{s}} + \frac{\partial F}{\partial\mathbf{s}} \\ &= -\frac{\partial F}{\partial\mathbf{m}}\left(\frac{\partial\mathbf{G}}{\partial\mathbf{m}}\right)^{-1}\frac{\partial\mathbf{G}}{\partial\mathbf{s}} + \frac{\partial F}{\partial\mathbf{s}}.\end{aligned} \tag{6.6}$$

6.5.4 The Adjoint method

Computing $\frac{dF}{d\mathbf{s}}$ requires the calculation of the Jacobian $\frac{d\mathbf{m}}{d\mathbf{s}}$ which is computationally very expensive. It requires solving n_s linear equations (for each column in Jacobian matrix $\frac{\partial\mathbf{G}}{\partial\mathbf{s}}$), and the procedure gets very costly for large n_s. Instead, we use the Adjoint method to efficiently compute the gradient $\frac{dF}{d\mathbf{s}}$. This method formulates the computation of the gradient as a constrained optimization problem, and then uses the dual form of this optimization problem for faster computation [84]. Other applications have also sought out the Adjoint method for similar purposes in the past [133].

In particular, the expression $-\frac{\partial F}{\partial\mathbf{m}}\left(\frac{\partial\mathbf{G}}{\partial\mathbf{m}}\right)^{-1}$ in eq. 6.6 can be interpreted as the solution to the linear equation

$$\left(\frac{\partial\mathbf{G}}{\partial\mathbf{m}}\right)^{\intercal}\lambda = -\left(\frac{\partial F}{\partial\mathbf{m}}\right)^{\intercal}, \tag{6.7}$$

where $^\intercal$ is the matrix transpose operator. The vector λ is called the vector of *adjoint variables* and the linear equation is called the *adjoint equation*. Finally, $\frac{dF}{d\mathbf{s}}$ takes on the following form

$$\frac{dF}{d\mathbf{s}} = \lambda^\intercal \frac{\partial \mathbf{G}}{\partial \mathbf{s}} + \frac{\partial F}{\partial \mathbf{s}} \,. \tag{6.8}$$

Such a computation of $\frac{dF}{d\mathbf{s}}$ now involves solving only one linear equation (eq. 6.7), followed by one matrix-vector multiplication and one vector addition (eq. 6.8). This is much more efficient as compared to solving n_s linear equations for $\frac{d\mathbf{m}}{d\mathbf{s}}$ computation earlier.

6.5.5 Motion optimization

So far, we have described our framework to optimize the structure and motion of a robot, given a task. We used a cost function $F(\mathbf{s}, \mathbf{m})$ to encode the task specifications. We now describe how $F(\mathbf{s}, \mathbf{m})$ is constructed. To this end, we use a set of objectives that capture users' requirements, and constraints that ensure task feasibility.

Objectives

We allow the users to define various high-level goals to be achieved by their robot designs such as moving in desired direction with specific speeds, different motion styles, etc. To capture the desired direction and speed of motion, we define the following objectives:

$$\begin{aligned} E_{\text{Travel}} &= \frac{1}{2}||\mathbf{x}_T - \mathbf{x}_1 - \mathbf{d}^D||^2 \,, \\ E_{\text{Turn}} &= \frac{1}{2}||\tau(\mathbf{q}_T) - \tau(\mathbf{q}_1) - \tau^D||^2 \,, \end{aligned} \tag{6.9}$$

where $\mathbf{x_i}$ is the robot's COM as defined in eq. 6.2, $\tau(\mathbf{q_i})$ is the turning angle computed from pose $\mathbf{q_i}$, while $\mathbf{d}^D$ and τ^D are desired travel distance and turning angles respectively. E_{Travel} ensures that the robot travels a specific distance in desired time, while E_{Turn} can be used to make a robot move on arbitrary shaped paths.

Motion style is highly effected by gait or foot-fall patterns that define the order and timings of individual limbs of a robot. We internally define various foot-fall patterns for different motion styles such as trotting, pacing, and galloping. When users select a specific motion style, our system automatically loads the necessary foot-fall patterns, thereby allowing novice users to create many expressive robot motions. Motion style is also effected by robot poses. For expert users, we allow the capability to specify and achieve desired poses, if needed, using the following objectives:

$$\begin{aligned} E_{\text{StyleCOM}} &= \frac{1}{2}\sum_{i}^{T} ||\mathbf{x}_i - \mathbf{x}_i^D||^2 \,, \\ E_{\text{StyleEE}} &= \frac{1}{2}\sum_{i}^{T}\sum_{j}^{k} ||\mathbf{e}_i^j - {\mathbf{e}_i^j}^D||^2 \,, \end{aligned} \tag{6.10}$$

where k is the number of end-effectors, $\mathbf{x}_i^D$ and $\mathbf{e}_i^D$ represent desired robot COM, and end-effector positions respectively. Apart from these, motion smoothness is often desired by the users, which is encoded by the following objective:

$$E_{\text{Smooth}} = \frac{1}{2}\sum_{i=2}^{T-1} ||\mathbf{q}_{i-1} - 2\mathbf{q}_i + \mathbf{q}_{i+1}||^2 \,. \tag{6.11}$$

Constraints

We next define various constraints to ensure that the generated motion is stable.

Kinematic constraints: The first set of constraints ask the position of COM, and end-effectors to match with the pose of the robot. For every time step i, and end-effector j:

$$\begin{aligned} \varphi_{COM}(\mathbf{q}_i) - \mathbf{x}_i &= 0\,, \\ \varphi_{EE}(\mathbf{q}_i)^j - \mathbf{e}_i^j &= 0\,, \quad \forall j, \end{aligned} \tag{6.12}$$

where φ_{COM} and φ_{EE} are forward kinematics functions outputting the position of COM and end-effectors respectively.

We also have a set of constraints that relate the net force and torque to the acceleration and angular acceleration of the robot's COM:

$$\begin{aligned} \sum_{j=1}^{k} c_i^j \mathbf{f}_i^j &= M\ddot{\mathbf{x}}_i\,, \\ \sum_{j=1}^{k} c_i^j(\mathbf{e}_i^j - \mathbf{x}_i^j) \times \mathbf{f}_i^j &= \mathbf{I}\ddot{\mathbf{o}}_i\,, \end{aligned} \tag{6.13}$$

where M is the total mass of the robot, and $\mathbf{I}$ is the moment of inertia tensor. The acceleration $\ddot{\mathbf{x}}_i$ can be evaluated using finite differences: $\ddot{\mathbf{x}}_i = (\mathbf{x}_{i-1} - 2\mathbf{x}_i + \mathbf{x}_{i+1})/h^2$, where h is the time step. Similarly, the angular acceleration $\ddot{\mathbf{o}}_i$ can be expressed as $\ddot{\mathbf{o}}_i = (\mathbf{o}_{i-1} - 2\mathbf{o}_i + \mathbf{o}_{i+1})/h^2$. We note that the orientation of the root $\mathbf{o}_i$ is part of the pose $\mathbf{q_i}$, and it uses axis-angle representation.

Friction constraints: To avoid foot-slipping, we also have the following constraints for each end-effector j:

$$c_i^j(\mathbf{e}_{i-1}^j - \mathbf{e}_i^j) = 0,\; c_i^j(\mathbf{e}_i^j - \mathbf{e}_{i+1}^j) = 0\,, \tag{6.14}$$

for all $2 \leq i \leq T-1$, which implies that the end-effectors are only allowed to move when they are not in contact with the ground. Further, to account for different ground surfaces, we enforce the friction cone constraints:

$$f_{i\,\parallel}^j \leq \mu f_{i\,\perp}^j\,, \tag{6.15}$$

where $f_{i\,\parallel}^j$ and $f_{i\,\perp}^j$ denote the tangential and normal component of $\mathbf{f}_i^j$ respectively, and μ is the coefficient of friction of the ground surface.

Limb collisions: For physical feasibility, we propose a collision constraint that ensures a safe minimum distance between the limb segments of robot over the entire duration of the motion.

$$d(\mathbf{V}_i^{k_1}, \mathbf{V}_i^{k_2}) \geq \delta \,, \tag{6.16}$$

where $\mathbf{V}_i^k$ represents a 3D segment representing the position and orientation of k^{th} limb, $d(\cdot)$ computes the distance between k_1 and k_2 limbs, and δ is the threshold distance beyond which collisions may happen.

Motion periodicity: If the users prefer a periodic motion, we can add an additional constraint that relates the start pose $\mathbf{q}_1$ and the end pose $\mathbf{q}_T$ of the robot:

$$\mathbf{J}(\mathbf{q}_T) - \mathbf{J}(\mathbf{q}_1) = 0 \,, \tag{6.17}$$

where $\mathbf{J}(\mathbf{q_i})$ extract the orientation of the root and joint parameters from pose $\mathbf{q_i}$.

Finally, $F(\mathbf{s}, \mathbf{m})$ is computed as the sum of the objectives in eq. 6.9, 6.10, 6.11. Quadratic penalty costs for violation of constraints in eq. 6.12, 6.13, 6.14, and 6.17 are also added to F. Note that, while these objective terms are defined using the motion parameters $\mathbf{m}$, F is indirectly affected by the structural parameters $\mathbf{s}$ as discussed in Sec. 6.5.3.

6.6 Evaluation

We explore three simulated examples to study the utility and effectiveness of our approach. Although, we only show our current findings in simulation, we have confirmed that the simulation environment matches physical results, in our past work (see Sec. 4.6 in Chapter 4). The first example is aimed to show how our design optimization can change an initial robot design that is inadequate to perform a user-specified task. The second example shows how our system modifies design to ensure feasibility while maintaining task performance. Finally, we show how our system can be used to optimize designs for multiple tasks. We also demonstrate how different tasks may require varied design changes, and how our optimization can automatically generate these design changes in matters of minutes, thereby illustrating its computational efficiency. Note that all the initial designs for these examples were manually designed with our interface. Our video[2] demonstrates this process for the robot in fig. 6.4(a).

Improving inadequate designs: When novices design robots, it can be hard for them to decide where the actuators should be located and how they should be oriented for achieving a specific behavior. Figure 6.4(a) shows one such example of a 'puppy' robot with three motors per leg. Even with enough number of actuators, the robot can only walk in one direction (forward) owing to its actuator placements. In particular, all actuators rotate about the same axis thereby reducing the effective degrees of freedom (DOF) of each limb. By parameterizing the actuator orientations $\mathbf{a_i}$ in eq. 6.1, we enable our design optimization to change them for equipping the robot to walk in any specific direction. Without the optimization of such structural parameters, it may be impossible to achieve such tasks. For instance, the optimization of only motion parameters for the initial 'puppy' design fails to produce the desired behavior of walking sideways at a specific speed (see fig. 6.4(b)).

[2]Available at – `https://youtu.be/zrMZBgTbJho`

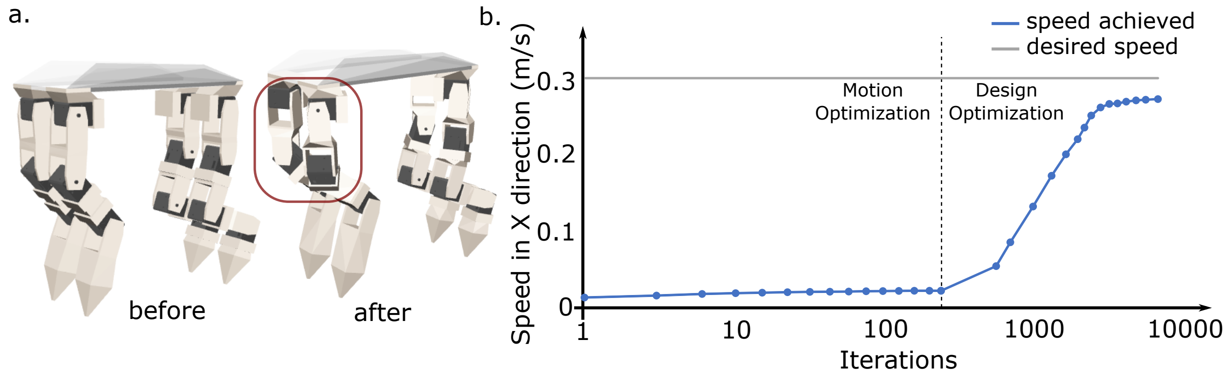


Figure 6.4: Design optimization for achieving desired behavior – (a) Initial and optimized designs of a 'puppy' robot are shown. The initial design can only walk forward (in Z direction) owing to the deficient placement of actuators at its joints. Our design optimization changes the actuator orientations to enable the robot to walk sideways (in X direction). (b) Optimizing motion parameters is not enough for overcoming such design deficiencies as seen in the initial iterations of the plot. On the other hand, optimization of the structure parameters modifies the design to walk sideways with desired speed. Figure 6.1 shows the original, and optimized designs in action.

Achieving desired task performance, while ensuring feasibility: Even when a design can theoretically achieve the desired behavior, it may be rendered infeasible due to real world constraints such as collisions. Fig. 6.5(a) shows a robot that can walk in the user-specified direction at desired speeds. However, when walking speeds increase above 0.1 ms^{-1}, the robot's limbs start colliding. It is hard to anticipate such issues apriori. Along with helping the user to test such scenarios in simulation, our system can also automatically prevent them by using feasibility constraints during motion optimization as discussed in Sec. 6.5.5. However, these constraints prevent the required range of limb motions needed for fast walking, limiting the ability of the robot to perform the intended task (see fig. 6.5(b)). On the other hand, design optimization is able to change the design to achieve both these contradicting tasks successfully as seen in fig. 6.5(c).

Designing for multiple behaviors: Finally, designing robots for multiple tasks is also highly challenging, especially if the tasks ask for opposing design characteristics. Consider the task of walking and pacing for a quadruped robot shown in fig. 6.6(a). The original design can only walk forward owing to its actuator placements (similar to the 'puppy' robot in fig. 6.4(a)). Its wider body and shorter limbs prevent it from pacing in stable manner. The tasks are further challenging because of limited number of actuators. While three actuators may enable sufficient DOF for 3D movements without careful placement of individual actuators, same may not be true for designs with lower number of actuators. Individually optimizing the design for pacing and walking may not be sufficient for enabling the robot to perform both tasks. Pace based design optimization generates a slim bodied robot, while walk based design optimization produces a wider body size to increase stability during fast walking (see fig. 6.6(a)). Such a wider body in turn, negatively affects the pacing behavior (fig. 6.6(b)). To achieve reasonable performance for both these tasks, a trade-off is thus required. Our system allows users to jointly optimize their

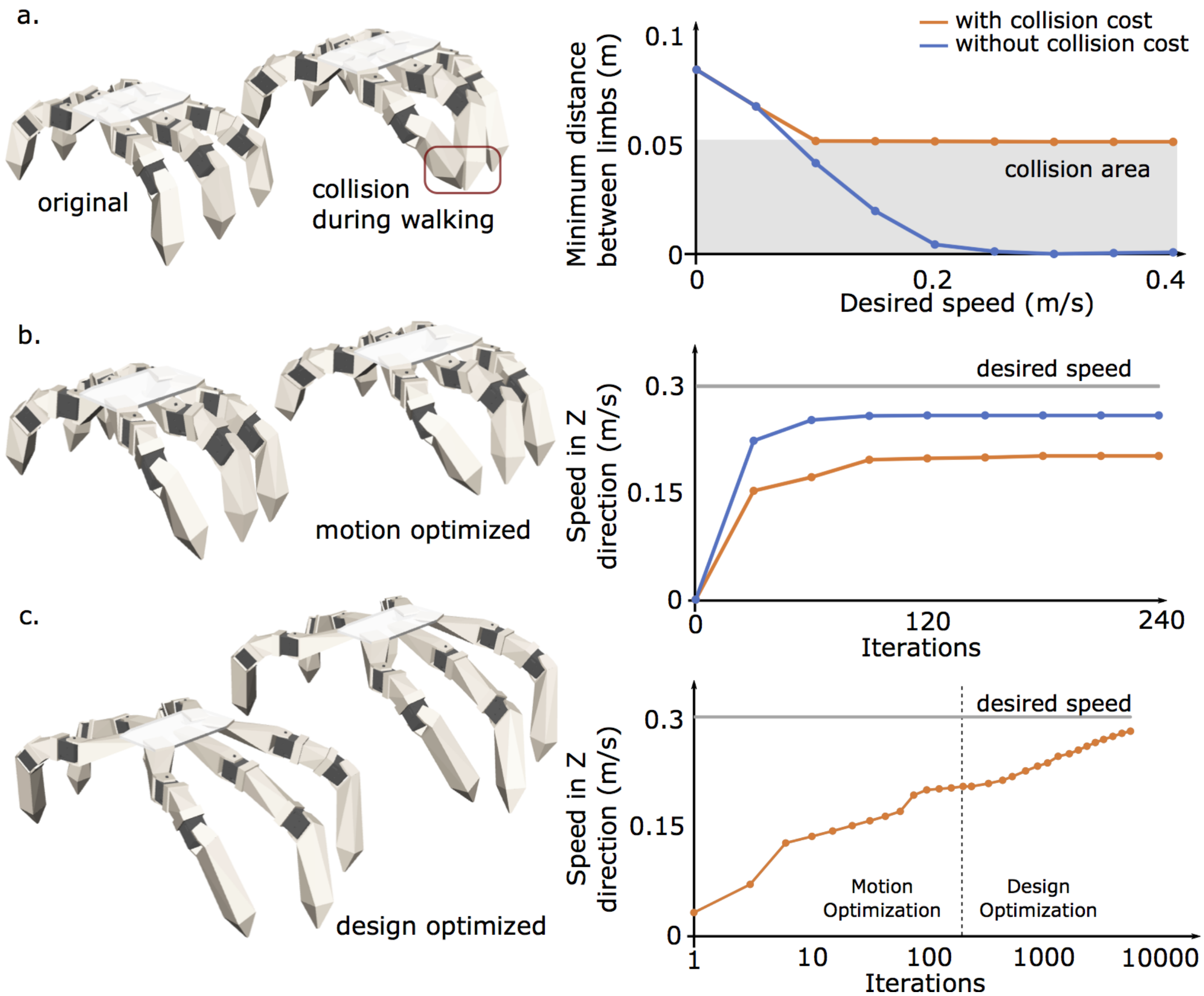


Figure 6.5: Achieving feasible behavior, and desired task performance simultaneously with design optimization – (a) A hexapod robot's limbs start colliding at higher speed. Accounting for collisions during motion optimization prevents this, but also restricts the robot from walking faster as seen in the plots in (a), and (b). (c) Design optimization is able to deal with this trade-off by increasing the spacing between limbs, and their lengths. This enables the robot to walk faster without any collisions. Our video illustrates this example in detail.

designs for multiple tasks, to handle such scenarios. The individual requirements for each task $F_i(\mathbf{s}, \mathbf{m_i})$ can be combined in weighted manner into $F_{joint}(\mathbf{s}, \mathbf{m}) = \sum w_i F_i(\mathbf{s}, \mathbf{m_i})$. Weights w_i representing the importance of each task can be set by the users. Such joint optimization of walking and pacing (with $w_1 = w_2 = 0.5$) for quadruped in fig. 6.6(a) succeeds in achieving the necessary trade-off as illustrated in the resultant medium bodied optimized design, and the corresponding task performance.

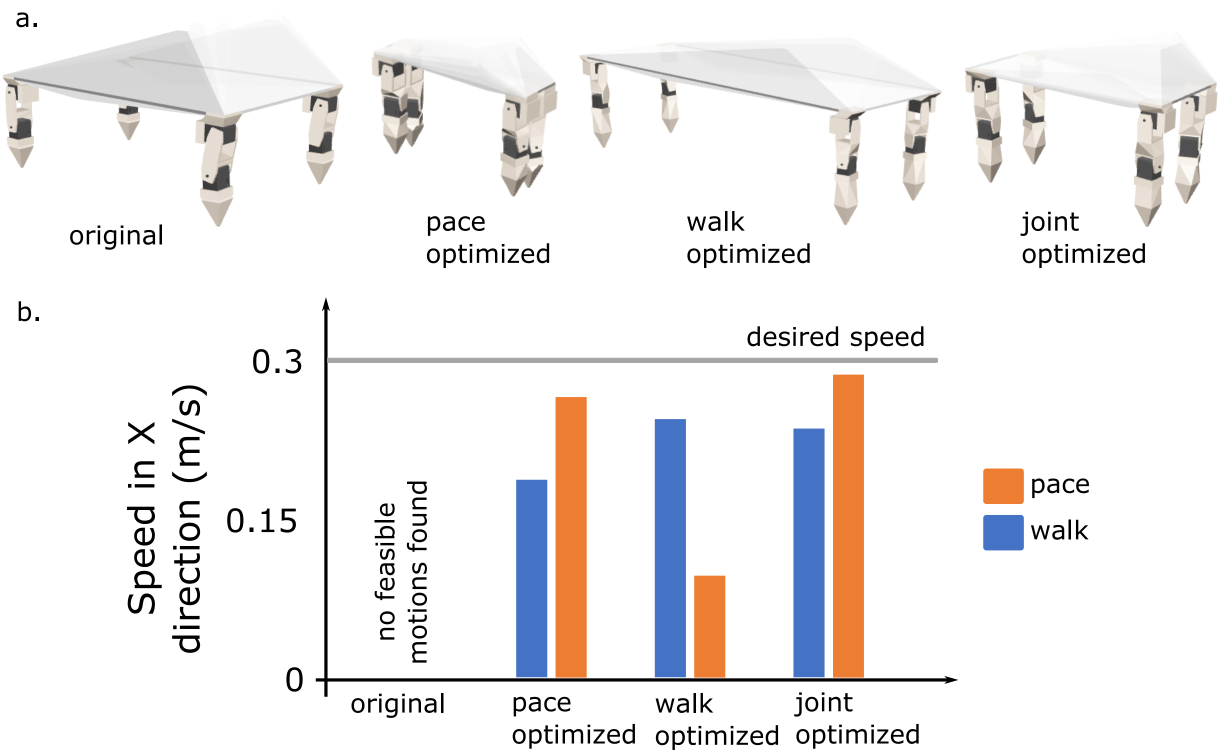


Figure 6.6: Designing for multiple behaviors – (a) An initial quadruped robot design that can only walk forward, is optimized to pace and walk sideways respectively. A design that was jointly optimized for both these behaviors simultaneously is also shown. Note that the body aesthetics was manually added to each design (b) Unlike designs generated by optimization that considered the tasks of walking and pacing separately, jointly optimized design achieves a reasonable trade-off between the performance of both tasks.

Example statistics: Table 6.1 shows the design times for optimizing the designs of robots in fig. 6.4, 6.5, 6.6. For quadruped in fig. 6.6 these statistics are reported for the joint optimization scenario. Note that, even when its number of optimization parameters are roughly similar to that of the hexapod, there is a significant difference in the number of optimization iterations, and the time required. This is because of the contradicting requirements that the two tasks demand, making the problem more challenging. Also note that for each iteration of design optimization, multiple iterations of motion optimization are executed (see Algorithm 3). However, as illustrated in the statistics, the large number of these iterations needed to update the designs are executed in matter of minutes. Such computational efficiency is at the core of interactivity in our system. Apart from an efficient implementation in C++, a scalable approach using the Adjoint method

Robot	Number of Parameters	Motion Opt. Iterations	Design Opt. Iterations	Time (s)
Puppy (Fig. 6.4)	614	6207	32	107.66
Hexapod (Fig. 6.5)	1044	5013	53	97.47
Quadruped (Fig. 6.6)	1050	14873	100	124.47

Table 6.1: Design optimization statistics for example robots. All experiments were run on a standard desktop with a 3.6 GHz i7 CPU and 16 GB RAM.

allows us to achieve the same. While it is hard to make a direct comparison owing to differences in parameterization and implementation, our design times are significantly faster than the current state-of-the-art that co-optimizes form and function for many such similar robots [199]. For instance, their system took on an average 685 seconds to optimize a biped robot design for the task of walking on flat ground (750 optimization parameters) [199].

6.7 Frequently asked questions (FAQ)

1. **Why are there no hardware experiments to support the system?**
   We used the same simulation framework as in our modular design system (Chapter 4), which was validated with hardware prototypes. While the design optimization will benefit from a hardware evaluation, it is unclear how to perform such an evaluation owing to the diversity of design failures as seen in the Section 6.6.
2. **What about local minima? Can the system get stuck in one?**
   Like any gradient-based approach, our system will only find locally optimal solutions. In practice, if such a solution does not satisfy user requirements, users can either make small modifications in their designs, or change the threshold in Algorithm 3 and rerun the optimization. However, none of our experiments described in Sec. 6.6 required such random restarts.
3. **Does the system consider robot dynamics?**
   We do not account for dynamic factors such as inertial and Coriolis forces on the limbs in our system. Our current focus is on kinematic design of robots. A purely kinematic approach may not be sufficient for supporting expert designers in robot development, owing to the load and performance requirements of designs for their applications. However, our kinematic design approach can cater to a wide variety of robot design needs of novices as highlighted in various design scenarios in Sec. 6.6 as well as by our modular design system in Chapter 4.

6.8 Limitations

Limited behavior support: Our motion optimization framework currently only supports periodic locomotion-based tasks. A generic system would need to account for a much broader class of motions and behaviors, including climbing, carrying weights or avoiding obstacles. Like with the task-specific design tool in Chapter 5, testing out this framework on other types of robots would be important for its scalability and generality.

Limiting assumptions: Our system also currently does not perceive the mass or the number of motors as design parameters. However, our approach is generic, in that the adjoint method can be applied to any trajectory optimization scheme that provides analytical gradients and Hessians. Therefore, although our motion optimization currently does not account for the mass of the limbs, we believe that this limitation can be eliminated in the future.

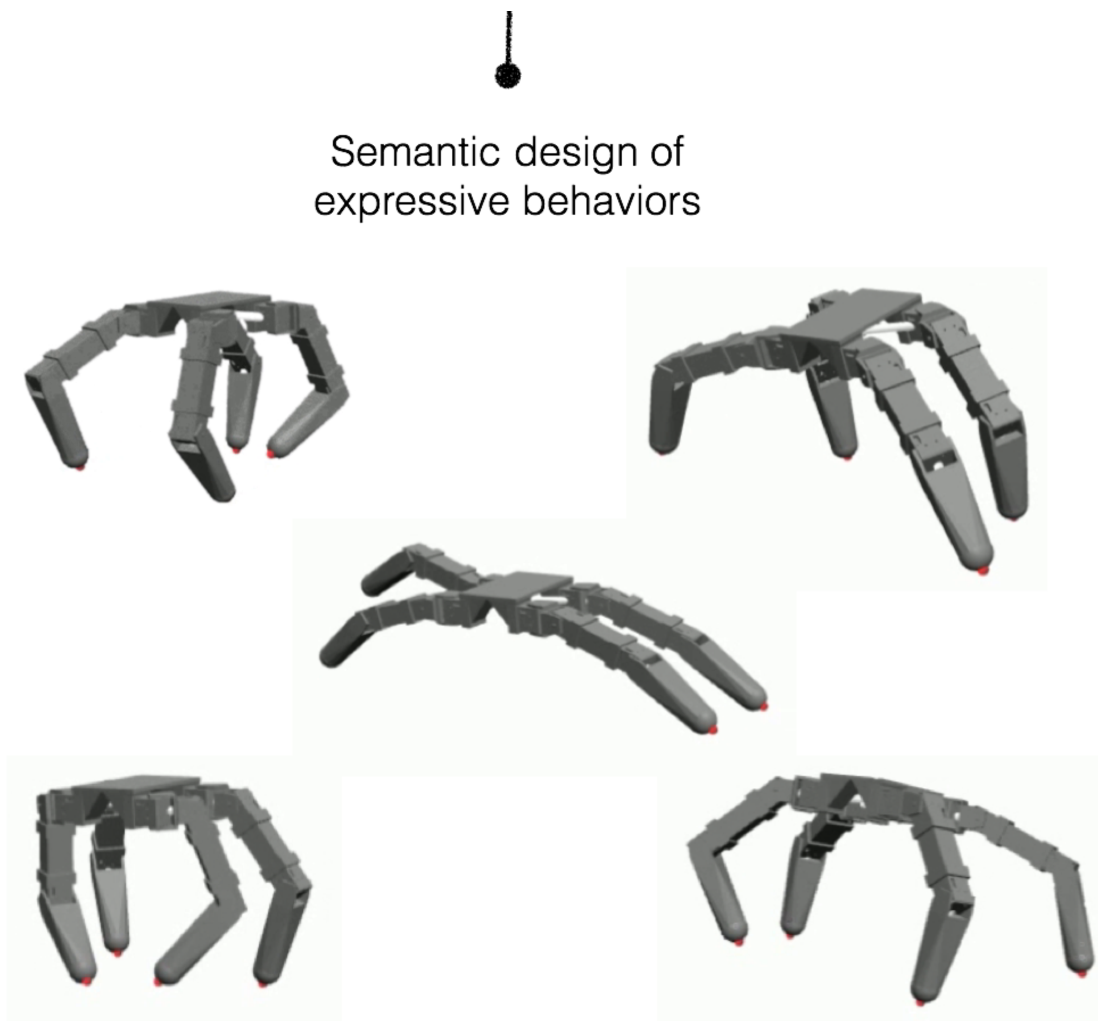
Limited transparency in user control: The optimization processes are relatively invisible to the user in our system, which provides little room of manipulation for experts and specialists who may desire finer control over the design process. We therefore believe that it is important to find the right balance between automation and user control during design. A detailed study may be required towards better understanding this balance for novice, intermediate, and experienced robot designers, for well-informed design of future tools.

6.9 Publication and dissemination

This is joint work with Beichen Li (Tshingua University), and Ye Yuan (Carnegie Mellon University). A short version of this paper was published in International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines (CLAWAR) 2018. At CLAWAR, it was awarded second place in the best technical paper award category. A video showing various design optimization examples, as well as an illustrative design session can be found here – `https://youtu.be/zrMZBgTbJho`. The design optimization approach was also integrated into another system developed in the lab for designing skating robots, published in SIGGRAPH 2018 [82].

Part III

Function design tools



Chapter 7

Geppetto: Semantic design of expressive robot behaviors

7.1 Preamble

We now describe our functionality editing tool for designing expressive behaviors of articulated robots. Unlike our previous tools, this tool doesn't change the structure of the robot. Instead, it focuses on enabling users to edit the motion behavior of the robots intuitively. The two major features of this tool, towards developing accessible robotics tools for novices, are: (a) semantic design for capturing high-level user intent, and (b) visual design space exploration of a high-dimensional, complex, and non-intuitive design space.

To enable robot behavior design using high-level and semantic descriptions of behavior properties such as the desired emotional expression, we devise a data-driven and simulation-powered framework. In particular, our system combines a physics-based simulation that captures the robots motion capabilities, and a crowd-powered framework that extracts relationships between the robots motion parameters and the desired semantic behavior. By leveraging these relationships for a mixed-initiative design, the system guides users to explore the space of possible robot motions. Our design system breathes life into robots by endowing them with expressive behaviors, and hence is named after the fictional character *Mister Geppetto*, who similarly brought the inanimate character Pinocchio to life [65].

7.2 Introduction

As robots become more prevalent in social environments, from factory floors to personal homes, endowing robots to express themselves can enhance and enrich our experience of interactions with them. The paradigm of enabling robots to express intent and emotions via movements is extremely generic and powerful [69, 125, 146, 174]. Instead of relying on anthropomorphic features or morphology, this paradigm leverages the human ability of identifying emotions and intent from mere behavior for establishing meaningful communication during interactions [10, 79, 224]. For instance, a robotic arm that collaborates with human workers on a factory floor could communicate its confusion about a task, or alert human workers if needed, by moving in a specific manner.

However, creating such expressive movement behaviors for robots is highly challenging [34]. Similar to digital character animation, creating behaviors for robotic characters requires tremendous skill and effort [59]. Apart from the inherent task complexity and domain knowledge requirements, robot behavior design also suffers from the lack of suitable design tools. Existing animation tools such as Blender [31], or Maya [21], enable design with absolute human control but offer limited options for integration with physical hardware. On the other hand, conventional robot control tools (e.g., ROS [178]) have extensive support for robot's physical simulation and control, but do not allow for mixed-initiative expressive behavior design. In comparison, our goal is to facilitate easy and intuitive design of expressive movements for robotic systems over a wide variety of applications ranging from art to social interactions.

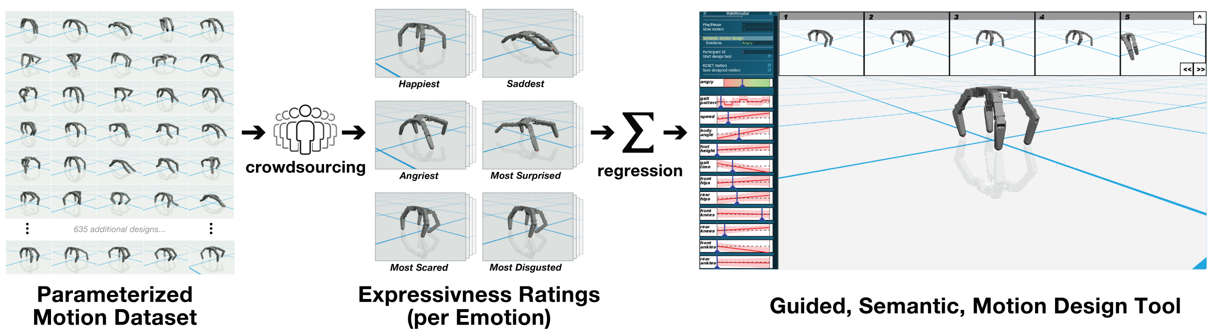


Figure 7.1: Overview of our semantic motion design framework – Our framework consists of four main building blocks: (a) a dataset of parameterized expressive robot motions, (b) a crowdsourcing set-up for estimating the emotional perception of motions in the dataset, (c) regression analysis for establishing relationships between motion parameters and the emotional perception of the resultant motion, and (d) an intuitive design tool backed by these data-driven parameter-emotion relationships.

Guided by feedback from a systematic survey of experts from animation, art, and robotics, we attempt to fill in this gap present in existing robot behavior design tools. We present *Geppetto*, a simulation-driven robot motion design system that enables the design of expressive behaviors using high-level and semantic descriptions of behavior properties. *Geppetto* currently explores the creation of behaviors that convey emotions, which is an important and challenging problem within Human-Robot Interaction (HRI) [106, 197]. Beyond emotional expression, *Geppetto*'s framework could easily be extended to support other semantic descriptions related to how a robot behaves, or what its movements should look like. Apart from physics-based motion simulation, *Geppetto* builds upon two recent advances in HCI and graphics research - Crowd- powered Parameter Analysis [116] and Semantic Editing [239]. These techniques are combined into a novel data-driven framework for the domain of robot behavior design.

Inspired by the work of Koyama *et al.* [116], crowdsourcing is used to obtain subjective scores pertaining to the perceptual quality of emotional expression for a generated dataset of parameterized robot motions. Using regression analysis, functional relationships are inferred between robot motion parameters and the corresponding emotional expressions. Using these relationships, a semantic interface is developed to enable gently guided intuitive editing, and visual exploration of the space of possible robot motions (Figure 7.1, right). A mixed-initiative approach is used for handling the unique properties of our data, such as the noise from crowd-sourcing, and the inherent subjectivity of emotional behaviors.

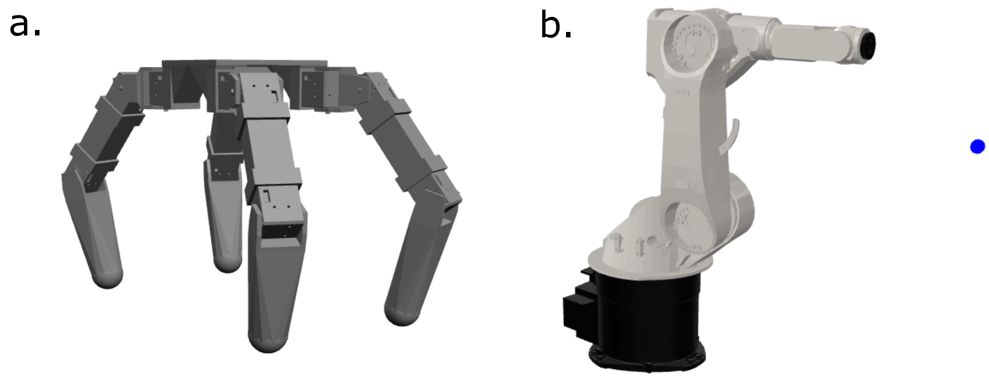


Figure 7.2: We enable casual users to design expressive motions for two very different types of robots – (a) a quadruped, and (b) a robotic arm. We consider periodic walking motions for the quadruped, while for the robotic arm, we consider the task of moving towards a target point (shown in blue). Each of these motions can be performed with different styles and expressions. Our tool enables the users to explore the space of these motions.

To the best of our knowledge, this is the first system that enables casual users, without any domain knowledge of animation or robotics, to design semantically meaningful robotic behaviors. The systems utility is shown with a user-study, which indicated that users were able to create high-quality expressive robot motions. The generalizability of the presented framework is demonstrated by using it for two distinct robotic systems: walking robots, and manipulator arms (Figure 7.2).

7.3 Related work

This work is inspired and builds upon prior work on semantic editing, crowd-powered editing, and robot motion design.

Semantic editing and design space exploration Editing using semantic or context-specific attributes has been explored for many complex design domains such as 3D models [43, 239], images [114, 119, 163], fonts [154], and garments [118]. Each of these approaches extract relevant and human-understandable attributes for their design domain, and learn a mapping between the design parameters and these attributes. With this mapping, they enable intuitive, attribute based editing at design time. We wish to extend this methodology to the domain of robotics. Unlike the domain of 3D models and images, there is no existing large dataset of expressive robot motions. We therefore parameterize and synthesize a wide variety of such motions using a physics-based simulation.

Along with semantic editing, visual design space exploration is another useful approach. Researchers have proposed intuitive low-dimensional control spaces for predictable editing, and design space exploration of complex design problems such as editing material appearance [190], or 3D models [43, 239]. Instead of finding a low-dimensional control space, we expose the current parameter space in a more visual and meaningful manner.

This work builds on Koyama *et al.*, which enables intuitive editing of continuous parameters corresponding to digital content such as images, visually, using a crowd-powered framework [116]. Parameter sliders with heat-map visualizations are used to gently guide the users to a relevant region in the design space. *Geppetto* deals with design spaces that consist of both continuous and discrete parameters and is particularly suited to design spaces represented by low fidelity or noisy data. We leverage mixed-initiative design for scenarios where the available datasets capture the design space in a limited manner, or when the data is relatively noisy. This is achieved by providing relevant guidance in a transparent manner. Specifically, parameter sliders are annotated with curves that indicate both parameter-semantic attribute relationships, and the degrees of uncertainty within those relationships. Finally, unlike most crowd-powered systems, *Geppetto* provide user interface features that enable users to combine their individual preferences with crowds preferences at design time.

Designing expressive robotic motion: Many data-driven, or model based approaches have been explored for motion synthesis. In particular, motion capture and video data have been extensively used for increasing the style and expressiveness of anthropomorphic characters [13, 161, 191]. However, it is unclear how to obtain or use such data for more generic and non-anthropomorphic robots such as robotic arms. A complementary user-driven approach is to animate toy robots, or virtual characters using puppeteering [24, 47, 85, 195]. However, it is hard to pose highly articulated robots or characters so as to create natural looking feasible motions using puppets. Therefore, in spite of being simple and amiable to casual users, most puppeteering based approaches are either limited to very simple characters or robots [24, 195], or they fail to account for physical feasibility [47, 85]. Similar to puppeteering are Programming by Demonstration (PbD) based approaches that enable novices to design robot motions by simple demonstrations [30]. While PbD enables easy creation of natural motions, designing semantically meaningful and expressive motions remain challenging with PbD. Finally, models that encode animation principles [216, 228] have been leveraged to improve expressiveness of robotic systems for enhanced human-robot interaction [174, 206, 209]. Unfortunately, many of these principles are abstract and generic. They are therefore typically used either as add-on primitives for pre-existing motions [206], or as high-level guides for manual design, similar to how animators would use them [174]. The principles do not provide any guidance about synthesizing distinct emotive motions from scratch. Instead, our system enables users to design motions by editing parameterized robot motions in simulation.

Researchers have shown a strong relation between motion parameters and attribution of affect, for robots with different embodiments [92, 183]. In particular, speed and robot pose [92, 111, 210], acceleration, and motion path curvature [25, 111, 183], and motion timing [111, 240] have been found to affect perceptions of motions. We therefore parameterize the walking robots motion using features such as pose, speed, and motion timing. The robot arms motion is parameterized in the task space[1] instead of the joint space, inspired by how abstract trajectories could convey different emotions [25]. The systems semantically-guided parameter editing ap-

[1]Task space is the lower-dimensional subspace of motion directly relevant for the task.

proach complements recent research on optimization-guided and keyframe-based motion editing for animated characters [115].

Crowd-sourcing in robotics and design: Crowdsourcing is used to understand the coupled effect of various motion parameters on the overall emotional perception. Crowdsourcing enables the use of human expertise for tasks that are complex for computers, and has been widely used for a variety of tasks ranging from labeling, to gathering common-sense knowledge and opinions [238]. In robotics, crowdsourcing has been used to enable robots to recognize objects or actions [73, 204], as well as for robot control [46, 158]. Our work is closer to the research on understanding visual perception, and enabling better design through crowdsourcing [68, 116, 149, 239]. We build on these approaches and use a crowd-powered pairwise comparison approach for evaluating motions. The crowdsourcing pipeline is customized to deal with the greater difficulty and cost associated with evaluating our motion designs, which results from the length of the animation needing to be judged, and uncertainty due to the high subjectivity of the task. Notably, we use a modified Swiss-system tournament [56] approach with an added elimination step, and use *TrueSkill* [143] to efficiently compute the perceptual quality scores for the synthesized motions.

7.4 Survey of current design approaches

To understand the current challenges of robotic motion design, a survey of experts who design expressive behaviors for applications ranging from art and entertainment to Human-Robot Interaction (HRI) was conducted.

7.4.1 Survey instrument

HRI and robotics researchers, artists, and animators participated in a survey. In addition to background questions, the survey consisted of 5-point Likert-scale and free-form questions. All Likert scales used the following anchors: 5 = extremely, 1 = not at all. The questions elicited information about the types of behaviors they designed, how and why they designed them, as well as the time taken and tools used in design. We also asked their opinion about the tools they used, in terms of ease of use, learnability, and suitability for various robot behavior design tasks.

7.4.2 Responses

Eight experts (4 HRI researchers, 4 artists/animators) with design experience ranging from 0.5 years 27 years (average 11.3 years) participated in the survey. The experience of these experts covered a diverse range of contexts such as 2D/3D character behavior design, industrial and social robot design, and kinetic art sculpture.

Despite the diversity in applications, a common motivation behind designing expressive behaviors was to improve the communication, involvement, and interaction of the technology they were developing (e.g., P3: *"I want my robots to be more human-readable."*, P4: *"[I want] to turn viewers into involved, emotionally invested participants."*). In response to why would they design expressive robot behaviors, experts provided further insights (P3: *"Being expressive is part of being communicative, which is critical for functional and fluent human-robot interactions. Emotion can be useful for communicating a robot's goal."*, P7: *"I see a robot's bodily motion as a lower-level means of broadcasting complex information to surrounding people."*), to tell a story and develop relationships with users (P6: *"Many engineering 'stories' do not show realistic motion which allows the viewer to dismiss the concepts."*, P2: *"To develop relationships with users through tangible actions."*).

A common theme highlighting the effort required to design behaviors also emerged. Experts who designed short-length behaviors of less than a minute (50% of our participants) reported a design time of greater than one hour. Likewise, experts who designed longer behaviors (lasting multiple minutes or hours) spent several days and sometimes several weeks for their design.

Another common theme was the lack of tools for designing robotic behaviors. Researchers as well as artists emphasized that the existing tools were not well suited for robot behavior design (5-pt Likert score, where 5 = extremely unsuitable: M = 4.0, SD = 0.92). Experts typically relied on animation tools or ended up developing their own software. Several experts reported on the difficulty of obtaining robot simulation models (P3: *"Putting kinematic robot models into simulation takes a long time."*), pre-visualization of robot capabilities (P4: *"Pre-visualization can be quite difficult. One needs to have the actual robot working in a realistic setting in order to test it."*), and manual behavior editing (P2: *"Manually creating gestures through motor positions is tedious, unintuitive."*, P5: *"My chief problem is the lack of software tools for authoring dynamic performances with shared autonomy; I end up having to write too much software."*). Experts further reported that the tools they used were difficult to learn and use (5-pt Likert score, where 5 = extremely difficult: M = 4.12, SD = 0.64). They also emphasized the consequential challenges faced by novices in such design applications (e.g., P1: *"Having to learn lots of different, changing software and then figuring out how to connect them is difficult for people just starting out."*, P3: *"The toolchain is complicated and tedious."*).

Overall, the survey validated the need for improved systems for the design of expressive robot behaviors. It revealed interesting use cases and current challenges, pointing to a need for new, more intuitive and efficient tools.

7.5 *Geppetto* – Semantic editing for robotics: Overview

Inspired by the challenges and desires found through the survey and in the literature, *Geppetto* enables robot motion design with the help of a physics-based simulation. Parameters affecting the robot motion are presented to the user, and the system aims to reduce the domain knowledge required when modifying these parameters to create desirable motions. In particular, the system supports editing based on semantic user intent, such as designing a "happy-looking" robot. The system currently supports such semantic design for six basic emotion categories happy, sad, angry, scared, surprised, and disgusted, , though it could be applied to other semantic aspects beyond emotions (e.g., expressing that a robot is busy, awaiting instruction, or friendly). The emotion categories are derived from Ekmans model of emotions [71], though we anticipate this approach can extend to other emotion models.

7.5.1 Interface design

The UI (Figure 7.3)) consists of three main elements a 3D preview window, motion gallery, and guided-editing pane. The 3D preview window renders the main robot and animates its simulated motion in real-time. The sliders in the editing pane allow users to specify the robots motion parameters. The motion gallery displays various expressive motions of different styles for a user-specified emotion category. This gallery is populated using the emotion specific top-ranking motions from our dataset, obtained using sampling and crowdsourcing analysis.

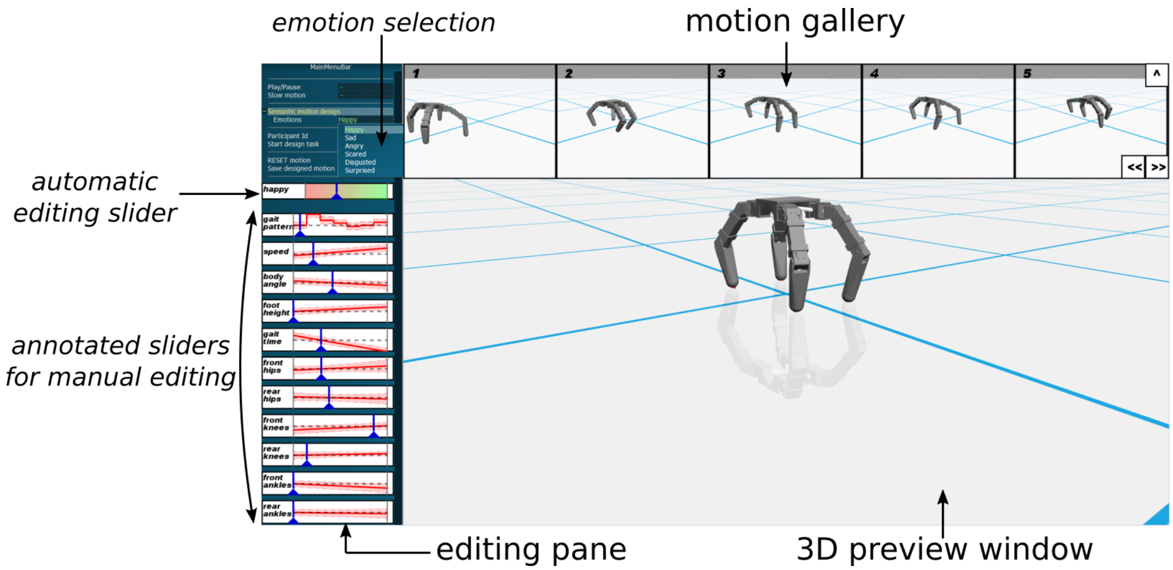


Figure 7.3: User interface overview – The 3D preview window renders the robot's motion. The gallery and annotated sliders provide semantically relevant information at design time.

7.5.2 Design process overview

The design process for creating an emotive robot behavior using *Geppetto* begins with users a user selecting a desired emotional expression (happy, sad, etc.) for the behavior from the editing pane. They can either start with a neutral default motion, or they can take advantage of the example motions in the gallery by browsing through the samples to get a sense of different motion alternatives, and then load a preferred example for further editing. Such an approach of using example-based inspiration has been found to support creativity in designers [121]. Gallery-based initialization is especially useful for novices who may not know what an expressive robot motion looks like. Once a motion is initialized, users can edit motion's expressiveness as desired using two guided editing modes – manual, and automatic. Each mode focuses on two different, and critical requirements of casual users – fast prototyping customization, and learning. The automatic mode enables users to quickly customize the robot's motion without worrying about low-level parameter editing. The manual mode, on the other hand, exposes users to parameter level editing such that they develop an inherent understanding of which parameters create the necessary expressiveness, as well as how to edit them. With every user edit, the simulation updates the robots motion in the preview to reflect the corresponding change.

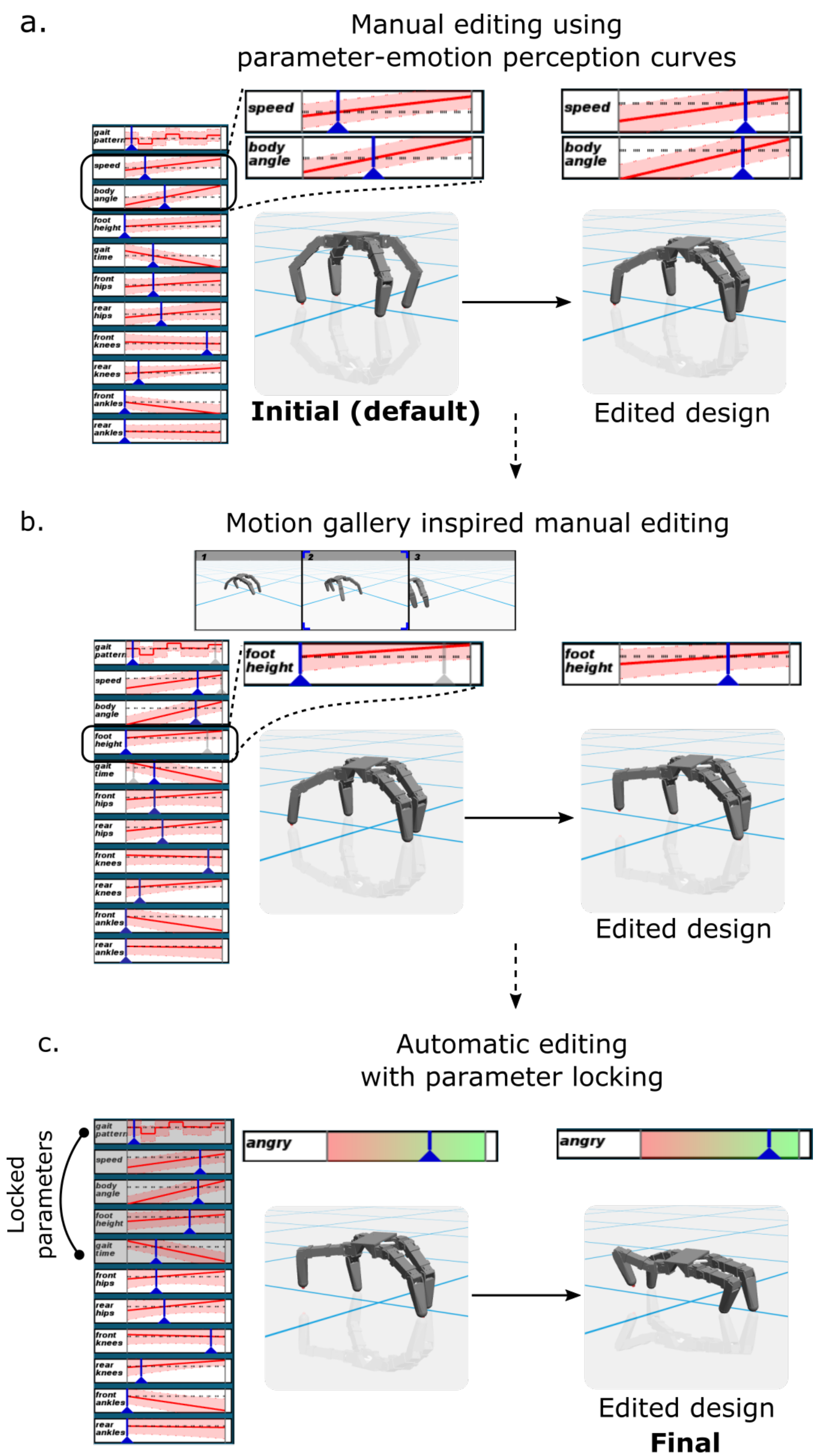


Figure 7.4: An example workflow of designing an angry robot.

To design an angry robot, the user starts with the default motion, and proceeds to manually edit it using parameter sliders (Figure 7.4). To understand which parameters to change and how to change them, the user takes advantage of *parameter-emotion perception relationship curves* visualized on each slider (Figure 7.5b). Based on these curves, the user increases the speed and tilts the robots torso downwards to make it look angrier (Figure 7.4a). The user then leverages the example motions in the gallery for further editing. The user hovers over the preferred gallery motion to understand which parameters created it, with the help of *parameter comparison cursors* (Figure 7.5c). Inspired by the feet stomping of second gallery example, the user edits the current motions feet height to achieve the same (Figure 7.4b). Finally, the user can also explore angrier motions automatically by dragging the *automatic editing* slider. In response, the system changes multiple parameters simultaneously to increase the motions expressiveness. To further explore preferred motions, e.g., angrier motions with similar speed, feet stomping, and torso tilt, the user activates *locking* of these parameters, before dragging the automatic editing slider (Figure 7.4c). The system now auto- updates multiple parameters except the locked ones, to change the motions expressiveness.

7.5.3 Interface editing features

As highlighted by the workflow, manual editing leads to user understanding of parameters, and is enabled by parameter-emotion perception relationship curves and parameter comparison cursors. On the other hand, automatic editing allows quick updates of the users motion design, based on users high-level intent of increasing or decreasing the intensity of robots emotional expression. It is powered by automatic slider and parameter locking feature.

Parameter-emotion perception relationship curves: These curves, which are accompanied with each slider, show the effect of changing the sliders parameter on the robots resultant emotional expression. Since these relationships are extracted from subjective crowd-sourced data, the UI also shows the systems confidence in these relationships visualized as non-linear error bands around the predicted score (see Figure 7.5b). This allows users to determine the extent to which they may want to follow the curves during parameter editing. The inclusion of these error bands brings transparency to the mixed-initiative editing process, allowing the user to better collaborate with the system to achieve their goals.

Parameter comparison cursors with motion gallery: Different sets of parameter values result in widely diverse motions; each corresponding to a different style or intensity of an emotion. To enable users in understanding how different parameter values result in motion diversity, we leverage the diverse examples in the motion gallery. Users can visualize parameter values corresponding to any motion in the gallery on the sliders, by hovering over that motion (Figure 7.4b). This enables the users to make parameter-level comparisons between the motions in the gallery and their design, and copy the preferred individual parameter values.

Automatic slider: By dragging the automatic slider, users can update multiple parameters simultaneously, rather than adjusting them individually. When the position of the slider is changed, the system automatically modifies multiple parameters to achieve the corresponding change in the robots emotional expression. This feature can be used in combination with parameter locking

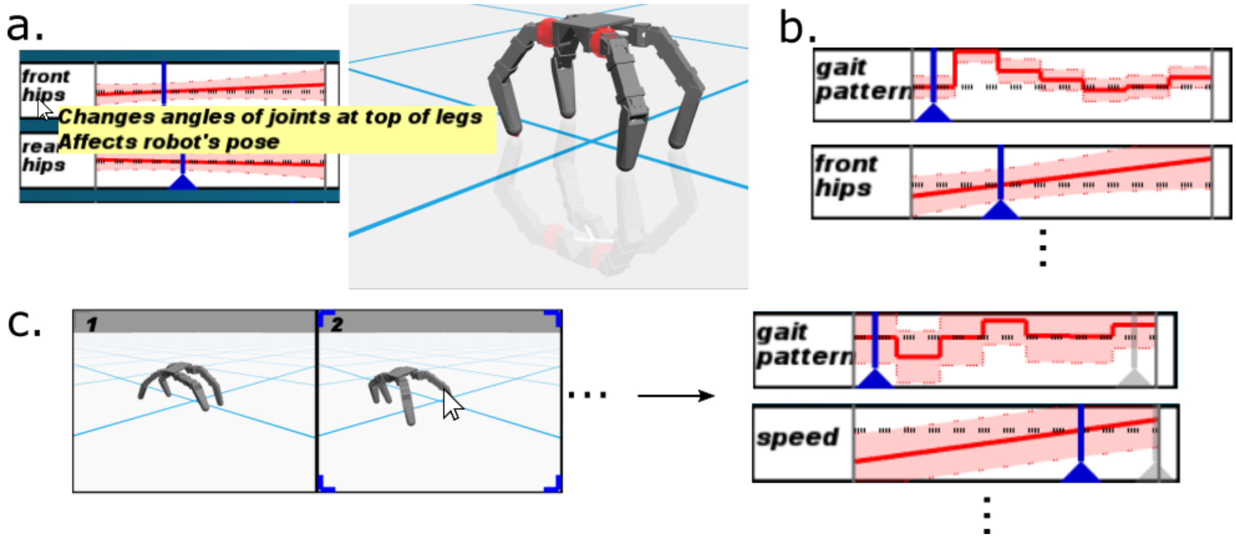


Figure 7.5: UI features – (a) Parameter information is displayed as tooltips, and highlighted directly on the robot. (b) Parameter-emotion perception curve (in red) is visualized with an uncertainty band (shaded red) on each slider. The dotted line corresponds to current motions estimated emotional perception. (c) When a user hovers over a gallery motion, the gallery motion's parameter values are highlighted on the sliders (in light gray) alongside the current motions parameters (shown in blue).

(explained next) to achieve the desired behavior.

Parameter locking: As the automatic slider updates multiple parameters at a time, changing the automatic slider by a small amount may drastically modify the resulting motion. As a result, the nuanced features of the robots motion achieved by a users earlier edits may be lost when using the automatic slider. To preserve the desirable features of their current motion during automatic editing, users can lock parameters. For instance, in the example scenario of angry motion design, the user may want to maintain the speed, torso tilt, and feet stomping achieved through manual editing, while exploring better limb poses. To achieve this, the user can lock all but the pose parameters through the editing panel, and then use the automatic slider to obtain an angrier robot motion with similar speed, feet stomping, and torso tilt (Figure 7.4c). Note that this is much quicker than the alternative of manually editing 6 pose parameters. Parameter locking thus allows users to combine their design preferences with crowd-powered guidance during automatic editing of designs.

The gallery motions are also updated to show more relevant examples after parameters are locked. To update the gallery, we sort the motions in the dataset based on the similarity of parameters to the values locked by the user, and the quality of emotion expression. This gives users alternate motions satisfying the preferences indicated by the locked features.

Our system thus supports various workflows for motion editing. An optimal workflow could combine both manual and automatic editing as needed. Our video[2] shows such workflows in action.

[2] An overview video about *Geppetto* is available here – `https://youtu.be/DXbnwodJ2Ks`

7.6 Robotic platforms and their motion synthesis

7.6.1 Robotic platforms

We use two diverse robotic platforms as testbed for our design system. The robotic arm is an industrial grade, six degrees of freedom (DOF) KUKA arm [117]. Similar robotic arms have been used for applications requiring expressive motions such as collaborative building [222], and interactive art [214], making it an ideal choice as our testbed. Another class of robotic systems – walking robots have also been used in interactive settings such as in animatronics [64], and consumer products [198]. As a representative from this class of robotic systems, we design a small-sized custom quadrupedal robot with three DOF per leg. Figure 7.2 illustrates these robots.

7.6.2 Motion synthesis for quadruped using physics simulation

Physics simulation and motion optimization framework

For generating motion dataset for the walking robot (in Fig. 7.2(a)), we use our physics simulation and motion optimization framework described in Sec. 6.5.5. Briefly, the quadruped robot's motion $\mathbf{m}$ consisting of periodic coordinated limb movements that repeat in cycles of time T (henceforth referred as gait cycle), is defined by time-indexed sequence of vectors P_i, as $\mathbf{m} = (P_1, \ldots, P_T)$. P_i represents joint and limb end-effector positions at i^{th} time-step (robot pose), as well as information about whether a limb is on the ground or moving in the air at that instant. We obtain such a robot motion $\mathbf{m}$ by solving a constrained trajectory optimization problem described below (see Sec. 6.5.5 for more details):

$$\begin{aligned} &\min_{\mathbf{m}} \sum_i E_i(\mathbf{m}) \\ &s.t. \quad \mathbf{C}(\mathbf{m}) = 0\,, \end{aligned} \tag{7.1}$$

where $E_i(\mathbf{m})$ represent quadratic penalty terms for achieving desired speed and motion style, while $\mathbf{C}(\mathbf{m}) = 0$ are various constraints necessary for a valid motion. These constraints include kinematic and limb collision avoidance constraints that ensure motion's physical validity, friction constraints to avoid foot slipping, and joint and torque limits. The goal of motion optimization is thus to make the robot move forward with a desired speed and motion style, while satisfying various physics-based and feasibility constraints. We use gradient-based solvers to find such a valid and desired motion [134].

Parameterization

Based on prior research, we expose 11 parameters affecting the robots motion style for generating a dataset of diverse motions [92, 111, 210]. In particular, various motion styles can be created by using different robot poses and gait patterns (e.g., galloping, trotting, walking etc.). Gait patterns are defined for a gait cycle, and are characterized by the order of limb movements, relative phase of limb swing, and stance [50]. Pose is defined using relative joint angles of

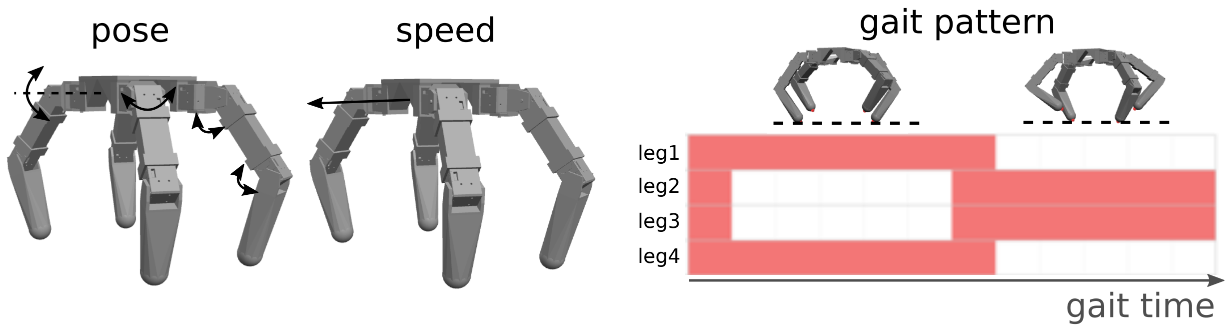


Figure 7.6: The quadruped's motion is parameterized using joint poses, walking speed, foot height, gait time, and gait patterns. An example gait pattern graph is shown here with corresponding robot limb phases (red bar corresponds to a leg in swing).

Table 7.1: Walking robot motion parameters

Parameter name	Parameter type	Parameter dimension
Gait pattern	Discrete	1
Gait time	Continuous	1
Speed	Continuous	1
Foot height	Continuous	1
Robot pose	Continuous	7

robot's limbs, as well as its global torso orientation angle defined relative to the ground plane. Pose consists of 7 angular values – torso angle, front and rear hip angles, front and rear knee angles, and front and rear ankle angles (Figure 6a). Apart from speed (1DOF), pose (7DOF), gait time (1DOF) and pattern (1DOF), we also parameterize foot height (1DOF) to create the effect of feet "stomping". The gait pattern corresponding to each gait style is discretely encoded using a graph (see Figure 7.6), while all other parameters are continuous. More gait patterns used for our dataset generation can be seen in Figure 7.7. Table 7.1 summarizes these parameters.

7.6.3 Motion synthesis for robotic arm using Boids flocking simulation

Expressiveness of robotic arms moving towards a goal can be affected by many features, such as the curvature of its path [25], the variability of its speed [240], and path smoothness. Instead of directly prescribing the robot arms path and speed, we empirically choose to use a Boids simulation to drive its motion similar to the approach used in Mimus [80].

The Boids framework uses virtual agents called boids, and a set of simple interactions between them to create smooth, complex and natural emergent behaviors [3] [172]. We define a flock of number of boids in the 3D task space, and then use the resultant average path of the flock as the target path for the robotic arm's end-effector, to be achieved through Inverse Kinematics (IK) (see Figure 7.8). The resultant motion and the path of the flock depend upon the interaction

[3]Emergence phenomenon is where larger entities arise through interactions among smaller or simpler entities such that the larger entities exhibit properties the smaller/simpler entities do not exhibit (Courtesy: Wikipedia [232]).

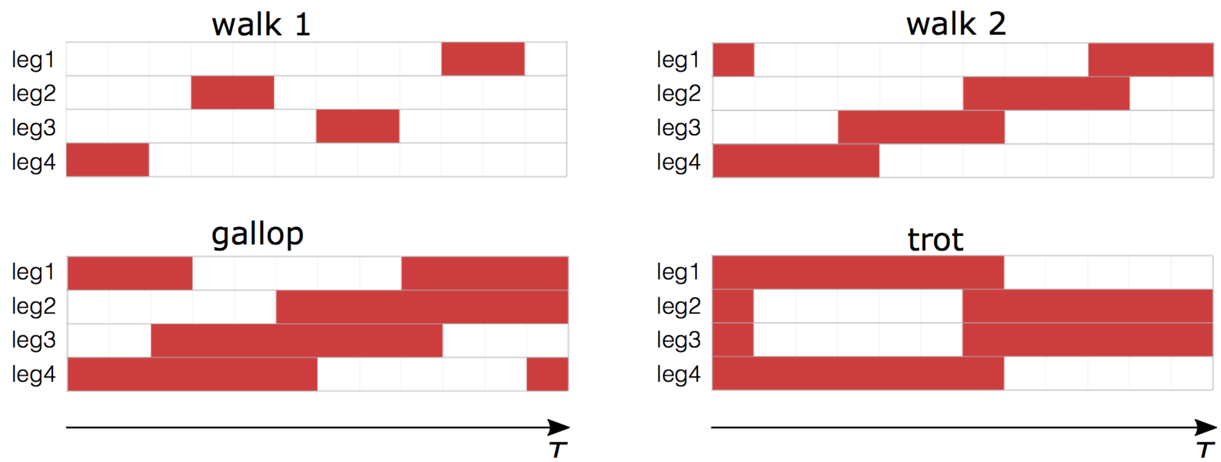


Figure 7.7: Some of the gait patterns used to create motion variations while generating walking motion dataset for the quadruped robot are shown using their gait graphs. Each is described over a single gait cycle of time T after which the motion repeats. Gait pattern describe the order, and timing of each leg in swing and stance phases (swing phase is highlighted in red). Trotting, galloping, and two kinds of walking gaits are shown.

rules that decide each boids movement, as a reaction to its nearby flock-mates within a small neighborhood around itself. We next elaborate on the framework, various interaction rules used in our implementation, and corresponding parameters.

Boids simulation framework

Each boid b_i is represented by a position $\vec{p}^{\,i}$, and velocity $\vec{v}^{\,i}$ that are typically initialized randomly in the beginning of simulation. As simulation progresses, each boid is subjected to a steering force $\vec{F}^{i}$ that updates its velocity and position (see eq. 7.2). For each b_i,

$$\begin{aligned} \vec{v}^{\,i}_t &= \vec{v}^{\,i}_{t-1} + \vec{F}^{i}_t \\ \vec{p}^{\,i}_t &= \vec{p}^{\,i}_{t-1} + \vec{v}^{\,i}_t\,, \end{aligned} \tag{7.2}$$

where t denotes the simulation time. Steering force represents a boid's reaction to its flock-mates within a certain small neighborhood around itself, based on the interaction rules. The *neighborhood* is thus a region within which flock-mates influence a boid's movement, and is characterized by a distance around each boid. Flock-mates beyond this distance are ignored. Emergence of different flocking behaviors thus depends on the neighborhood, and the *interaction rules*. Each rule produces a unique steering force on every boid. The net steering force $\vec{F}^{i}$ on each boid is defined as a weighted sum of steering forces from all rules (we mathematically define it later in eq. 7.8). A weight parameter corresponding to each rule determines the strength of its steering force. Using different weight combinations, one can create a wide variety of flocking behavior variations. Since each rule is applicable based on flock-mates within a neighborhood n around a boid, we first mathematically define flock-mates $\mathcal{F}^{i}_n$ for each boid b_i as follows:

$$\mathcal{F}_n^i = \left\{ \bigcup_{j=1}^{m} b_j \middle| \|\vec{p}^j - \vec{p}^i\| <= n \right\} , \tag{7.3}$$

where n is the neighborhood, m is the number of boids in the flock, $\vec{p}^i$ represents the position of a boid b_i, and $\|\cdot\|$ denotes the Euclidean norm. $\mathcal{F}_n^i$ is thus the set of all boids b_j within n distance of a boid b_i.

We use five interaction rules – three rules from the basic boids model [171]: separation, alignment, and cohesion; and two other custom rules: goal-seeking, and exploration (see Figure 7.8).

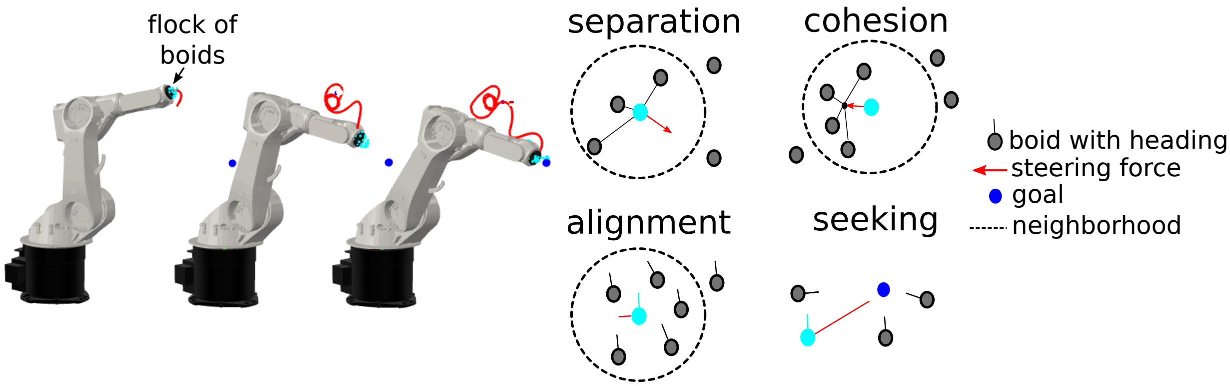


Figure 7.8: The robotic arm is driven by a Boids flocking simulation. Each boid reacts to its neighboring boids based on simple interaction rules such as separation, cohesion, alignment, seeking etc. to move in the space. The resultant emergent path of the boids flock is followed by the arm. The boids within the dotted circle, corresponding to a boid's neighborhood n, are called its neighboring flock-mates ($\mathcal{F}_n^i$). Steering force created by each rule is shown by a red arrow.

Separation rule: steers the boid to avoid crowding local flock-mates (see Fig. 7.8). The steering force, and weight parameter for separation rule are represented by $\vec{F}_s$, and w_s respectively. For each boid b_i, $\vec{F}_s^i$ is proportional to crowding $\vec{c}^i$, computed as follows:

$$\begin{aligned} \vec{c}^i &= \sum_{b_j \in \mathcal{F}_n^i} \vec{p}^i - \vec{p}^j \\ \vec{F}_s^i &= \frac{\vec{c}_i}{\mid \mathcal{F}_n^i \mid} - \vec{v}^i , \end{aligned} \tag{7.4}$$

where $\vec{p}^i$ and $\vec{v}^i$ represent the position and velocity of boid b_i respectively, $\mathcal{F}_n^i$ is the set containing flock-mates of b_i within a neighborhood n (defined by eq. 7.3), and $\mid \mathcal{F}_n^i \mid$ denotes its cardinality[4].

Alignment rule: steers the boid towards the average heading $\vec{h}$ of local flock-mates (see Fig. 7.8). The steering force, and weight parameter for alignment rule are represented by $\vec{F}_a$, and w_a respectively. For each boid b_i, the average flock heading $\vec{h}^i$, and $\vec{F}_a^i$ are defined as:

[4]Cardinality represents the number of elements in a set (Courtesy: Wikipedia [232]).

$$\vec{h}^i = \sum_{b_j \in \mathcal{F}_n^i} \vec{v}^j$$
$$\vec{F}_a^i = \frac{\vec{h}_i}{\mid \mathcal{F}_n^i \mid} - \vec{v}^i\,, \tag{7.5}$$

where $\vec{v}^i$ represent the velocity of boid b_i, and $\mathcal{F}_n^i$ is defined by eq. 7.3.

Cohesion rule: steers the boid to move toward the average position $\vec{a}$ of local flock-mates (see Fig. 7.8). The steering force, and weight parameter for alignment rule are represented by $\vec{F}_c$, and w_c respectively. For each boid b_i, the average flock position $\vec{a}^i$, and $\vec{F}_c^i$ are defined as:

$$\vec{a}^i = \sum_{b_j \in \mathcal{F}_n^i} \vec{p}^j$$
$$\vec{F}_c^i = \frac{\vec{a}_i}{\mid \mathcal{F}_n^i \mid} - \vec{v}^i\,, \tag{7.6}$$

where variables are defined as in eq. 7.5, and 7.4. Unlike separation, cohesion, and alignment rules, seeking and exploring do not require a set of neighboring flock-mates ($\mathcal{F}_n$).

Seek rule: steers the boid to move towards a pre-defined goal $\vec{p}_g$ in space (for instance, the blue goal point in Fig. 7.2(b)). We add this to our framework to ensure that the emergent motion of the flock eventually moves towards the goal for our application. This goal could also be defined by the user. Steering force for the goal-seeking behavior $\vec{F}_g$ is qualitatively equivalent to that of a virtual spring with unit stiffness, and is thus proportional to the distance of the boid from the goal. It is defined as follows:

$$\vec{v}_g^i = \frac{\vec{p}_g - \vec{p}^i}{\|\vec{p}_g - \vec{p}^i\|}$$
$$\vec{F}_g^i = \vec{v}_g^i - \vec{v}^i\,, \tag{7.7}$$

where $\vec{v}_g^i$ is the desired velocity for boid b_i to reach the goal $\vec{p}_g$, and $\vec{v}^i$ is its current velocity. Heuristics to dampen out $\vec{F}_g^i$ as the boid b_i nears the goal can be used to prevent b_i from oscillating around the goal. The rule's weight parameter is denoted as w_g.

Explore rule: steers the boid towards a random goal intermittently to enforce randomness in the flock behavior. This ensures interesting non-straight paths for the robot that eventually converge at the desired goal in our case. Apart from a weight parameter w_e, this rule also has two additional parameters – exploration radius r_e, and exploration frequency f_e. These parameters are used to decide where and when to set a new random goal $\vec{p}_e$ for the flock. $\vec{p}_e$ is sampled at random from within a sphere with radius r_e centered at the average position, at every f_e^{th} time-step of the simulation. Explore rule then steers each boid to move towards $\vec{p}_e$ using seeking behavior.

This rule is thus an extension of the seeking rule for random goals along the flocks path. We denote the corresponding steering force as $\overrightarrow{F}_e$ for clarity, even though it is calculated in the same manner as $\overrightarrow{F}_g$ in eq. 7.7.

Using the weight parameters, and the steering forces defined for these rules, we can now define the net force $\overrightarrow{F}^i$ on each boid b_i as follows:

$$\overrightarrow{F}^i = w_s\overrightarrow{F}^i_s + w_a\overrightarrow{F}^i_a + w_c\overrightarrow{F}^i_c + w_g\overrightarrow{F}^i_g + w_e\overrightarrow{F}^i_e \,. \tag{7.8}$$

Parameterization

We can now summarize the parameters for the Boids simulation. It is parameterized by interaction rules specific parameters such as the weights w_s, w_a, w_c, w_g, w_e corresponding to our five interaction rules (separation, alignment, cohesion, goal-seeking, and exploration), and the neighborhood n. A maximum speed v_{max} is also defined for the overall motion of the boids. v_{max} serves two purposes. First, it enables us to create flock motions (and thereby robot arm motions) with different overall speeds ranging from very slow to fast motions. Secondly, it ensures that the simulation doesn't become unstable by applying an unreasonably high steering force on the boids. Other additional parameters include – parameters for exploration rule (exploration radius r_e and frequency f_e), and those corresponding to flock initialization procedure (explained later). These parameters are also summarized in Table 7.2. Together, they form at 11-dimensional parameter space.

Table 7.2: Boids simulation parameters

Parameter name	Parameter type
Neighborhood n	Continuous
Separation rule weight w_s	Continuous
Alignment rule weight w_a	Continuous
Cohesion rule weight w_c	Continuous
Seek rule weight w_g	Continuous
Explore rule weight w_e	Continuous
Explore radius r_e	Continuous
Explore frequency f_e	Continuous
Maximum flock speed v_{max}	Continuous
Initial boid velocity direction $\overrightarrow{v}_{init}$	Discrete
Initialization time t_{init}	Continuous

Apart from integrating two custom interaction rules (goal-seeking and exploration), we extend the basic Boids simulation framework in two more ways. First, we define an initialization procedure for the boids to increase the diversity of flock behavior. Secondly, we develop a heuristics to smoothly blend the exploration rule with other interaction rules of the basic boids model.

Boid flock initialization procedure: This procedure was added to increase the variations of generated paths such as paths that go backwards or sideways before converging on the desired goal. We use two parameters for this purpose – an initialization time t_{init}, and an initialization direction $\vec{v}_{init}$. Each boid's velocity is initialized to be in $\vec{v}_{init}$ direction before the simulation. To create paths that go backwards, sideways etc., $\vec{v}_{init}$ is chosen from a set of six direction basis corresponding to *+X, -X, +Y, -Y, +Z, -Z* axis in the task space. The initialization time t_{init}, which is typically less than the overall motion time, represents a buffer time in the beginning of simulation when the flocking behavior is dominated in the $\vec{v}_{init}$ direction. This is to ensure that the initialization procedure effects the flock's behavior, and is achieved by gradual ramping of certain interaction rule weights.

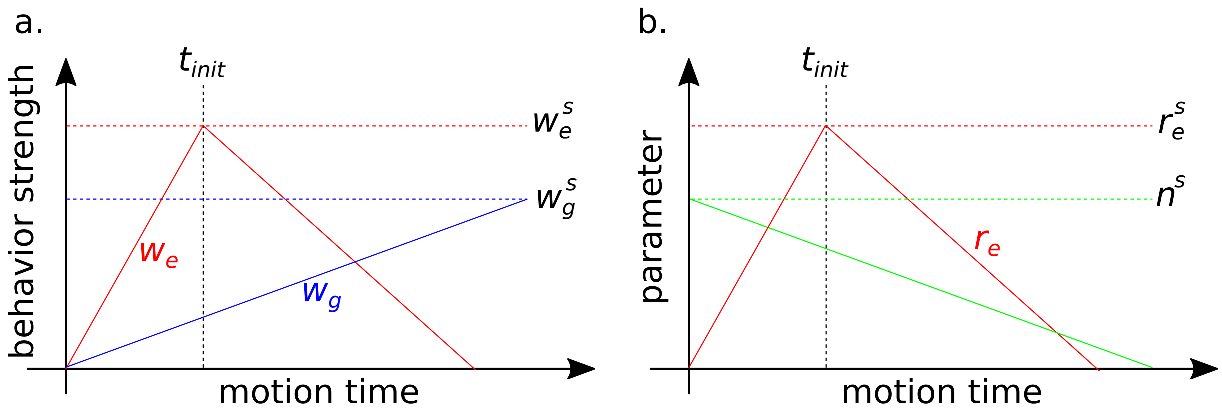


Figure 7.9: Ramping and damping of various boids parameters over the course of the motion – (a) Strength of various behaviors is controlled by ramping and damping their corresponding weights from their sampled values represented by ^{s} superscript. Explore behavior parameters (weight w_e, and explore radius r_e) are color-coded with red, and those of seeking behavior (weight w_g) are color-coded with blue. (b) We also ramp and damp other parameters such as explore radius r_e, and the boid neighborhood n. t_{init} is the initialization time.

Ramping-damping strategy for rule blending: Seeking behavior that aims to steer the boids towards a goal contradicts the other flocking behaviors that typically steer the boids locally, towards more random movements. To prevent these opposing behaviors from fighting against each other, which may create jerky flock movements, we adopt a ramping-damping strategy. We ramp up some parameters gradually over the course of the motion, while we damp down some parameters as the flock nears the goal. In particular, we linearly ramp up the seeking behavior weight w_g, and damp explore behavior parameters (weight w_e, and explore radius r_e), as well as boid neighborhood n. Damping of neighborhood ensures that boids converge at the goal without locally reacting to each other due to flocking behaviors such as separation and alignment. We use an initial buffer time t_{init} to ramp up the explore behavior parameters before damping damp down. This is to ensure that explore behavior doesn't interfere with the velocity initialization, and that the flock moves in the direction of initial velocity $\vec{v}_{init}$ in the beginning. Figure 7.9 outlines our ramping-damping strategy.

7.7 Semantic mapping framework

The semantic information about the robot motions is obtained through our mapping framework that relates the robots motion parameter space to emotional expression space. Our framework leverages the simulation to generate a dataset of diverse motions, evaluates the emotional expression of the dataset motions using crowdsourcing, and then uses regression to obtain the mapping between motion parameters and emotional expression (Figure 7.1).

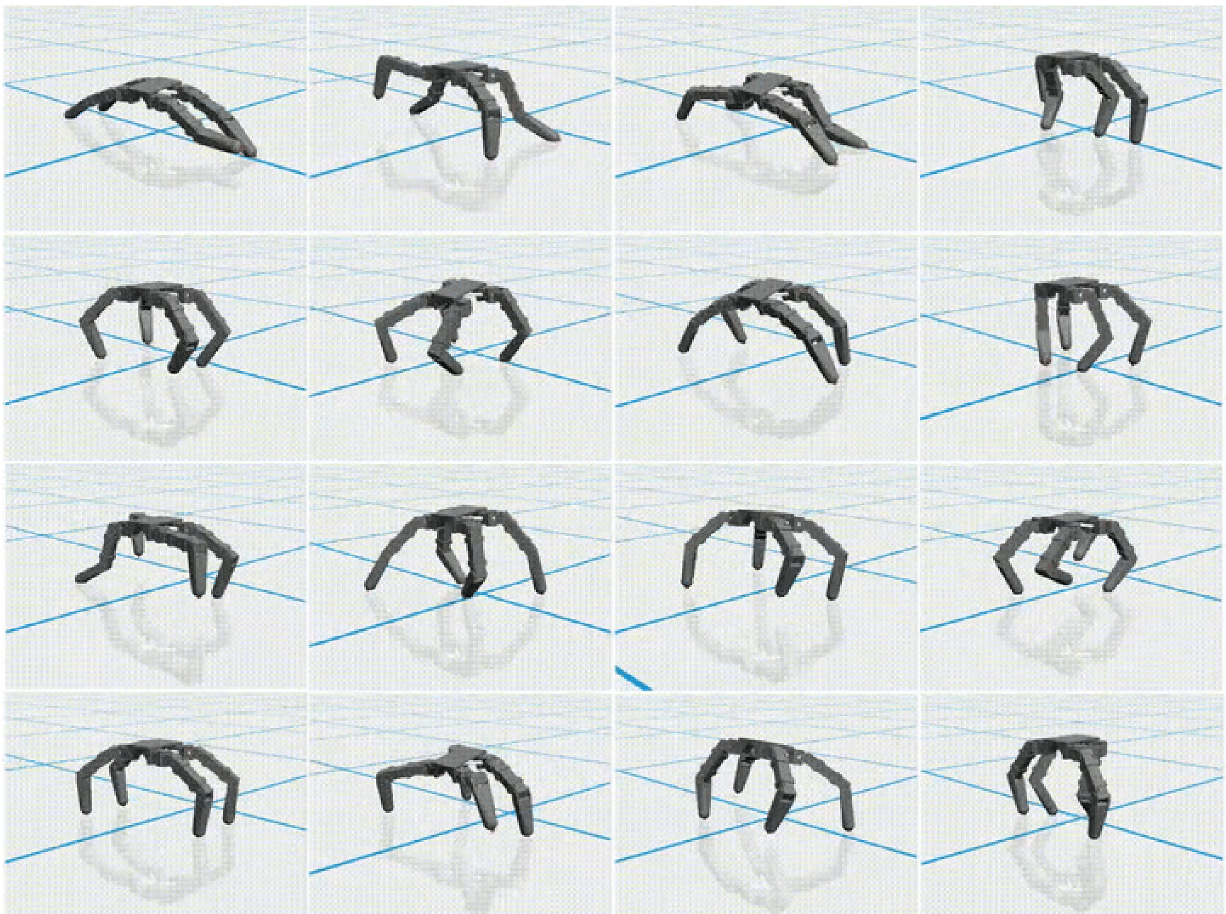


Figure 7.10: Motion samples from our quadruped walking dataset are shown here. Our motion parameters enable us to create various motion styles.

7.7.1 Motion dataset generation

We generate a dataset of diverse motions for the quadruped and the robotic arm using sampling of motion parameters. The sampling process captures the design space of possible motion styles that can be created by changing various motion parameters. We empirically choose a sampling range for all the continuous variables to generate sufficient motion variations while ensuring physical feasibility. The discrete parameters such as gait pattern for the quadruped, and initial direction of boids motion for the robotic arm are uniformly sampled from a fixed set of possible values. For each sampled motion parameter set, we record an animation of the corresponding robot motion for crowdsourcing evaluation. For the quadruped, 2,000 motion parameter sets were sampled, resulting in 2,000 unique motions. 1,230 motions were physically infeasible due to collisions or instability, resulting in 670 motions for the final dataset. Similarly, 1,000 motions were sampled for the robotic arm, all of which were physically feasible and retained. Figures 7.10 and 7.11 show some of the motions from our dataset that we generated in this manner for quadruped and robotic arm respectively.

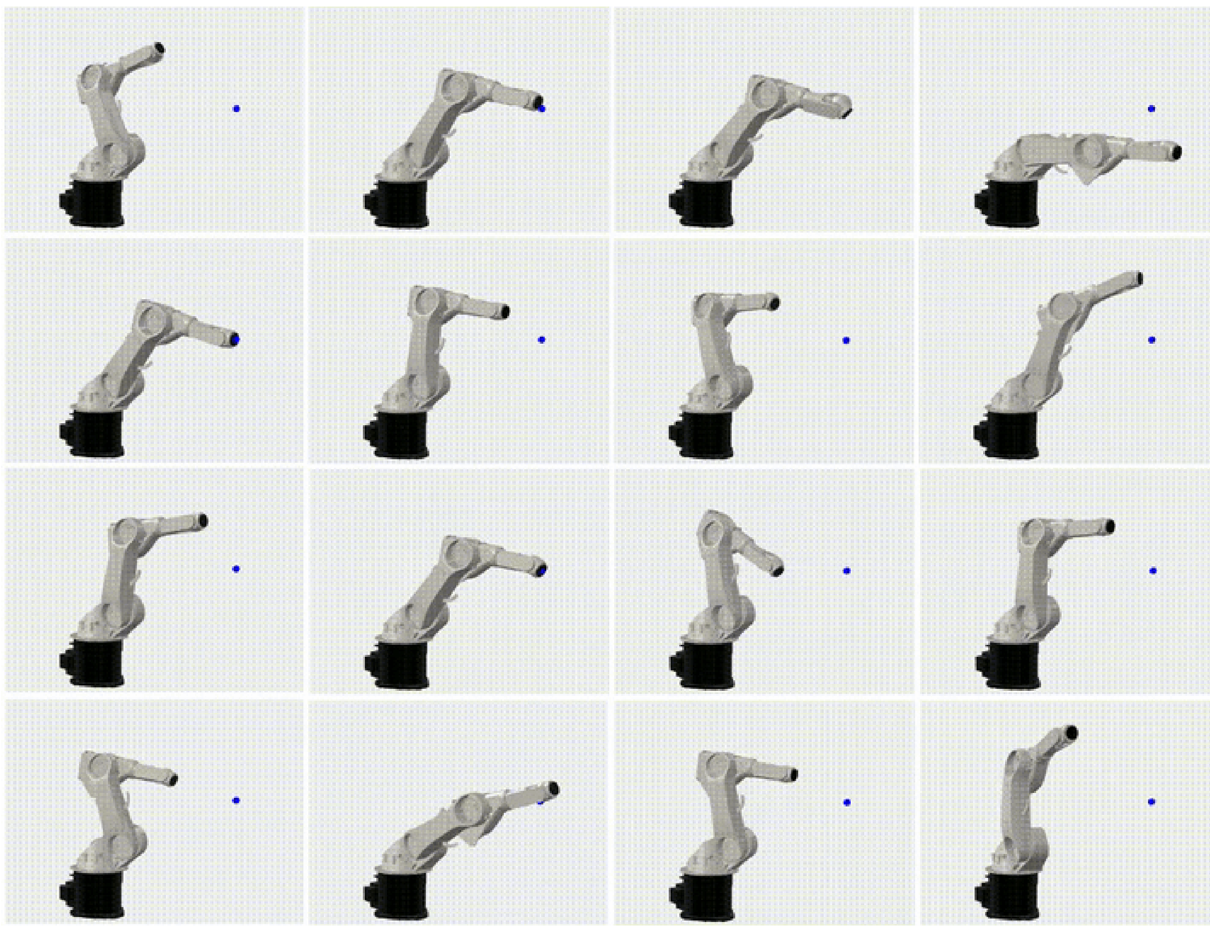


Figure 7.11: Motion samples from our robotic arm behaviors dataset are shown here.

7.7.2 Crowdsourcing evaluation of perceived emotion

By crowdsourcing emotion perception, the system can give a relative scoring to each motion, per emotion, such that a higher score reflects a better expression of an emotion.

While there is often consensus about the particular emotion that is expressed by a motion, the degree of expressiveness is highly subjective and its perception varies between individuals. Given this, we model the score as a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ with mean μ, and uncertainty σ. To compute the score, we create a modified Swiss-system style tournament [56] where each motion sample in the dataset is treated as a competitor, and competes with others to obtain the highest score per emotion category. We use the TrueSkill rating system [143] to convert the results of the tournament into Gaussian score estimates for individual samples.

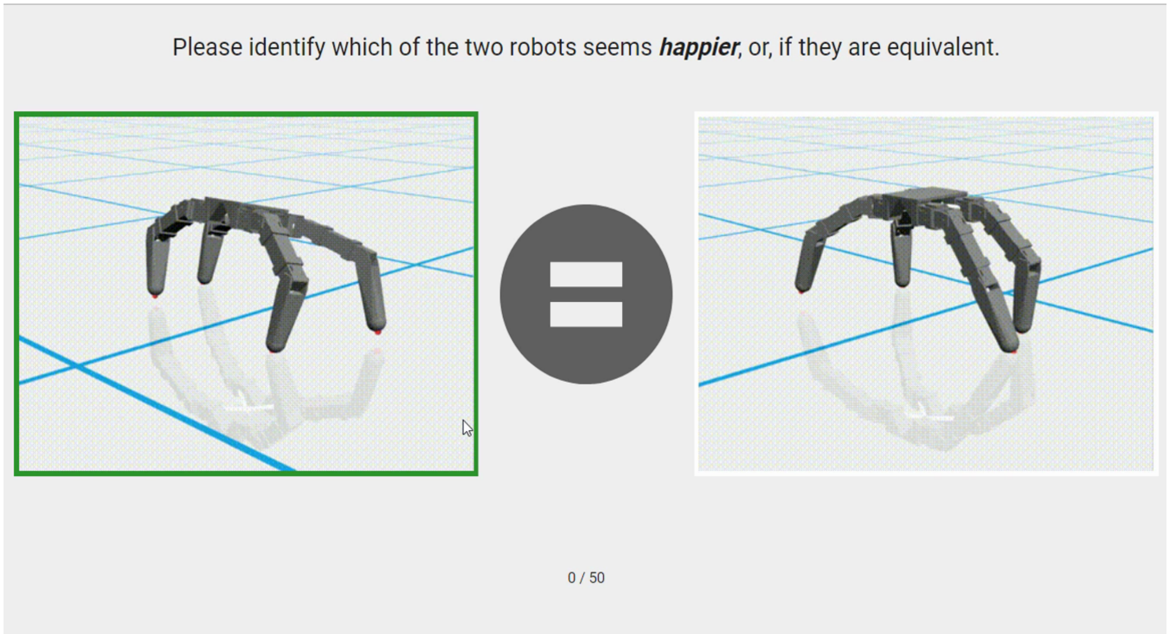


Figure 7.12: Crowdsourcing task for evaluating emotion perception of motion samples – Our interface for crowdsourcing that enables pairwise comparison between motion samples is shown here. A turker is shown videos of motion sample pairs, and is asked to choose the motion that better conveys an emotion relatively (selection is shown in green). Turkers can also mark motions to be equally expressive. Such pairwise comparison is then converted into a Gaussian score estimate for each motion sample.

To efficiently compute emotion ratings of motion samples of our dataset using TrueSkill, we use an elimination-based tournament set-up instead of exhaustively competing each sample against every other. This enables us to efficiently deal with a large number of samples, and the inherent subjectivity in the data, to get the top designs for each emotion. After one round of comparisons, wherein each sample is compared five times (against 5 different designs, by 5 different people), the designs ranked in the bottom half are eliminated. This process is repeated over three rounds (with five, five, and ten comparisons), until we obtain the top motion samples for a given emotion. Elimination of ambiguous, low-ranking samples in earlier rounds allows expressive, high-ranking samples to have a higher number of comparisons against other highly-ranked designs, which improves the quality of their score estimate (reducing the corresponding uncertainty of the estimate). This strategy provides more accuracy for the high-ranking samples, while minimizing resources spent on ambiguous or low-ranking samples. For the quadruped motion dataset there were a total of 3,355 comparisons to arrive at quality rankings for the top 25% of the samples. A more naive approach of a pure round-robin without elimination would require twice the number of comparisons (6,700), and the quality of the comparisons would be lower as there would be more comparisons to low-ranking designs.

To conduct the tournament, crowd workers on Amazon Mechanical Turk [11] serve as judges for each comparison between motion samples. For each comparison, a worker is shown a pair of robot motion videos, and asked *"Please identify which of the two robot motions seems _____, or, if they are equivalent"*, where _____ is one of: happier, sadder, angrier, more surprised, more scared, or more disgusted. (Figure 7.12). Such a pairwise comparison approach has been preferred in the literature over asking the workers to provide an absolute score for individual samples [239]. To ensure the quality of the ratings from crowdsourcing, we devise a filtering pipeline, described next. Inspite of such careful filtering, emotion perception is subjective leading to noisy data. The developed interface therefore accommodates uncertain data.

Filtering during crowdsourcing

Each worker is employed with a *task* of doing 50 comparisons (as in Figure 7.12). We also allow turkers to complete only one such task every 24 hours, in order to prevent a single turker from biasing the results. To further ensure the quality of the ratings, we filter out unreliable turkers. In particular, we use two methods for such filtering, inspired by [239]. First, we expect most turkers to complete a task within a similar period of time. We therefore filter out turkers who do not spend sufficient time on the task, and use less than half of the median task completion time. Secondly, to ensure consistency, we also filter out responses from turkers who either choose the same answer option, or swap their answer choices too frequently over multiple tasks. Specifically, we keep an account of the answer swapping frequency between tasks, and filter out turkers if their swapping frequency is below 35%, or above 75%.

7.7.3 Mapping parameters to emotion

After the data is collected, a mapping between movement and perceived emotion is computed. Specifically, given an n-dimensional motion parameter set ϕ_n and a corresponding real-valued emotion perception score μ, our goal is to learn a function $f : \phi_n \rightarrow \mu$, that predicts the score for any seen or unseen motion represented by its parameter set. Obtaining such a function f that can estimate the perceptual quality of any emotion for a motion allows us to (a) gauge the perceptual quality of a user's motion design at a given time – on the fly, and (b) help the user understand which parameters to edit and how to edit them for achieving the desired effect. The *predictor function* (f) thus powers our parameter-perception curves for manual editing, as well as our automatic slider.

Regression is used to compute such predictor functions for each emotion category. Various regression techniques including linear regression, random forest regression, kernel and nearest neighbor-based regression as well as Artificial Neural Networks (ANN) were explored to obtain these functions. ANN provided the best results on average. However, linear regression also produced results with accuracy similar to ANN, and was much faster to execute (more details are provided later in this section). We therefore used linear regression within *Geppetto*. For the quadruped, the best-fit emotion (happy) had R^2 score of 0.50, and the worst-fit (surprise) had $R^2 = 0.12$ using linear regression (Figure 7.13). The variation in the fit quality for different emotion categories is an indication of the subjective nature of emotion ratings, and the inherent difficulty in expressing nuanced emotions in a parameterized quadruped walking robot motion.

To account for this, we deemphasize ambiguous and noisy samples during regression. We further elaborate on this aspect as well as other details about the regression experiments next.

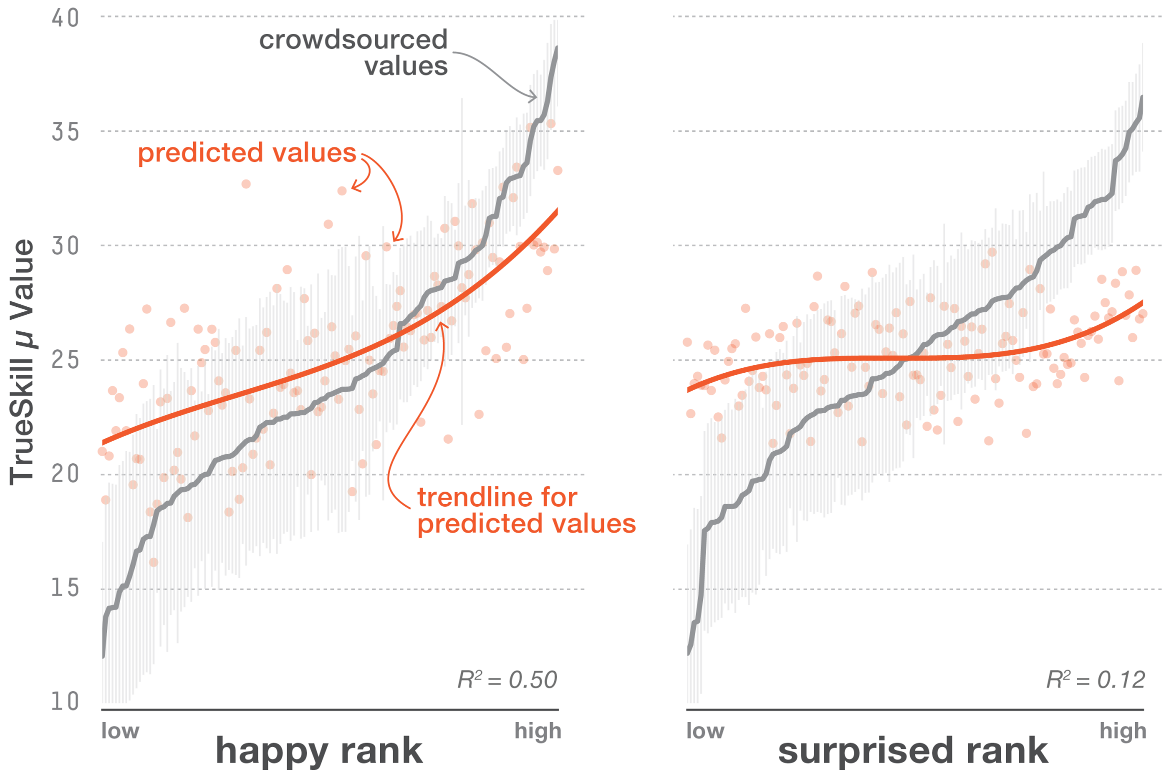


Figure 7.13: Comparison of predicted emotion values (orange) using linear regression with their crowdsourced values (gray) for the test samples of the quadruped motion dataset. The best (happy) and worst (surprised) fitting emotion categories are displayed.

Regression experiments

We now explain our regression experiments in detail for computing the predictor functions. We also elaborate on data preprocessing for these experiments. ANN-based fit was obtained using Tensorflow(r1.3) [87], while all other regression techniques including linear regression were implemented using Python's scikit-learn library [165].

Pre-processing: We convert the discrete parameters in a n dimensional motion parameter set ϕ_n using one-hot encoding, and standardize the continuous parameters to be amicable to the regression analysis [165]. Further, to prevent over-fitting of functions and to enable generalization beyond our sampled dataset, we fit the functions using 80% of the dataset (train set), and withheld the remaining data for testing the accuracy of the fitted functions. Finally, to deal with the noise in emotional score estimates μ (represented by σ, and obtained from Trueskill), we downweight the ambiguous samples with higher uncertainty σ, while computing the mean squared error cost for function fitting. Thus, a weighted mean squared error (MSE) is used for training.

Using weights that are inversely proportional to σ in error computation allows us to focus on fitting the well-expressive samples better.

Regression using ANN: We use separate fully-connected neural networks to fit individual function f for each of our emotion categories. Tensorflow's built-in *Adam* optimizer is used for training [109]. To find the best network for each emotion category, we tune various parameters for the network's architecture such as number of hidden layers, activation functions etc. using standard grid-search [165]. We also use recommended techniques such as drop-out to regularize our networks and to prevent over-fitting.

To evaluate the goodness of fit, we use a statistical measure called R-squared (R^2) or coefficient of determination. R^2 is the proportion of the variance in the dependent variable that is predictable from the independent variable(s) [232]. The weighted R^2 scores for all the predictor functions using the held out 20% of our dataset (test set) are summarized in Table 7.3 for both the walking robot and robotic arm dataset. We used the best found ANN architecture through grid-search in each case for this purpose. Corresponding values for a Linear Regression (LR) fit are also provided for comparison.

Predictor name	ANN R^2 (Quadruped)	LR R^2 (Quadruped)	ANN R^2 (Arm)	LR R^2 (Arm)
Happy	0.512	0.499	0.327	0.325
Sad	0.40	0.396	0.363	0.358
Angry	0.268	0.244	0.397	0.408
Scared	0.245	0.252	0.408	0.405
Disgusted	0.206	0.177	0.225	0.204
Surprised	0.10	0.124	0.20	0.207

Table 7.3: Accuracies of various perception predictor functions f that are learned as artificial neural networks or with linear regression, for the quadruped and robotic arm dataset are shown using the weighted R^2 score on test data.

As seen in Table 7.3 linear regression provided a similar fit and was much faster to execute. So we used linear regression to support design within our system.

Experiments with other regression techniques: Apart from ANN and linear regression, we also experimented with other regression techniques to see if we could get function fits with better accuracy. In particular, we tried out tree-based regressors called random forest regressors, kernel regressors with a variety of kernels (e.g., polynomial kernels, Sigmoid kernels), and nearest neighbor regressors. Similar to ANN regression, relevant hyperparameters were tuned for each regressor using grid-search methods available within the scikit-learn library [165]. Table 7.4 shows the accuracies of these regressors with weighted R^2 scores on quadruped test data for the happy emotion class. ANN and LR accuracies are also shown for comparison. No other regression technique outperformed ANN regression in terms of prediction accuracy on test data.

Regression technique	R^2 for happy emotion (Quadruped)
Random forest regression	0.469
K-nearest neighbor (KNN) regression	0.384
Polynomial kernel regression	0.509
Sigmoid kernel regression	0.472
Linear regression (LR)	0.499
Artificial neural networks (ANN)	0.512

Table 7.4: Accuracies of various regression techniques are shown using the weighted R^2 score on quadruped test data for predicting happy emotion scores.

7.8 Design using predictor functions

The predictor functions f obtained using our data-driven semantic mapping framework serve as oracles during the design process. Specifically, we use f to compute the parameter-emotion perception curves that are displayed with each parameter slider-control for informing the users about the effect of editing that parameter on the resultant expression of user-specified robot emotion. We also use f to drive the automatic emotion update slider.

7.8.1 Computation of parameter-emotion perception curve

Given a motion parameter set, a predictor function f for an emotion outputs the corresponding perception score. Let the motion parameter set corresponding to current robot motion be ϕ_n, such that it consists of n parameters p_i: $\phi_n = p_1, \ldots, p_n$. To compute the parameter-perception curve for the slider corresponding to p_1, we vary p_1 linearly over a range: $[p_1 - \delta, p_1 + \delta]$, and construct corresponding m motion parameter sets: $\phi_n^1 = \{p_1 - \delta, p_2, \ldots, p_n\}, \ldots, \phi_n^m = \{p_1 + \delta, p_2, \ldots, p_n\}$. the values of other parameters $p_2, \ldots, p_n$ in $\phi_n^1, \ldots, \phi_n^m$ remain unchanged from their values in ϕ_n. The corresponding perception scores that capture the effect of varying p_1 in ϕ_n can then be computed using f. The slider curve for a parameter p_1 is thus the plot of $f(\phi_n^1), \ldots, f(\phi_n^m)$. With every manual (or automatic) user-initiated operation that changes the current motion parameter set ϕ_n, we dynamically update all the slider curves. The slider curves also get updated when users change the target emotion for their motion design. Our system re-computes the curves using the appropriate predictor function corresponding to the user-specified emotion selection. Since the predictor functions are obtained from noisy data, the predicted scores are accurate only upto a degree. To illustrate the corresponding uncertainty, we compute and plot the 95% confidence interval (CI) [231] for the predicted score at each point along the parameter-emotion perception curve. The interval highlights the region within which the predicted values will fall with 95% probability. Thus, a wider interval represents higher uncertainty. We compute CI using Python's *Statsmodels* package [189].

7.8.2 Automatic emotion update

The predictor functions not only predict the perceptual quality of an emotion for a motion parameter set, but also provide information about regions in the parameter space that correspond to better emotional expression. Starting from a point in the parameter space ϕ, such regions can be reached by moving along the direction of predictor function f's gradient ($\frac{\partial f}{\partial \phi}$). The automatic emotion update slider leverages this to update the robot motion. Unfortunately, since ϕ consists of both discrete and continuous parameters, we cannot compute the gradient $\frac{\partial f}{\partial \phi}$ with respect to all parameters. Consequently, when the automatic slider is used, we update the discrete and continuous parameters one by one. We first update the discrete parameter to achieve the user requested change as best as possible. Given the discrete parameter's value, we then change the continuous parameters using the gradient-based update. Specifically, for a given motion parameter set ϕ with continuous parameters set ϕ_c, the updated parameter set ϕ'_c is $\phi'_c = \phi_c + \delta \frac{\partial f}{\partial \phi_c}$, where δ is the step-size along the gradient. The step-size is proportional to the change in the slider cursor position (Δ), which consequently reflects the desired change in robot's emotional expression (Δf). The step-size δ required to achieve the desired Δf is computed using backtracking line-search [151]. δ is positive if the user moves the automatic slider to increase the emotional expression (towards the green end of the slider's heat map), and is negative otherwise.

7.9 Evaluation with user-study

To evaluate *Geppetto*'s efficacy and features, we conducted a user-study with participants who had no prior experience in character animation, or HRI.

7.9.1 Participants

12 participants (9 males, 20-35 years of age) were recruited. Participants were reimbursed $25 USD for their time.

7.9.2 Study design

The study had a within subject design, with participants creating expressive motions for the quadruped using two versions of the system (Figure 7.14). The *parameter control* UI allowed editing robot motion parameters with sliders but did not provide informative curves, automatic sliders, or the gallery. The *semantic control* UI was the full interface as described above. Since the quality of guidance provided by the semantic control UI depends upon the predictor function accuracy for an emotion, the emotion categories with highest (happy, sad), and the lowest (surprised) predictor function accuracy were used. The order of the UI conditions and emotions were counterbalanced.

7.9.3 Procedure

The study began with an overview of the design task for 5 minutes, followed by participant training and motion design sessions for 50 minutes, concluding with a 5-minute survey. The survey

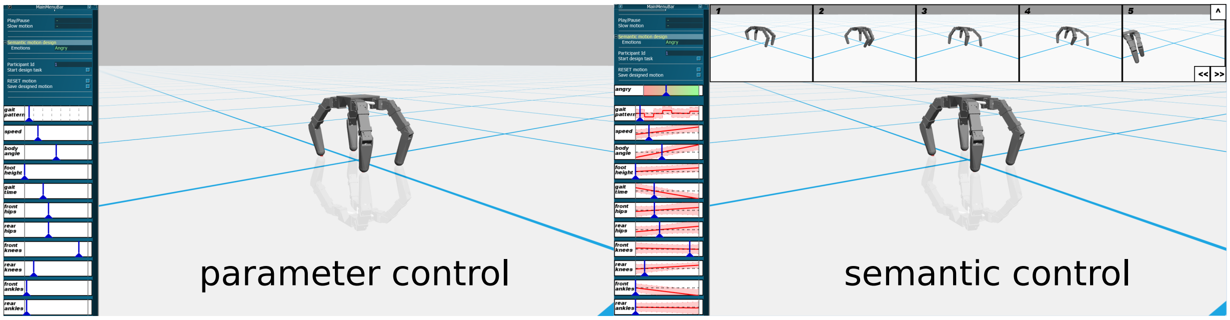


Figure 7.14: Interfaces used in the study – participants designed expressive robot behaviors with two UI versions: parameter control (left) and semantic control (right). Each version used the same underlying physics simulation and motion synthesis, but provided different controls for editing the motion.

consisted of 5-pt Likert-scale questions (anchors: 1= not at all; 5 = extremely) to understand user perception of various UI features and overall design experience. We discuss the survey questions and corresponding participant responses later in the chapter. For each condition, the participants were given the UIs demo and training for up to 10 minutes. Post training, participants were given up to 5 minutes each for designing happy, sad, and surprised robot motions. Thus, each participant designed 6 robot motions in total. Participants' motion designs were automatically saved every 30 seconds as well as when they indicated they were complete.

7.9.4 Results

Quantitative results

We compare the parameter control and semantic control UI using two quantitative measures – design time, and design quality. The perceptual quality of emotional expression in the user-created motion designs are evaluated using crowdsourcing, with the top and bottom 5 synthesized designs for each category included in the tournament. The tournament structure and crowdsourcing pipeline are similar to our earlier experiments. We analyze corresponding scores using confidence intervals and effect sizes, instead of null hypothesis significance testing [58]. This choice is inspired by increasing concerns over such hypothesis testing for experimental results in various research fields [57, 66, 185].

The resultant scores show that users were able to create better expressive motions on average using the semantic UI, across all emotions (Figure 7.15). *Effect sizes* help quantify how much more effective is semantic UI over parameter control UI in enabling expressive behavior design. Table 7.5 shows these effect sizes computed using Cohen's d parameter[5] for happy, sad, and surprised design conditions. Cohen's effect size value (d = .6 and higher) suggest a moderate to high practical significance or effectiveness, specifically for the happy and sad categories. Although surprised motions from the semantic control UI scored higher on average than motions from the parameter control UI (Figure 7.15),the lower effectiveness (0.35) can be attributed to

[5]Cohen's d is an effect size used to indicate the standardized difference between two means.

the limited data-driven guidance available, highlighting the effect of data certainty on semantic UIs performance.

Parameter vs. Semantic UI (emotion category)	Cohen's d
Happy	0.79
Sad	0.64
Surprised	0.35

Table 7.5: We quantify the effectiveness of semantic UI over parameter control UI using effect sizes computed using Cohen's d parameter. The d values of 0.6 and higher suggest a moderate to high practical effectiveness of semantic UI over parameter control UI.

We also find that the designs created using the semantic UI outperform the best motions from our original dataset in expression of various emotions (Figure 7.15, Semantic vs. Synthesized). This points towards both the strengths and drawbacks of our system. The dataset synthesized using sparse random sampling may not be capturing the design space with high fidelity. Subjective crowdsourcing analysis of the dataset adds further ambiguity and noise to the data. Despite this, *Geppetto* allows users to explore beyond the synthesized dataset, by enabling and leveraging their intuition of parameters at design time, guided by the emotion predictor functions.

Along with obtaining more emotive final outcomes, the participants also tended to take less design time on average with the semantic UI (Figure 7.16).

Overall, the semantic UI enables users to start with better designs and to explore higher quality designs during their session (Figure 7.17). The higher initial scores of designs from the semantic UI in Figure 7.17 can be attributed to the use of motion gallery. After this initial boost, however, the semantic UI enables users to further improve the quality of their designs through features such as the annotated sliders and parameter comparison cursors. This is evident in the upward slope of the orange line representing semantic UI in Figure 7.17. Thus, the gallery, the annotated sliders, and parameter comparison cursors together provide a powerful workflow that allows users to achieve more optimal designs.

Qualitative feedback and observations

The survey provided further insights about designing with our system.All participants reported that they are extremely likely to prefer semantic control UI to parameter control UI (5-pt Likert score, with 5 = extremely: M = 4.67, SD = 0.49). Participants also believed that with the semantic control UI, they could create relatively better designs (M = 4.67, SD = 0.49), in less time (M= 4.83, SD = 0.38). Figure 7.18 shows corresponding survey questions that enabled such a comparison of the semantic and the parameter control UIs. This feedback further corroborates the quantitative results. Participants design satisfaction varied across emotions, and was dependent on the quality of semantic information provided. Consequently, 11 of 12 participants were satisfied with their happy design, while only 2 of 12 participants were satisfied with their surprised design (Figure 7.19).

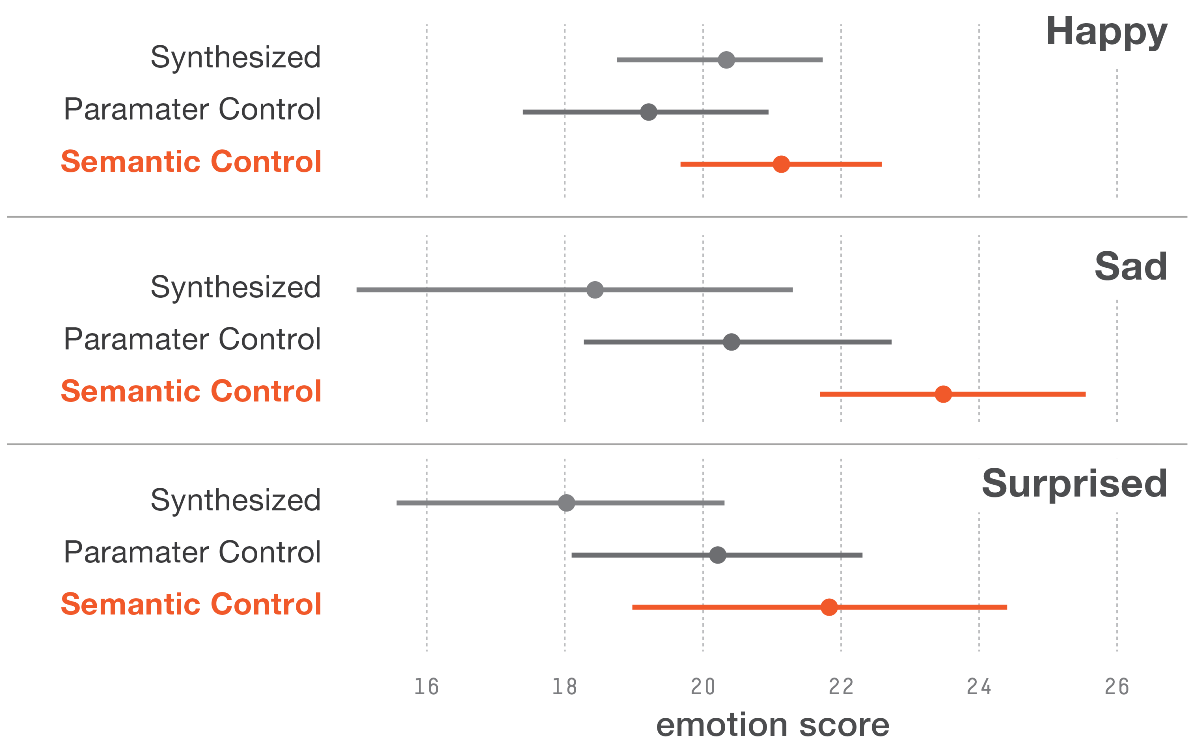


Figure 7.15: Mean emotion perception scores of the top 5 designs from the original dataset (Synthesized) with those created by the study participants. Bars show 95% CIs. Note that the emotion score ranges were based on the output of Trueskill system, with higher score corresponding to better quality of expression.

Figure 7.16: Individual and average design times are shown using dots and lines respectively, for both of our UIs. Shaded regions represent 95% CI.

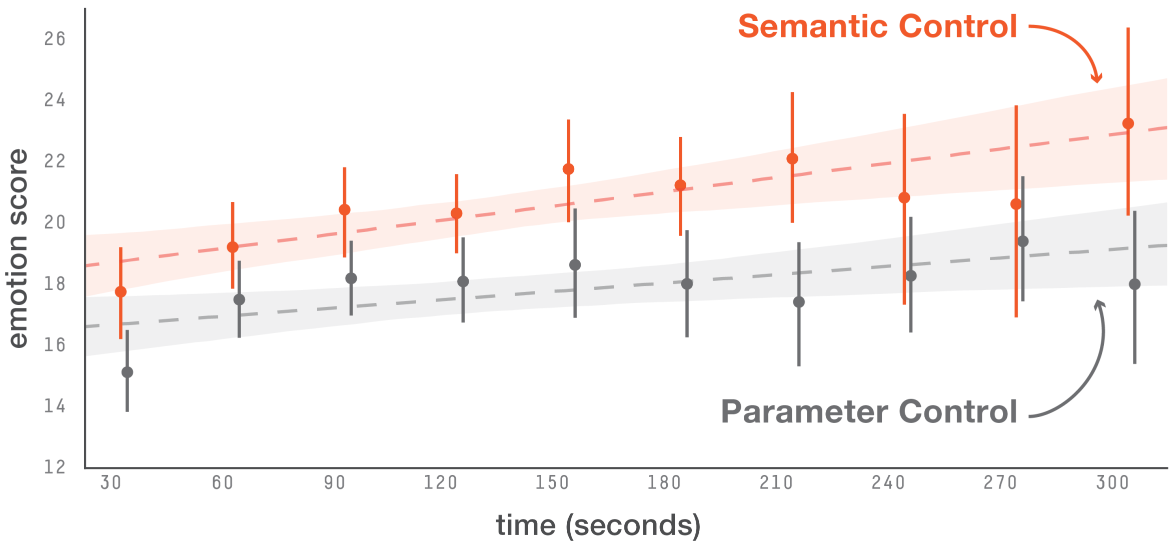


Figure 7.17: The evolution of the quality of user-designs (bars represent 95% CIs at each time step). The dotted lines represent the linear fit of mean scores over all emotions and participants, and the bands are a 95% CI around the fit.

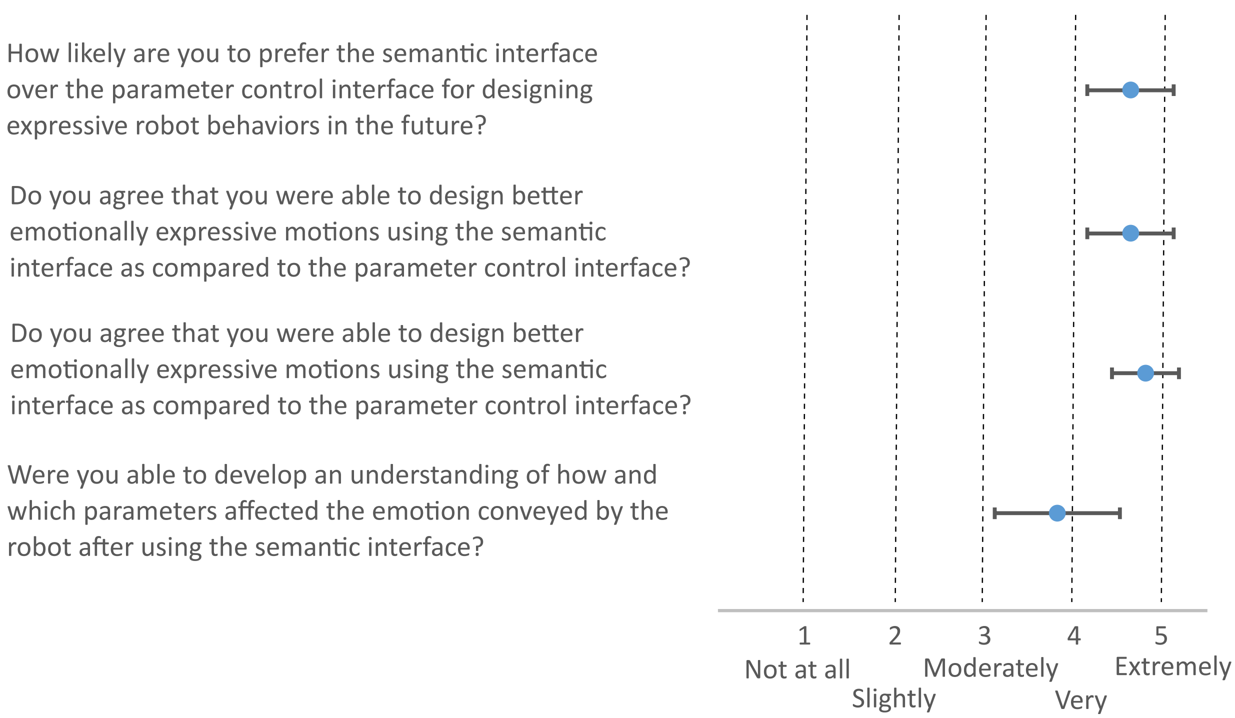


Figure 7.18: Survey questions and responses highlighting the participants' reception of the semantic interface. Average scores and standard deviation are shown for 12 participants.

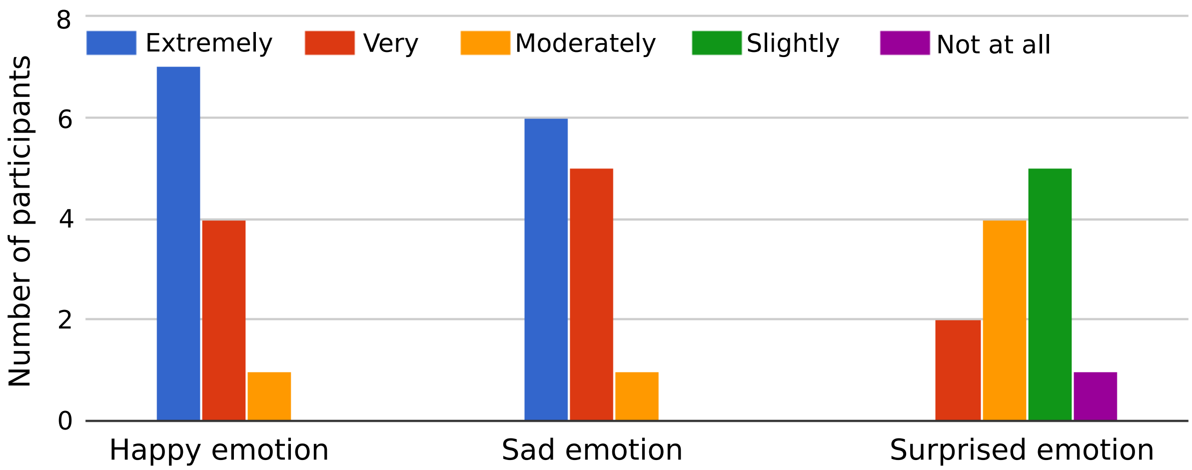


Figure 7.19: A histogram of number of participants vs. the participants' amount of satisfaction for their robot behavior designs is shown.

We also asked the participants for feedback on individual UI features using Likert-scale questions (Figure 7.20). 10 participants found the motion gallery and slider curves to be extremely or very useful. The parameter-comparison cursors and the automatic slider were also found to be extremely or very useful by 6 participants. The gallery catered to participants who were unclear about how to express an emotion, as well as to participants who had crude ideas about their desired design by providing them with design alternatives. Uncertainty information on slider plots was also found useful. Specifically, two participants commented that since the surprised emotion parameter-emotion perception curves had high uncertainty (surprise is our worst-fitted emotion), they trusted the curves less, and explored editing on their own. Parameter locking was only found to be very useful by 3 participants. Only the participants who had a clearer idea of what they wanted used parameter locking. Finally, 8 participants reported the semantic UI to be extremely or very useful in developing an understanding of the effect of parameters on emotional expressions. Overall, the participants explored more while designing with semantic control UI owing to the availability of more features and design alternatives.

The feedback and usage patterns points to the diversity of interactions and workflows that emerged during the study. Participants combined manual and automatic editing features fluidly. The feature usage also varied across participants. For instance, some subjects only used the motion gallery for design initialization, while some others leveraged it, with the help of parameter comparison cursors, to better learn and understand how specific body poses and other subtle motion features could be achieved. The automatic slider was also used in multiple ways; some used it to fine-tune their manually edited motions, while others used it to obtain a good starting point especially when they were dissatisfied with the gallery examples. This highlights the dependence of workflows on the noise in the data and accuracy of semantic information. Since surprise was not well captured by our dataset or individual predictor functions, participants used the automatic slider the most for this emotion.

The participants also provided feedback about the limitations of our design system. Some participants found the automatic slider to be very aggressive since it made major changes in

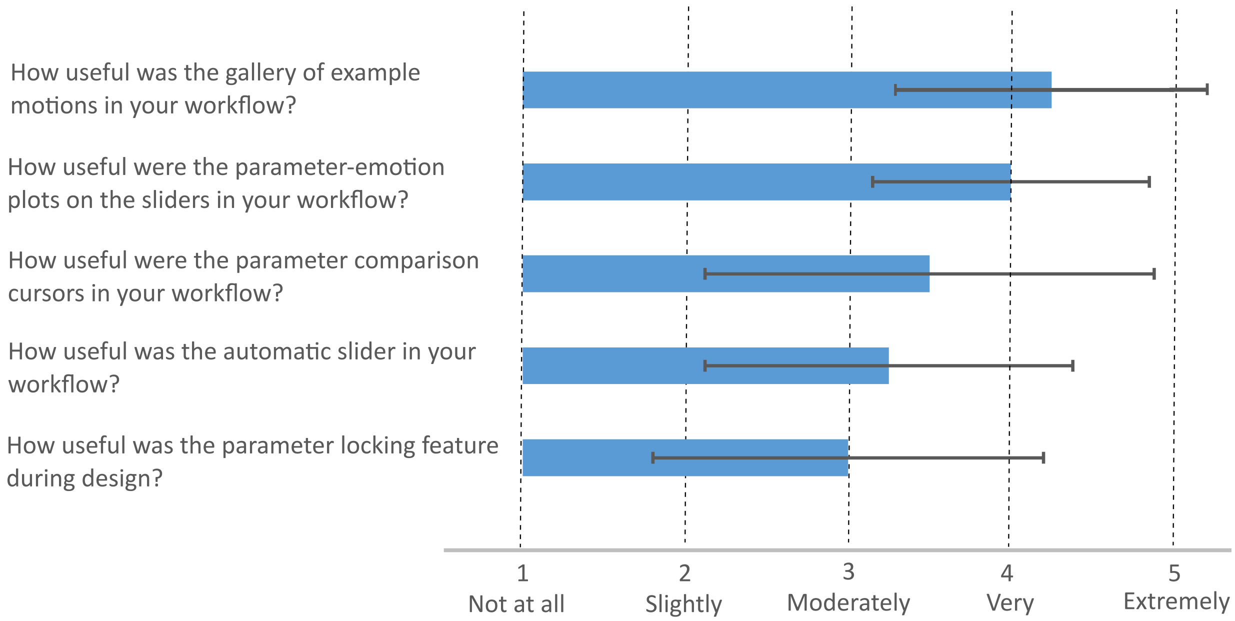


Figure 7.20: Participants provided feedback about individual UI features using 5-pt Likert scale questions. Average scores and standard deviation are shown for 12 participants.

the motion, resulting in the loss of nuanced features of the motion. While parameter locking helps with capturing user intent about desired improvement and preserving nuances, it needs more understanding of the parameters and desired motion characteristics for effective use. The majority of participants requested an edit history and better navigation of their design trajectory. Some participants also requested the ability to edit robot structure and aesthetics for more expressiveness. Finally, participants echoed the need of capturing and enabling motion design with additional semantic information. Many participants thought about expressive motions in the space of actions and wanted to understand the mapping between parameters and space of possible and meaningful actions, so as to combine these actions into a behavior. For instance, a participant wanted to edit the parameters to make the robot drag its feet for appearing sad, while another participant wanted the robot to jump in place to express excitement. While our gallery enables users to map parameters to these desirable actions indirectly, users may or may not find the action they are looking for in the gallery owing to the sparse sampling of the dataset that powers the gallery.

7.10 Generalization and scalability

Geppetto's framework generalizes across different kinds of robots as demonstrated by the quadruped and robot arm examples. Overall, the most challenging part of making *Geppetto* work for a new robot is obtaining a parameterized motion simulation. Once a dataset of motions is created, *Geppetto* requires approximately 4.5 hours and $150 USD for crowdsourcing per expression. While this may not be a significant amount of effort, re-using semantic information extracted from a particular robots dataset to enable the design of a robot with different morphology will improve

Figure 7.21: A happy motion of KUKA robot arm (left) was directly transferred to another robot arm (right) consisting of different link lengths and DOFs. We also used a different target point position (shown in blue) for the custom arm. The nature of motion was not effected by change in the target point position either. The swirling and swinging characteristics of the motion that led to the perception of excitement/happiness in the KUKA arm manifested in a similar manner on the custom robot arm. These motions can be found in our video or in the animated version of this thesis.

the scalability of *Geppetto*. Such transfer of the data-driven semantic map between robots, however, is dependent on the underlying motion behavior parameterization. For the quadruped, since the motion is parameterized in the joint space, the parameterization and the corresponding semantic map is dependent on the robot's morphology. To design behaviors of a six-leg hexaped, for instance, using the quadruped's semantic map, the joints of the extra pair of hexaped's legs will have to be mapped to quadruped's front or back leg joints. Since this might limit the possible hexaped behaviors that can be designed with *Geppetto*, collecting a new hexaped motion dataset might be more preferable over re-using the quadruped dataset.

On the other hand, the robot arm's motion is parameterized in task space and is thus independent of the robot morphology. The corresponding semantic map might hence transfer well to different types of robot arms. We conducted a preliminary experiment to validate this. Using the semantic map of the 6 DOF KUKA arm, we tried to design motions for a 5 DOF custom robot arm. Apart from the difference in number of joints (DOF), the two robot arms also had very different link lengths. We were able to directly transfer the example motions of KUKA arm from the motion gallery to the custom arm, while maintaining the resultant expression perception to a reasonable extent. Figure. 7.21 shows one such KUKA arm motion for happy emotion that was well transferred to the custom arm.

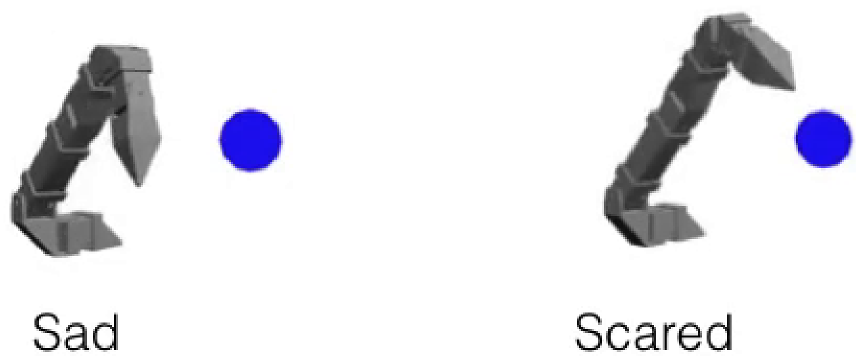


Figure 7.22: By re-using the parameter-emotion relationships of KUKA arm, we were able to design emotional expressions for a very different robot arm. These motions can be found in our video or in the animated version of this thesis.

The parameter-emotion relationships derived for the KUKA arm were also applicable to the motions of custom arm. This allowed direct editing of custom arm's motion using the annotated sliders in *Geppetto*. To enable such editing we updated the ranges of speed parameter, as the custom arm consisted of servo motors that are much less powerful than KUKA's actuators. Apart from speed, no other parameter needed such range re-scaling (see Table 7.2 for list of robot arm motion parameters). Figure 7.22 shows a sad and a scared motion of the custom arm designed manually using the sliders annotated with parameter-emotion curves, which were derived based on the KUKA arm's dataset. Further work is needed to validate these experiments explicitly.

7.11 Frequently asked questions (FAQ)

1. **Why does *Geppetto* focus on emotional robot behavior design?**
   Geppetto explores emotional expressions as they are important to HRI [26], and they are a challenging problem to tackle. However, *Geppetto*'s framework can likely extend to aspects beyond emotion (e.g., expressing that a robot is busy, awaiting instruction, or friendly).
2. **How were the emotion categories decided for *Geppetto*?**
   Various emotion classification models have been developed by researchers in the past. Prominent among them are Russell's circumplex model [181] and Ekman's model of basic emotions [71]. Russell's model uses the axes of arousal and valence for defining a variety of emotions. Instead, we grounded our emotion selection using Ekman's classification of six basic emotions, as we believed it to be more easily understood by crowdworkers (who may struggle with the concepts of arousal and valence).
3. **How much effort is needed to set up *Geppetto* for a new robot or a new expression?**
   The effort needed to set up *Geppetto* for a new robot or expression is dependent on the complexity of robot and expression, and robot's parameterization. Usually, given a good parameterization and a dataset, as mentioned earlier, the only effort needed to add an expression is crowdsourcing + regression, which in our experience was approximately 4.5 hours and $150 USD. For a new robot, obtaining a parameterized dataset is most challenging in our experience. For quadruped, once a parameterization was understood based on literature, it took two days to obtain our current dataset. For the robotic arm, multiple parameterizations were tried e.g., spline-based, animation principles-based etc., before finalizing on Boids. Once parameterization was finalized, it took six hours to generate the dataset. The quadruped's simulation had more feasibility constraints than the robot arm's simulation, which resulted in longer times for the dataset generation.
4. **Why did you encode emotions using sampling + crowdsourcing approach, in-spite of the noise inherent in crowdsourcing?**
   The high dimensionality of the motion design space makes it hard to encode expressions with high fidelity using any approach. Encoding expressions using experts is a possibility but is less accessible as compared to our sampling + crowdsourcing approach. To ensure quality of crowdsourcing the data, we perform various checks, such as enforcing a minimum time requirement for each comparison, and excluding data from workers who choose the same response (i.e., left, right, or equal) repeatedly. The subjectiveness of evaluating emotions still results in a inherently noisy dataset. Thus, one of our goals is to enable design with noisy data. Specifically, we take extra steps to deal with noise during regression. We also show how mixed-initiative design can work when the noise is high, by using uncertainty band annotations on the sliders.
5. **Why does *Geppetto* struggle with creation of surprised expression for quadruped? Why was surprise emotion included in *Geppetto*?**
   Surprise is a difficult emotion to convey well for the quadruped, in part, because of the periodic gait parameterization. We included, and highlighted this result in the paper to show how choice of parameterization may limit the achievable expressions. That said, some par-

ticipants created designs that were better than the synthesized ones for surprise (Fig. 7.15), highlighting the benefits of our mixed-initiative approach for situations wherein limitations of parameterization are not known apriori.

6. **Why was the robot arm motion synthesized using Boids?**
Using Boids flocking simulation to generate robot arm motions was purely an empirical choice. We consulted an animator and previous research for identifying relevant motion trajectory properties for expressiveness such as smoothness, curvature, variable speed, and timing. We initially experimented with spline-based parameterization and modeling of animation principles such as "anticipation" in the beginning of the motion and "settling" at the goal [216], for obtaining such trajectories. However, achieving natural looking yet variable speed and arm orientation over the trajectory turned out to be non-trivial. We then experimented with Boids, inspired by an artist created interactive arm exhibit called Mimus [214]. Boids resulted in relatively better motions on average and so we used it for our final system.

7. **Why were the low-level parameter editing sliders provided in *Geppetto*? Isn't it sufficient to use the automatic slider that can directly edit the motions semantically?**
The automatic slider complements the low-level parameter sliders, instead of replacing them. Specifically, the low-level sliders provide users with control to edit "preferred" parameters.

8. **How well will the motions designed with *Geppetto* work on real robot hardware?**
Our simulation accounts for collisions, friction, gravity, joint limits, and motor torque limits necessary for physical feasibility using constraints. This greatly helps the transfer of motions from *Geppetto* to real hardware. Previously, Megaro et al. have tested motions generated using this simulation framework on three types of walking robot hardware prototypes [134]. However, such a transfer is most times impeded by extreme user edits, which may drive the robot motion parameters to infeasible regions at design time (e.g., an extreme user-commanded pose for the quadruped could result in the robot dragging its feet in the ground – an infeasible motion for a real robot). *Geppetto* currently warns the users when such infeasibility occurs, but doesnt provide feedback to the user in correcting such issues. Users themselves have to reverse some of their extreme edits to do so. We hope to provide more support to the users for the same in the future.

9. **It seems that the higher quality of participant designs created with semantic UI can be attributed to the use of motion-gallery. Doesn't this mean that people are better at *evaluating* expressive motions rather than *creating* them? Isn't this a crippling co-found of the user-study?**
The motion gallery is a valuable part of the interface, and because it is one of the contributions of our system, including it doesn't cripple the study. The study is indeed measuring behavior creation rather than behavior evaluation. None of the participants merely selected from the gallery and finished without modifying their designs - all utilized the automatic slider or the comparison cursors and curves to understand and tweak the low-level parameters. Additionally, participant designs on average outperformed the gallery designs (Fig. 7.15), demonstrating that they were in fact creating new/better designs rather than

simply evaluating existing motions. This is further shown by the upwards slope of the orange line in Fig. 7.17, which remains roughly parallel to the gray (parameter UI) line. The semantic UI line starts higher, indicating that the gallery may provide an initial bootstrapping effect, while the other interface elements enable users to create new and better motions.

7.12 Limitations

Limited encoding of user-intent: Currently, our system allows design space exploration and editing given a single high-level semantic goal. Enabling concurrent design for multiple semantic design goals such as to express a mixture of emotions will provide the users with greater flexibility of design. The user-design experience may be further improved by capturing user-intent in more detail. Instead of using only high-level semantics to capture user-intent, high-level semantics could be further coupled with mid-level semantics relevant to the task. For instance, mid-level semantics corresponding to emotionally expressive motion design may correspond to actions such as dragging feet, jumping, or appearing crouched. Such a representation could also enable users to gain a better understanding of the design space.

Limitations of the dataset generation: Our system will also benefit from better dataset generation techniques. In particular, adaptive sampling which focuses on regions with better designs would allow the system to capture the design space with more fidelity. Additionally, on-demand sampling and dataset generation at design time may further enable the system to provide customized guidance and design alternatives based on user-preferences.

Limited generalization: Re-using semantic information extracted from a particular robot's dataset to enable behavior design for a different robot will also be very essential for the scalability of *Geppetto*. Our preliminary generalization experiments showed how parameterization was a bottleneck for the same (Sec. 7.10). Principled approaches that can map the motion parameters of different robots or their underlying controllers systematically will therefore be important in the future.

Limitations of motion synthesis: Finally, a simulation-driven design system like ours can only be as good as the underlying simulation. Our current motion parameterization and simulation doesn't produce motions suitable for conveying subtle emotions such as disgust and surprise. Parameterizing and synthesizing emotionally expressive robotic behaviors is an exciting future area of research. We also currently limit ourselves to the creation of robotic expressions and personality through motions only. However, aesthetics and physical structure are equally important for visual appeal. Parameterization and intuitive editing of aesthetics is thus an interesting open problem. In particular, we envision a semantic design system that exposes the coupling of physical structure and motion towards creating appealing robots. Such a system will not only support the design of next generation of social and collaborative robots, but will be equally valuable for consumer robotics.

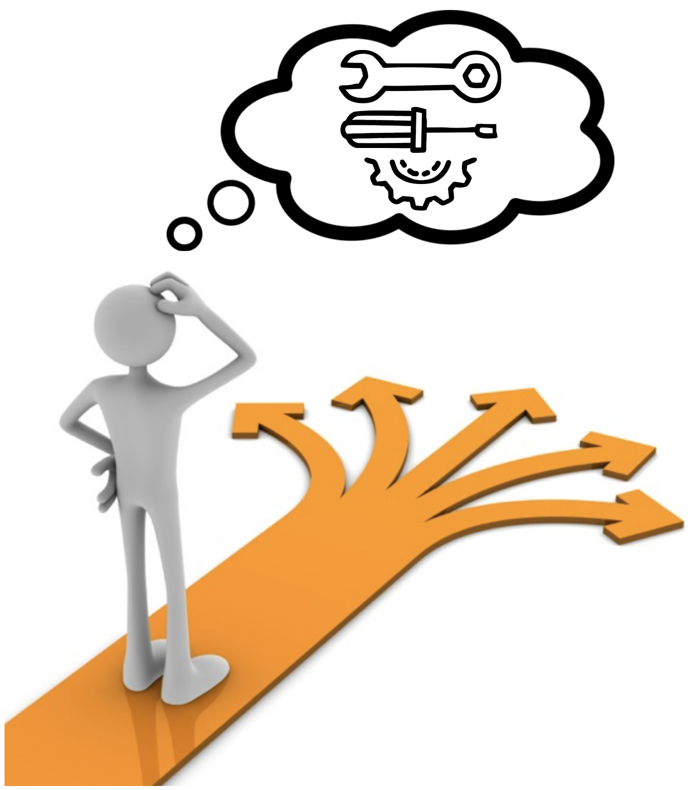
7.13 Publication and dissemination

This work is under review for ACM CHI Conference on Human Factors in Computing Systems 2019. It was done in collaboration with Fraser Anderson, Justin Matejka, and Tovi Grossman from the User Interface research group at Autodesk Research, Toronto.

An overview video about *Geppetto* is available at – `https://youtu.be/DXbnwodJ2Ks`

Part IV

The road ahead



Chapter 8

Summary and future work

8.1 Summary of contributions

This thesis explored a relatively less-treaded research area in robotics – tools for creating robots. A lot of current research in robotics focuses on making a given robot smarter and intelligent, rather than on the design and creation of a robot (for instance, the top 3 keywords in the most recent and largest robotics conference, ICRA 2018, were deep learning, planning, and system adaptation[1]). In contrast, our goal was to draw attention to the equally important area of robot design. We specifically focused on supporting novices that are interested in building robots, and leveraged prior work in computational design, graphics, human-computer interaction (HCI), and robotics for the same.

To aid novice users in designing and building robots, this thesis presented software design tools that enable accessible mixed-initiative design. Accessibility was achieved in particular, by encoding relevant domain knowledge about the design task. Algorithms and numerical optimization techniques were leveraged for automating tedious and difficult parts of the design. The tools still kept the user in the loop of the design process through visual interfaces and necessary design feedback. Specifically, simulation-driven, data-driven, and model-driven feedback modalities were explored for allowing users to be an active part of the design process. We also explored how user-intent and preferences can be captured with limited inputs from the users. Table 8.1 summarizes these fundamental characteristics of the tools presented in this thesis.

Our work also highlighted many fundamental challenges in grounding the design tools for target users as well as in evaluating such tools. While user-studies and surveys are under-represented in robotics community, we found them to be immensely useful for understanding the needs and struggles of target users, as well as for evaluation (as in assembly-aware design system (Chapter 3) and in *Geppetto* (Chapter 7)). These studies and surveys also exposed us to the many variations that exist in possible interactions and design workflows for creative design applications. There is a fine line between too much automation and too little automation that needs to be treaded with care in such applications. This fine balance can only be found through experimentation as per our experience. Finally, hardware prototypes are equally important to

[1]as per statistics shown by ICRA 2018 program chair Peter Corke. More information can be found here:`https://danieltakeshi.github.io/2018/05/23/icra-day1/`

Table 8.1: Summary of contributed tool features

	Principle features of the tools		
	Domain knowledge encapsulation	User-in -the-loop	User support
Non-articulated structure design tool	Spatio-temporal device design model	Model-driven feasibility and fabricability feedback	Automation of tedious, difficult design steps
Modular, articulated structure design tool	Model of components and connections	Feedback on possible robot structure & motion	Design iterations in software
Co-design tools for articulated robots	Model of coupling between robot's structure & function	Simulation-driven feedback on robot's task-specific function	Automatic design generation/improvement for user-specified task
Semantic design tool for robot behaviors	Data-driven model of robot behaviors	Data-driven feedback on robot's motion properties	Design space exploration

support the capabilities of robot design tools, since enabling real world designs is the ultimate goal of these tools. We therefore used the combination of user-studies and fabricated prototypes as our core evaluation strategy. We would also highly recommend such a combination for validation of robot design tools in the future.

8.2 Directions for future work

8.2.1 Towards more accessible design

While this thesis took steps towards democratizing the robot design process, there is yet a lot to be done for enabling accessible holistic development of robotic systems in the real world. Truly accessible robot development will require capturing user-intent at different granularities, while still allowing users to provide inputs at higher-level. Enabling users to develop robots based on high-level, semantic descriptions such as 'I want to create a robot for task X with size Y and Z power requirements' is indeed the holy grail of accessible robotics (Fig 8.1). However, our efforts also showed how a single tool might not be able to cater to the needs of a wide variety of robots out there. Thus, tools tailored to specific classes of robots that account for all necessary design aspects in a holistic manner might be a more plausible future for robot design tools.

Towards increasing accessibility, immediate directions to explore include leveraging multiple modalities for capturing user-intent, and taking advantage of existing robot designs. Next, we briefly discuss relevant opportunities for the same.

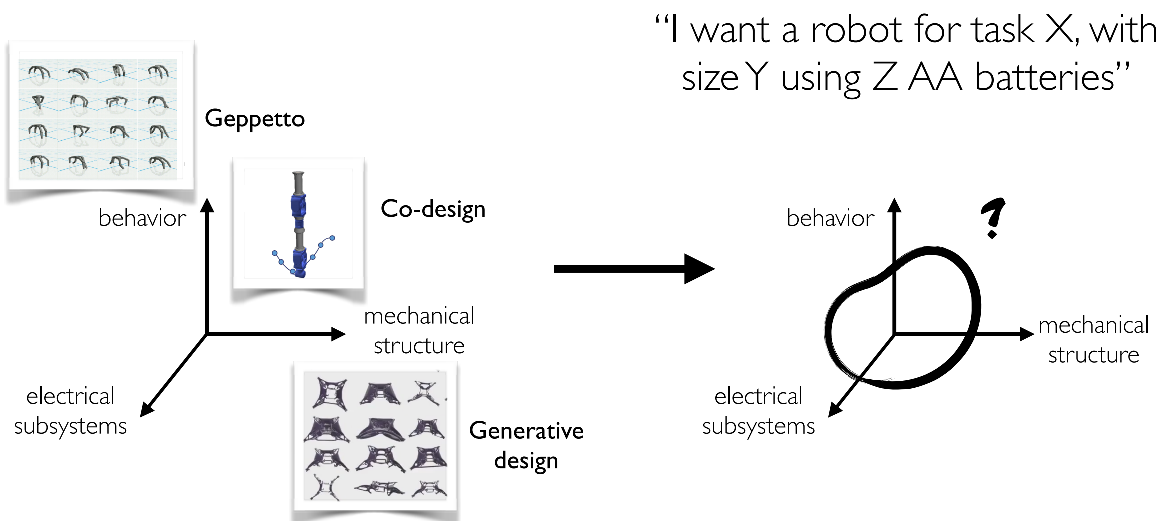


Figure 8.1: Future vision for accessible holistic robot development – Allowing users to design and build robots based on high-level descriptions is a powerful approach for democratizing the robot design process. This thesis showed such an approach specifically for robot behaviors in *Geppetto*, and for co-design of robot's structure and behavior. Other approaches such as generative design have similarly shown design of mechanical structures that satisfy high-level user-specified constraints. In the future, enabling design of all aspects necessary for a robot based on high-level user-provided descriptions will be essential for truly accessible and holistic robot design.

Exploring and fusing multiple modalities for capturing user-intent

Along with semantic, high-level user inputs, multi-modal inputs have immense potential in capturing and conveying user-intent. For a given design application, users might be able to specify what they desire better using either example images and sketches, or even through natural language. As a more concrete example, consider the scenario of using a generative design tool for designing an artistic monitor stand. Typically, users have to specify localized weight/loading for their desired design, boundary conditions etc. as inputs to a generative design tool, for obtaining relevant design alternatives [131]. Instead, sketches can be leveraged to capture aesthetic requirements. Combining them with inputs in natural/text-based language that specify intended use of their desired design might lead to a much accessible design system (Figure 8.2).

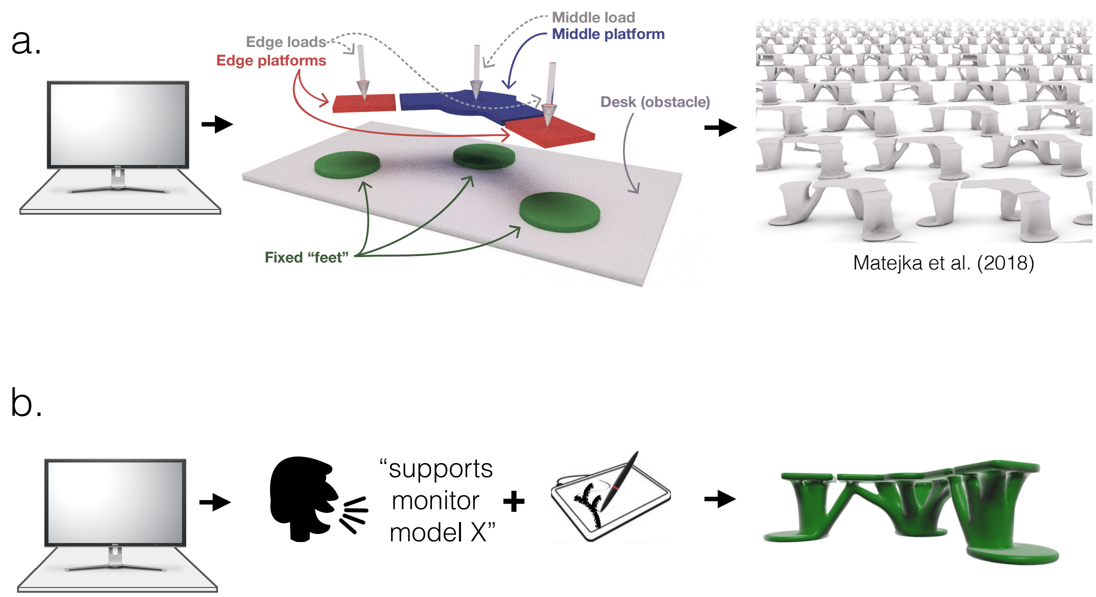


Figure 8.2: Multimodal inputs may be immensely useful for capturing user-intent, and may simplify the use of complex design tools. (a) For instance, current generative design tools require loading, boundary conditions etc. as inputs from the users. (b) Instead, a combination of natural language and sketch-based inputs may be leveraged to not only enable users to provide inputs easily, but also for providing users with more relevant design alternatives that better match their needs.

Universal format for robot design

Re-use of existing and relevant artifacts has greatly benefited varied communities ranging from digital art and media designers to programmers. Existing designs can not only serve as inspiration, but can also bootstrap the design process. Currently, existing robot designs are hard to re-use because of multiple formats involved in storing electrical, mechanical, and behavioral components. Formats such as Unified Robot Design Format (URDF) used by ROS [179] attempt to store a bulk of information about kinematics, environment obstacles etc., but they do not contain fabrication level information about CAD files or components. A universal format that stores information about all of these aspects together will support interoperability, and will enable the creation of large scale datasets of re-usable robot designs. Such datasets could then support the next generation of designers as well as future data-driven and accessible design tools.

Beyond accessibility, the development of next generation robot design tools will benefit by – (a) taking inspiration from the software and cyber-physical systems (CPS) design communities, and (b) by thinking beyond automation for supporting the users.

8.2.2 Leveraging ideas from software and cyber-physical systems design communities

Software development shares many challenges of robotic system development in terms of required scalability and complexity of design approaches. Further, similar to robot development, software development also requires one to deal with a multitude of design aspects. Modular frameworks, version control, continuous integration, unit testing etc. have tremendously helped the software development community in dealing with these challenges (Figure 8.3). Creating a robot development ecosystem that embeds these capabilities is an exciting direction of future work. Such an ecosystem will be able to support not only novices but also experts in efficiently designing robots.

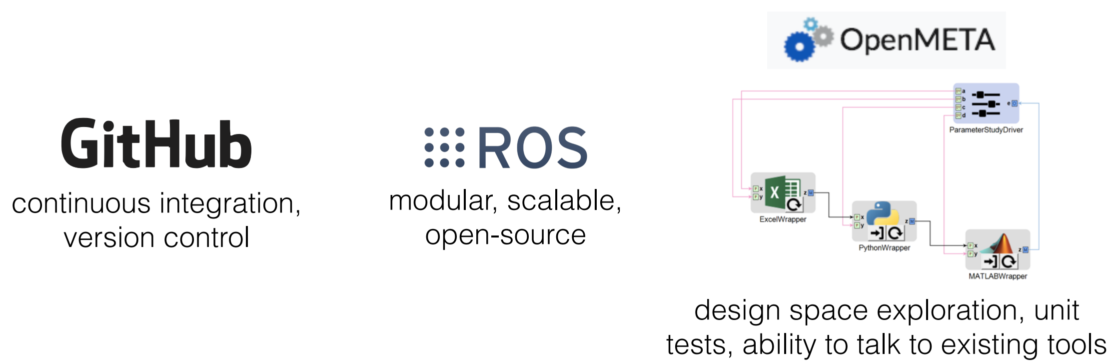


Figure 8.3: Essential features for a future robot development ecosystem – Robot development ecosystems in the future will have to embed ideas from software and CPS design communities such as modularity, version control, ability to talk to existing tools etc. to efficiently support expert and novice designers.

Future robot development ecosystems should also be capable of reusing existing commercial and open source tools as identified by another relevant design community – the CPS design community [208]. Integration of the best available design platforms such as CAD tools for mechanical design, Matlab for behavior design etc. within a single framework will allow designers to holistically look at robot designs. Designers will be able to perform design space exploration and unit-testing at the scale of diverse design aspects. They would be able to ask questions such as *“What would happen if I change my robot body material from steel to aluminum?”*, and obtain relevant automatically generated statistics about the same. Towards this goal, OpenMETA – a tool from the CPS community can serve as a good starting point [156, 207]. Specifically, OpenMETA allows talking to existing design tools, and to perform design space exploration at multiple scales.

8.2.3 Beyond automation: Intelligent tools that also enable learning

Tools presented in this thesis as well as existing design tools out there (see Chapter 2) focus only on how to make it easy for people to perform a complex design task. These tools do not enable people to learn over time so as to become better at the respective design tasks. However, development of tools that can make complex design tasks easier for people while enabling them to be smarter over time will be essential for truly supporting the next generation of designers. For instance, wouldn't it be great if a photo-editing tool that currently only supports color and sharpness editing, also teaches users about better photo compositions along the way?

For robot design tasks, *Geppetto* took tiny steps towards aiding user learning by explicitly teaching people about the effect of various design parameters on the resultant designs. We also found value in using simulation and model-driven feedback for implicitly teaching people about intrinsic design characteristics as in our structure design tools. In the future however, upcoming immersive technologies such as augmented, virtual, and mixed reality may be leveraged to provide a wide variety of feedback to the users, ultimately supporting the development of design intuition in the users. Finally, there is also value in exploring and developing hardware testbeds that could complement design tools in systematically teaching people about robotic system development. Recent work in the robotics community on developing one such testbed called *Duckietown* for the design of ground robots is an encouraging step in this direction [164].

Bibliography

[1] *Bullet Physics Library*, 2015. http://bulletphysics.org/. 3.7.1, 4.5.2, 5.5.2

[2] *Arduino*, 2017. https://www.arduino.cc/. 1.2, 2

[3] *Make magazine*, 2017. https://makezine.com/category/digital-fabrication/. 1.2

[4] *MyMiniFactory*, 2017. https://www.myminifactory.com/. 4.9

[5] *Sketch-up*, 2017. https://www.sketchup.com/. 2.2, 3.2

[6] *Robot Makers Workshop*, RSS 2016. http://www.seas.upenn.edu/~nicbezzo/RoMa2016/. 2.3

[7] Fusion 360. Autodesk, 2018. https://www.autodesk.com/products/fusion-360/. 2.1

[8] *Adobe Creative Cloud*. Adobe, 2018. https://www.adobe.com/creativecloud.html. 1.1

[9] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 828–837. ACM, 2003. 3.4.2

[10] Kaat Alaerts, Evelien Nackaerts, Pieter Meyns, Stephan P Swinnen, and Nicole Wenderoth. Action and emotion recognition from point light displays: an investigation of gender differences. *PloS one*, 6(6):e20989, 2011. 7.2

[11] *Mechanical Turk*. Amazon, 2017. https://www.mturk.com/mturk/welcome. 7.7.2

[12] Mogens Myrup Andreasen, Steen Kähler, and Thomas Lund. *Design for assembly*. IFS, 1988. 3.4.2

[13] Deepali Aneja, Alex Colburn, Gary Faigin, Linda Shapiro, and Barbara Mones. Modeling stylized character expressions via deep learning. In *Asian Conference on Computer Vision*, pages 136–153. Springer, 2016. 7.3

[14] *Cozmo*. Anki, 2016. https://www.fastcodesign.com/3061276/meet-cozmo-the-pixar-inspired-ai-powered-robot-that-feels. 1.1

[15] Anvel. Anvel Sim, 2018. https://anvelsim.com/what-is-anvel. 2.1

[16] Daniel Ashbrook, Shitao Stan Guo, and Alan Lambie. Towards augmented fabrication: Combining fabricated and existing objects. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1510–1518. ACM,

2016. 3.3

[17] Yves F Atchadé, Gareth O Roberts, and Jeffrey S Rosenthal. Towards optimal scaling of metropolis-coupled markov chain monte carlo. *Statistics and Computing*, 21(4):555–568, 2011. 3.7.2, 3.7.2

[18] Joshua Auerbach, Deniz Aydin, Andrea Maesani, Przemyslaw Kornatowski, Titus Cieslewski, Grgoire Heitz, Pradeep Fernando, Ilya Loshchilov, Ludovic Daler, and Dario Floreano. RoboGen: Robot Generation through Artificial Evolution. In *Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pages 136–137. The MIT Press, 2014. doi: 10.7551/978-0-262-32621-6-ch022. URL `http://www.robogen.org/`. 4.3, 6.3

[19] *Autodesk 123D Design*. Autodesk, 2014. `http://www.123dapp.com/design`. 2.2, 3.2

[20] *Tinkerplay*. Autodesk, 2015. `http://investors.autodesk.com/phoenix.zhtml?c=117861&p=irol-newsArticle&ID=2026270"`. 4.3

[21] *Autodesk Maya*. Autodesk, 2017. `https://www.autodesk.com/products/maya/overview`. 1.1, 7.2

[22] *Tinkercad*. Autodesk, 2018. `https://www.tinkercad.com/`. 2.2

[23] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. Spin-it: optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG)*, 33(4):96, 2014. 3.3

[24] Connelly Barnes, David E Jacobs, Jason Sanders, Dan B Goldman, Szymon Rusinkiewicz, Adam Finkelstein, and Maneesh Agrawala. Video puppetry: a performative interface for cutout animation. In *ACM Transactions on Graphics (TOG)*, volume 27, page 124. ACM, 2008. 7.3

[25] Lyn Bartram and Ai Nakatani. What makes motion meaningful? affective properties of abstract motion. In *Image and Video Technology (PSIVT), 2010 Fourth Pacific-Rim Symposium on*, pages 468–474. IEEE, 2010. 7.3, 7.6.3

[26] Cindy L Bethel and Robin R Murphy. Survey of non-facial/non-verbal affective expressions for appearance-constrained robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(1):83–92, 2008. 1

[27] Nicola Bezzo, Junkil Park, Andrew King, Peter Gebhard, Radoslav Ivanov, and Insup Lee. Demo abstract: Roslaba modular programming environment for robotic applications. In *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, pages 214–214. IEEE, 2014. 2.3

[28] 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. 2.3

[29] Gaurav Bharaj, David IW Levin, James Tompkin, Yun Fei, Hanspeter Pfister, Wojciech Matusik, and Changxi Zheng. Computational design of metallophone contact sounds. *ACM Transactions on Graphics (TOG)*, 34(6):223, 2015. 3.3, 3.7.2

[30] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008. 7.3

[31] *Blender*. Blender Foundation, 2018. `https://www.blender.org/`. 7.2

[32] Geoffrey Boothroyd. *Assembly Automation and Product Design, Second Edition (Manufacturing Engineering and Materials Processing)*. CRC Press, Inc., Boca Raton, FL, USA, 2005. ISBN 1574446436. 3.2, 3.4.2, 3.6

[33] Johannes Braumann and Sigrid Brell-Cokcan. Visual robot programming: linking design, simulation, and fabrication. In *Proceedings of the Symposium on Simulation for Architecture & Urban Design*, page 17. Society for Computer Simulation International, 2014. 2.3

[34] Cynthia Breazeal, Atsuo Takanishi, and Tetsunori Kobayashi. Social robots that interact with people. In *Springer handbook of robotics*, pages 1349–1369. Springer, 2008. 7.2

[35] Erin Buehler, Shaun K Kane, and Amy Hurst. Abc and 3d: opportunities and obstacles to 3d printing in special education environments. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility*, pages 107–114. ACM, 2014. 1, 1.3.2

[36] Bradley Canaday, Samuel Zapolsky, and Evan Drumwright. Interactive, iterative robot design. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1188–1195. IEEE, 2017. 6.3

[37] George Casella and Edward I George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992. 3.7.2

[38] Marco Ceccarelli and Chiara Lanni. A multi-objective optimum design of general 3r manipulators for prescribed workspace limits. *Mechanism and Machine Theory*, 39(2): 119–132, 2004. 5.3

[39] Andrea Censi. A class of co-design problems with cyclic constraints and their solution. *IEEE Robotics and Automation Letters*, 2(1):96–103, 2017. 3.2

[40] Duygu Ceylan, Wilmot Li, Niloy J Mitra, Maneesh Agrawala, and Mark Pauly. Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics (TOG)*, 32(6):186, 2013. 3.3

[41] Shi-Kuo Chang. *Visual languages*. Springer Science & Business Media, 2012. 2.3

[42] Siddhartha Chaudhuri and Vladlen Koltun. Data-driven suggestions for creativity support in 3d modeling. *ACM Transactions on Graphics (TOG)*, 29(6):183, 2010. 1.3.5

[43] Siddhartha Chaudhuri, Evangelos Kalogerakis, Stephen Giguere, and Thomas Funkhouser. Attribit: content creation with semantic attributes. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 193–202. ACM, 2013. 1.3.4, 7.3

[44] Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. On the difficulty of co-optimizing morphology and control in evolved virtual creatures. In *Proceedings of the Artificial Life Conference*, pages 226–234, 2016. 6.3

[45] Choreographe. Aldebaran Robotics, 2018. `http://doc.aldebaran.com/1-14/software/choregraphe/choregraphe_overview.html`. 2.2

[46] Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017. 7.3

[47] Loïc Ciccone, Martin Guay, Maurizio Nitti, and Robert W Sumner. Authoring motion cycles. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 8. ACM, 2017. 7.3

[48] *Scratch for Arduino*. Citilab, 2017. `http://s4a.cat/`. 2.2

[49] Jason Cong and Majid Sarrafzadeh. Incremental physical design. In *Proceedings of the 2000 international symposium on Physical design*, pages 84–92. ACM, 2000. 3.7.2

[50] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel Van De Panne. Locomotion skills for simulated quadrupeds. In *ACM Transactions on Graphics (TOG)*, volume 30, page 59. ACM, 2011. 7.6.2

[51] 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 (TOG)*, 32(4):83, 2013. 3.3

[52] Olivier Coudert, Jason Cong, Sharad Malik, and Majid Sarrafzadeh. Incremental cad. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 236–244. IEEE Press, 2000. 3.7.2

[53] Jim Crenshaw, Majid Sarrafzadeh, Prithviraj Banerjee, and Pradeep Prabhakaran. An incremental floorplanner. In *VLSI, 1999. Proceedings. Ninth Great Lakes Symposium on*, pages 248–251. IEEE, 1999. 3.7.2

[54] Jennifer Cross, Christopher Bartley, Emily Hamner, and Illah Nourbakhsh. A visual robot-programming environment for multidisciplinary education. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 445–452. IEEE, 2013. 2.3

[55] Nigel Cross. Design cognition: Results from protocol and other empirical studies of design activity. 2001. 1.3.5

[56] László Csató. Ranking by pairwise comparisons for swiss-system tournaments. *Central European Journal of Operations Research*, 21(4):783–803, 2013. 7.3, 7.7.2

[57] Geoff Cumming. The new statistics: Why and how. *Psychological science*, 25(1):7–29, 2014. 7.9.4

[58] Geoff Cumming and Sue Finch. Inference by eye: confidence intervals and how to read pictures of data. *American Psychologist*, 60(2):170, 2005. 7.9.4

[59] Sébastien Dalibard, Nadia Magnenat-Talmann, and Daniel Thalmann. Anthropomorphism of artificial agents: a comparative survey of expressive design and motion of virtual characters and social robots. In *Workshop on Autonomous Social Robots and Virtual Humans at the 25th Annual Conference on Computer Animation and Social Agents (CASA 2012)*, 2012. 7.2

[60] Ruta Desai, Ye Yuan, and Stelian Coros. Computational abstractions for interactive design of robotic devices. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1196–1203. IEEE, 2017. 3.3, 4.9, 5.3, 5.4

[61] Ruta Desai, Margarita Safonova, Katharina Muelling, and Stelian Coros. Automatic design of task-specific robotic arms. *arXiv preprint arXiv:1806.07419*, 2018. 5.9

[62] Lego Digital Designer. LEGO, 2018. `https://www.lego.com/en-us/ldd`. 2.2

[63] James P Diprose, Bruce A MacDonald, and John G Hosking. Ruru: A spatial and interactive visual programming language for novice robot programming. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 25–32. IEEE, 2011. 2.3

[64] *Lucky the Dinosaur*. Disney, 2003. `https://disneyparks.disney.go.com/blog/2013/08/today-in-disney-history-lucky-the-dinosaur-walks-on-the-scene/`. 7.6.1

[65] *Geppetto*. Disney, 2018. `http://disney.wikia.com/wiki/Geppetto`. 7.1

[66] Pierre Dragicevic, Fanny Chevalier, and Stephane Huot. Running an hci experiment in multiple parallel universes. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*, pages 607–618. ACM, 2014. 7.9.4

[67] Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. Computational multicopter design. *ACM Transactions on Graphics (TOG)*, 35(6):227, 2016. 2.3, 3.3

[68] Abhimanyu Dubey, Nikhil Naik, Devi Parikh, Ramesh Raskar, and César A Hidalgo. Deep learning the city: Quantifying urban perception at a global scale. In *European Conference on Computer Vision*, pages 196–212. Springer, 2016. 7.3

[69] Magda Dubois, Josep-Arnau Claret, Luis Basañez, and Gentiane Venture. Influence of emotional motions in human-robot interactions. In *International Symposium on Experimental Robotics*, pages 799–808. Springer, 2016. 7.2

[70] *ENABLING THE FUTURE*. E-NABLE, 2017. `http://enablingthefuture.org/`. 1.1

[71] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200, 1992. 7.5, 2

[72] *Sporepedia*. Electronic Arts Inc., 2009. `http://www.spore.com/sporepedia`. 4.3

[73] Victor Emeli. Robot learning through social media crowdsourcing. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2332–2337. IEEE, 2012. 7.3

[74] Pro Engineer. PTC, 2018. `https://www.ptc.com/en/products/cad`. 2.1

[75] *humanitarian supplies made-in-the-field*. FIELD READY, 2017. `https://www.fieldready.org/`. 1.1

[76] Fusion 360 for Students. Autodesk, 2018. `https://www.autodesk.com/products/fusion-360/students-teachers-educators`. 2.1

[77] NAOqi Framework. Aldebaran Robotics, 2018. `http://doc.aldebaran.com/1-14/`

dev/naoqi/index.html. 2.1

[78] Ran Gal, Lior Shapira, Eyal Ofek, and Pushmeet Kohli. Flare: Fast layout for augmented reality applications. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 207–212. IEEE, 2014. 3.3, 3.7, 6

[79] Vittorio Gallese, Christian Keysers, and Giacomo Rizzolatti. A unifying view of the basis of social cognition. *Trends in cognitive sciences*, 8(9):396–403, 2004. 7.2

[80] Madeline Gannon. *Human-Centered Interfaces for Autonomous Fabrication Machines*. PhD thesis, Carnegie Mellon University, 207. 7.6.3

[81] Gazebo. Open Source Robotics Foundation, 2018. http://gazebosim.org/. 2.1

[82] Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. Skater bots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels. *ACM Transactions on Graphics (TOG)*, 2018. 6.9

[83] Charles J Geyer. Markov chain monte carlo maximum likelihood. 1991. 3.2, 3.7.2

[84] Michael B Giles and Niles A Pierce. An introduction to the adjoint approach to design. *Flow, turbulence and combustion*, 65(3-4):393–415, 2000. 6.2, 6.5.4

[85] Oliver Glauser, Wan-Chun Ma, Daniele Panozzo, Alec Jacobson, Otmar Hilliges, and Olga Sorkine-Hornung. Rig animation with a tangible and modular input device. *ACM Transactions on Graphics (TOG)*, 35(4):144, 2016. 7.3

[86] *Do-it-yourself artificial intelligence*. Google, 2018. https://aiyprojects.withgoogle.com/. 1.2

[87] *tensorflow*. Google Inc., 2017. https://www.tensorflow.org/. 7.7.3

[88] Grasshopper. Scott Davison, 2018. http://www.grasshopper3d.com/. 2.2

[89] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Joint optimization of robot design and motion parameters using the implicit function theorem. 6.3

[90] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Computational co-optimization of design parameters and motion trajectories for robotic systems. *The International Journal of Robotics Research*, page 0278364918771172, 2018. 2

[91] Ramsey F Hamade, Hassan A Artail, and Mohamad Y Jaber. Evaluating the learning process of mechanical cad students. *Computers & Education*, 49(3):640–661, 2007. 1, 2.1

[92] John Harris and Ehud Sharlin. Exploring the affect of abstract motion in social human-robot interaction. In *RO-MAN, 2011 IEEE*, pages 441–448. IEEE, 2011. 7.3, 7.6.2

[93] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. 3.7.2

[94] Hebi Robotics, 2017. http://hebirobotics.com/products/. 1.1, 5.2

[95] Chris Hecker, Bernd Raabe, Ryan W Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies.

ACM Trans. on Graphics(TOG), page 27, 2008. 4.3

[96] Jan LM Hensen and Roberto Lamberts. *Building performance simulation for design and operation*. Routledge, 2012. 4.3

[97] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 159–166. ACM, 1999. 1.3.1

[98] Nathaniel Hudson, Celena Alcock, and Parmit K Chilana. Understanding newcomers to 3d printing: Motivations, workflows, and barriers of casual makers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 384–396. ACM, 2016. 1.2, 3

[99] Instructables. *Instructables - How to make anything*, 2017. `https://www.instructables.com/`. 3.8

[100] Inventor. Autodesk, 2018. `https://www.autodesk.com/products/inventor/`. 2.1

[101] Mark Jerrum and Alistair Sinclair. The markov chain monte carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems*, pages 482–520, 1996. 3.7.2

[102] James Jessiman. *LDraw*, 2018. `http://www.ldraw.org/`. 2.2

[103] Pablo Jiménez. Survey on assembly sequencing: a combinatorial and geometrical perspective. *Journal of Intelligent Manufacturing*, 24(2):235–250, 2013. 3.2, 3.3, 3.7

[104] K Jittorntrum. An implicit function theorem. *Journal of Optimization Theory and Applications*, 25(4):575–577, 1978. 6.3, 6.5.3

[105] Michael D Jones, Kevin Seppi, and Dan R Olsen. What you sculpt is what you get: Modeling physical interactive devices with clay and 3d printed widgets. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 876–886. ACM, 2016. 3.3

[106] Malte F Jung. Affective grounding in human-robot interaction. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 263–273. ACM, 2017. 7.2

[107] Sung-Gaun Kim and Jeha Ryu. New dimensionally homogeneous jacobian matrix formulation by three end-effector points for optimal design of parallel manipulators. *IEEE Transactions on Robotics and Automation*, 19(4):731–736, 2003. 5.3

[108] Young J Kim, Miguel A Otaduy, Ming C Lin, and Dinesh Manocha. Fast penetration depth computation for physically-based animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 23–31. ACM, 2002. 3.7.1

[109] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7.7.3

[110] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simmulated annealing. *science*, (4598):671–680, 1983. 3.3, 3.7.2

[111] Heather Knight and Reid Simmons. Laban head-motions convey robot state: A call for robot body language. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 2881–2888. IEEE, 2016. 7.3, 7.6.2

[112] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004. 4.3

[113] Aminata Kone and David A Kofke. Selection of temperature intervals for parallel-tempering simulations. *The Journal of chemical physics*, 122(20):206101, 2005. 3.7.2

[114] Adriana Kovashka, Devi Parikh, and Kristen Grauman. Whittlesearch: Image search with relative attribute feedback. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2973–2980. IEEE, 2012. 1.3.4, 7.3

[115] Yuki Koyama and Masataka Goto. Optimo: Optimization-guided motion editing for keyframe character animation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 161. ACM, 2018. 7.3

[116] Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. Crowd-powered parameter analysis for visual design exploration. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 65–74. ACM, 2014. 7.2, 7.3

[117] *Industrial robots*. KUKA, 2017. `https://www.kuka.com/en-us/products/robotics-systems/industrial-robots`. 5.2, 7.6.1

[118] Ranjitha Kumar and Kristen Vaccaro. An experimentation engine for data-driven fashion systems. 2017. 7.3

[119] Pierre-Yves Laffont, Zhile Ren, Xiaofeng Tao, Chao Qian, and James Hays. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM Transactions on Graphics (TOG)*, 33(4):149, 2014. 7.3

[120] Microsoft Visual Programming Language. Microsoft, 2018. `https://msdn.microsoft.com/en-us/library/bb483088.aspx`. 2.2

[121] Brian Lee, Savil Srivastava, Ranjitha Kumar, Ronen Brafman, and Scott R Klemmer. Designing with interactive example galleries. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2257–2266. ACM, 2010. 7.5.2

[122] Patrick (Chris) Leger. *Automated Synthesis and Optimization of Robot Configurations: An Evolutionary Approach*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1999. 4.3, 6.3

[123] Leopoly. Leopoly, Ltd., 2018. `https://leopoly.com/`. 2.2

[124] Dingzeyu Li, David IW Levin, Wojciech Matusik, Changxi Zheng, Timothy R Langlois, Changxi Zheng, Doug L James, Gaurav Bharaj, David IW Levin, James Tompkin, et al. Acoustic voxels: Computational optimization of modular acoustic filters. *ACM Transactions on Graphics*, 33:2, 2016. 3.3, 3.7.2

[125] Yueh-Hung Lin, Chia-Yang Liu, Hung-Wei Lee, Shwu-Lih Huang, and Tsai-Yen Li. Evaluating emotive character animations created with procedural animation. In *Intelligent*

Virtual Agents, pages 308–315. Springer, 2009. 7.2

[126] Hod Lipson and Jordan B. Pollack. Towards continuously reconfigurable self-designing robotics. In *ICRA*, pages 1761–1766. IEEE, 2000. ISBN 0-7803-5889-9. URL http://dblp.uni-trier.de/db/conf/icra/icra2000.html#LipsonP00. 4.3

[127] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989. 6.5.2

[128] Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. Build-to-last: strength to weight 3d printed objects. *ACM Transactions on Graphics (TOG)*, 33(4):97, 2014. 3.3

[129] Thomas Ludwig, Oliver Stickel, Alexander Boden, and Volkmar Pipek. Towards sociable technologies: an empirical study on designing appropriation infrastructures for 3d printing. In *Proceedings of the 2014 conference on Designing interactive systems*, pages 835–844. ACM, 2014. 2, 1.3.2

[130] Makeblock. *Makeblock electronic modules*, 2015. Available at http://www.makeblock.com/electronic-robot-kit-series-STEM. 3.8

[131] Justin Matejka, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. Dream lens: Exploration and visualization of large-scale generative design datasets. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 369. ACM, 2018. 8.2.1

[132] MATLAB and Simulink for Robotics. Mathworks, 2018. https://www.mathworks.com/solutions/robotics.html. 2.1

[133] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. In *ACM Transactions On Graphics (TOG)*, volume 23, pages 449–456. ACM, 2004. 6.5.4

[134] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG)*, 34(6):216, 2015. 2.3, 3.3, 4.2, 4.3, 4.5, 6.4, 7.6.2, 8

[135] Ankur Mehta, Joseph DelPreto, and Daniela Rus. Integrated codesign of printable robots. In *Journal of Mechanisms and Robotics*, volume 7, 2015. 2.3, 4.3

[136] Ankur M. Mehta and Daniela Rus. An end-to-end system for designing mechanical structures for print-and-fold robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, June 2014. 2.3, 4.3

[137] Ankur M. Mehta, Joseph DelPreto, Benjamin Shaya, and Daniela Rus. Cogeneration of mechanical, electrical, and software designs for printable robots from structural specifications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2014. 2.3, 4.3

[138] Ankur M Mehta, Daniela Rus, Kartik Mohta, Yash Mulgaonkar, Matthew Piccoli, and Vijay Kumar. A scripted printable quadrotor: Rapid design and fabrication of a folded mav. In *Robotics Research*, pages 203–219. Springer, 2016. 2.3

[139] Ankur M Mehta, Joseph DelPreto, Kai Weng Wong, Scott Hamill, Hadas Kress-Gazit, and

Daniela Rus. Robot creation from functional specifications. In *Robotics Research*, pages 631–648. Springer, 2018. 2.3, 3.2

[140] Paul Merrell, Eric Schkufza, and Vladlen Koltun. Computer-generated residential building layouts. In *ACM Transactions on Graphics (TOG)*, volume 29, page 181. ACM, 2010. 3.3

[141] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. In *ACM Transactions on Graphics (TOG)*, volume 30, page 87. ACM, 2011. 3.3, 3.7, 3.11.2, 6

[142] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953. 3.7.2

[143] *TrueSkill*. Microsoft Research, 2017. `http://trueskill.org/`. 7.3, 7.7.2

[144] *Daniela Rus*. MIT CSAIL, 2017. `http://danielarus.csail.mit.edu/index.php/about-daniela-2/research-2/`. 1.1

[145] Modbot, 2017. `https://www.modbot.com/product/`. 1.1, 5.2

[146] Brian K Mok, Stephen Yang, David Sirkin, and Wendy Ju. Empathy: interactions with emotive robotic drawers. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 250–251. ACM, 2014. 7.2

[147] morphi. Inventery, Inc., 2018. `http://www.morphiapp.com/`. 2.2

[148] Przemyslaw Musialski, Thomas Auzinger, Michael Birsak, Michael Wimmer, Leif Kobbelt, and TU Wien. Reduced-order shape optimization using offset surfaces. *ACM Transactions on Graphics (TOG)*, 34(4):102, 2015. 3.3

[149] Nikhil Naik, Jade Philipoom, Ramesh Raskar, and César Hidalgo. Streetscore-predicting the perceived safety of one million streetscapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 779–785, 2014. 7.3

[150] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162, 2011. 3.3

[151] Jorge Nocedal and Stephen J Wright. Numerical optimization. 2006. 3.11.2, 6.5.2, 7.8.2

[152] Illah Reza Nourbakhsh. *Robot Futures*. MIT Press, 2013. 1.1

[153] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. Learning layouts for single-pagegraphic designs. *IEEE transactions on visualization and computer graphics*, 20(8): 1200–1213, 2014. 3.3

[154] Peter O'Donovan, Jānis Lībeks, Aseem Agarwala, and Aaron Hertzmann. Exploratory font selection using crowdsourced attributes. *ACM Transactions on Graphics (TOG)*, 33 (4):92, 2014. 7.3

[155] C.D. Onal, M.T. Tolley, R.J. Wood, and D. Rus. Origami-inspired printed robots. *Mechatronics, IEEE/ASME Transactions on*, PP(99):1–8, 2014. ISSN 1083-4435. doi: 10.1109/TMECH.2014.2369854. 4.3

[156] OpenMETA. MetaMorph Inc., 2018. `https://openmeta.metamorphsoftware.com/why-openmeta/`. 2.1, 8.2.2

[157] OpenRAVE. OpenRAVE, 2018. http://openrave.org/. 2.1

[158] Sarah Osentoski, Christopher Crick, Grayin Jay, and Odest Chadwicke Jenkins. Crowdsourcing for closed loop control. 7.3

[159] Gerhard Pahl and Wolfgang Beitz. *Engineering design: a systematic approach*. Springer Science & Business Media, 2013. 3.4.2, 3.6

[160] Francisco Palacios, Michael R Colonno, Aniket C Aranake, Alejandro Campos, Sean R Copeland, Thomas D Economon, Amrita K Lonkar, Trent W Lukaczyk, Thomas WR Taylor, and Juan J Alonso. Stanford university unstructured (su2): An open-source integrated computational environment for multi-physics simulation and design. *AIAA Paper*, 287: 2013, 2013. 4.3

[161] Wei Pan and Lorenzo Torresani. Unsupervised hierarchical modeling of locomotion styles. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 785–792. ACM, 2009. 7.3

[162] Chris Paredis and Pradeep K Khosla. An approach for mapping kinematic task specifications into a manipulator design. 1991. 5.3

[163] Devi Parikh and Kristen Grauman. Relative attributes. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 503–510. IEEE, 2011. 1.3.4, 7.3

[164] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, et al. Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1497–1504. IEEE, 2017. 8.2.3

[165] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 7.7.3, 7.7.3

[166] KUKA — prc. Johannes and Sigrid @ Association for Robots in Architecture, 2018. http://www.robotsinarchitecture.org/kuka-prc. 2.2

[167] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. Retrofab: A design tool for retrofitting physical interfaces using actuators, sensors and 3d printing. *Proc. of SIGCHI. ACM*, 2016. 3.3

[168] Rohit Ramesh, Richard Lin, Antonio Iannopollo, Alberto Sangiovanni-Vincentelli, Björn Hartmann, and Prabal Dutta. Turning coders into makers: the promise of embedded design generation. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*, page 4. ACM, 2017. 3.2, 8

[169] *Rasberry Pi*. RASPBERRY PI FOUNDATION, 2017. https://www.raspberrypi.org/. 1.2

[170] *Baxter*. Rethink Robotics, 2012. https://spectrum.ieee.org/robotics/industrial-robots/rethink-robotics-baxter-robot-factory-worker. 1.1

[171] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH computer graphics*, 21(4):25–34, 1987. 7.6.3

[172] Craig W Reynolds. Steering behaviors for autonomous characters. In *Game developers conference*, volume 1999, pages 763–782, 1999. 7.6.3

[173] Rhinoceros. Robert McNeel and Associates, 2018. `https://www.rhino3d.com/`. 2.1

[174] Tiago Ribeiro and Ana Paiva. The illusion of robotic life: principles and practices of animation for robots. In *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on*, pages 383–390. IEEE, 2012. 7.2, 7.3

[175] Gareth O Roberts, Andrew Gelman, Walter R Gilks, et al. Weak convergence and optimal scaling of random walk metropolis algorithms. *The annals of applied probability*, 7(1): 110–120, 1997. 3.7.2

[176] *Dynamixel X Series*. Robotis, 2015. `http://www.robotis.us/dynamixel-xm430-w210-r/`. 4.2

[177] Bruce Romney, Cyprien Godard, Michael Goldwasser, G Ramkumar, et al. An efficient system for geometric assembly sequence generation and evaluation. *Computers in Engineering*, pages 699–712, 1995. 3.3

[178] ROS. Open Source Robotics Foundation, 2018. `http://www.ros.org/`. 2.1, 7.2

[179] *Unified Robot Description Format (URDF)*. ROS, 2018. `http://wiki.ros.org/urdf`. 8.2.1

[180] ROSLab. University of Pennsylvania, 2018. `http://www.seas.upenn.edu/~nicbezzo/ROSLab.html`. 2.3

[181] James A Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980. 2

[182] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*, volume 2. 4.5.2, 4.5.2, 4.5.2, 4.5.2, 5.5.1

[183] Martin Saerbeck and Christoph Bartneck. Perception of affect elicited by robot motion. In *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, pages 53–60. IEEE, 2010. 7.3

[184] Valkyrie Savage, Sean Follmer, Jingyi Li, and Björn Hartmann. Makers' marks: Physical markup for designing and fabricating functional objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 103–108. ACM, 2015. 3.3

[185] Frank L Schmidt. Eight common but false objections to the discontinuation of significance testing in the. *What if there were no significance tests?*, page 37, 2013. 7.9.4

[186] Stefan Schneegass, Alireza Sahami Shirazi, Tanja Döring, David Schmid, and Albrecht Schmidt. Natcut: an interactive tangible editor for physical object fabrication. In *Proceedings of the extended abstracts of the 32nd annual ACM conference on Human factors in computing systems*, pages 1441–1446. ACM, 2014. 3.3

[187] A Schug, T Herges, and W Wenzel. All-atom folding of the three-helix hiv accessory

protein with an adaptive parallel tempering method. *Proteins: Structure, Function, and Bioinformatics*, 57(4):792–798, 2004. 3.7.2

[188] Adriana Schulz, Cynthia Sung, Andrew Spielberg, Wei Zhao, Robin Cheng, Eitan Grinspun, Daniela Rus, and Wojciech Matusik. Interactive robogami: An end-to-end system for design of robots with ground locomotion. *The International Journal of Robotics Research*, 36(10):1131–1147, 2017. 2.3

[189] Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*, volume 57, page 61. SciPy society Austin, 2010. 7.8.1

[190] Ana Serrano, Diego Gutierrez, Karol Myszkowski, Hans-Peter Seidel, and Belen Masia. An intuitive control space for material appearance. *ACM Transactions on Graphics (TOG)*, 35(6):186, 2016. 7.3

[191] Ari Shapiro, Yong Cao, and Petros Faloutsos. Style components. In *Proceedings of Graphics Interface 2006*, pages 33–39. Canadian Information Processing Society, 2006. 7.3

[192] Ben Shneiderman. Creativity support tools: Accelerating discovery and innovation. *Communications of the ACM*, 50(12):20–32, 2007. 1.3.5, 3.13

[193] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 15–22, New York, NY, USA, 1994. ACM. ISBN 0-89791-667-0. doi: 10.1145/192161.192167. URL `http://doi.acm.org/10.1145/192161.192167`. 4.3, 6.3

[194] Labview's Robotic Simulator. National Instruments, 2018. `http://www.ni.com/white-paper/14133/en/`. 2.1

[195] Ronit Slyper, Guy Hoffman, and Ariel Shamir. Mirror puppeteering: Animating toy robots in front of a webcam. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 241–248. ACM, 2015. 7.3

[196] Solidworks. Dassault Systemes, 2017. `https://www.solidworks.com`. 2.1, 3.9.1

[197] Sichao Song and Seiji Yamada. Expressing emotions through color, sound, and vibration with an appearance-constrained social robot. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 2–11. ACM, 2017. 7.2

[198] *Aibo*. Sony, 2017. `http://www.sony-aibo.com/`. 7.6.1

[199] Andrew Spielberg, Brandon Araki, Cynthia Sung, Russ Tedrake, and Daniela Rus. Functional co-optimization of articulated robots. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5035–5042. IEEE, 2017. 2.3, 6.3, 6.6

[200] Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Měch. Stress relief: improving structural strength of 3d printable objects. *ACM Transactions on Graphics (TOG)*, 31(4):48, 2012. 3.3

[201] Robot Studio. ABB, 2018. `http://new.abb.com/products/robotics/robotstudio`. 2.1

[202] Robotics Developer Studio. Microsoft, 2018. `https://msdn.microsoft.com/en-us/library/bb648760.aspx`. 2.1

[203] Yuan Sun and S Shyam Sundar. Psychological importance of human agency how self-assembly affects user experience of robots. In *2016 11th ACM/IEEE Intl. Conf. on Human-Robot Interaction (HRI)*, pages 189–196, 2016. 1.3.1

[204] Yuyin Sun and Dieter Fox. Neol: Toward never-ending object learning for robots. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1621–1627. IEEE, 2016. 7.3

[205] Cynthia Sung and Daniela Rus. Foldable joints for foldable robots. In *Journal of Mechanisms and Robotics*, volume 7, 2015. 4.3

[206] Daniel Szafir, Bilge Mutlu, and Terrence Fong. Communication of intent in assistive free flyers. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 358–365. ACM, 2014. 7.3

[207] Janos Sztipanovits, Ted Bapty, Sandeep Neema, Larry Howard, and Ethan Jackson. Openmeta: a model-and component-based design tool chain for cyber-physical systems. In *Joint European Conferences on Theory and Practice of Software*, pages 235–248. Springer, 2014. 8.2.2

[208] Janos Sztipanovits, Ted Bapty, Sandeep Neema, Xenofon Koutsoukos, and Ethan Jackson. Design tool chain for cyber-physical systems: lessons learned. In *Proceedings of the 52nd Annual Design Automation Conference*, page 81. ACM, 2015. 8.2.2

[209] Leila Takayama, Doug Dooley, and Wendy Ju. Expressing thought: improving robot readability with animation principles. In *Proceedings of the 6th international conference on Human-robot interaction*, pages 69–76. ACM, 2011. 7.3

[210] Haodan Tan, John Tiab, Selma Šabanović, and Kasper Hornbæk. Happy moves, sad grooves: Using theories of biological motion and affect to design shape-changing interfaces. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pages 1282–1293. ACM, 2016. 7.3, 7.6.2

[211] *The do-it-yourself revolution*. TEDx, 2017. `http://www.tedxthessaloniki.com/talk_video/the-do-it-yourself-revolution/`. 1.1

[212] Michael Terry and Elizabeth D Mynatt. Side views: persistent, on-demand previews for open-ended tasks. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 71–80. ACM, 2002. 1.3.5

[213] Michael Terry, Elizabeth D Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. Variation in element and action: supporting simultaneous development of alternative solutions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 711–718. ACM, 2004. 1.3.5

[214] *Fear and Love Exhibit*. the DESIGN MUSEUM, 2017. `https://designmuseum.org/exhibitions/fear-and-love`. 7.6.1, 6

[215] Thingiverse. *Thingiverse - Digital Designs for Physical Objects*, 2017. `https://www.thingiverse.com/`. 3.8

[216] Frank Thomas, Ollie Johnston, and Frank. Thomas. *The illusion of life: Disney animation*. Hyperion New York, 1995. 7.3, 6

[217] NETLab Toolkit. Philip Van Allen, 2018. `http://www.netlabtoolkit.org/`. 2.2

[218] Virtual Robotics Toolkit. Cogmation robotics, 2018. `http://www.coppeliarobotics.com/index.html`. 2.2

[219] *6 axis robots*. Universal Robots, 2017. `https://www.universal-robots.com/`. 5.2

[220] V-REP. Coppelia robotics, 2018. `http://www.coppeliarobotics.com/index.html`. 2.1

[221] Gino Van Den Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference*, volume 170, 2001. 3.7.1

[222] Lauren Vasey, Tovi Grossman, Heather Kerrick, and Danil Nagy. The hive: a human and robot collaborative building process. In *ACM SIGGRAPH 2016 Talks*, page 83. ACM, 2016. 7.6.1

[223] Marynel Vázquez, Eric Brockmeyer, Ruta Desai, Chris Harrison, and Scott E Hudson. 3d printing pneumatic device controls with variable activation force capabilities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1295–1304. ACM, 2015. 3.2

[224] Gentiane Venture, Hideki Kadone, Tianxiang Zhang, Julie Grèzes, Alain Berthoz, and Halim Hicheur. Recognizing emotions conveyed by human gait. *International Journal of Social Robotics*, 6(4):621–632, 2014. 7.2

[225] *SnapCAD*. VEX-Robotics, 2018. `https://www.vexrobotics.com/vexiq/resources/cad-snapcad`. 2.2, 4.3

[226] Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. . net gadgeteer: a platform for custom devices. In *Pervasive Computing*, pages 216–233. Springer, 2012. 3.3

[227] Voxel8. *Voxel8: 3D Electronics Printing*, 2015. `http://www.voxel8.co`. 3.2

[228] Jue Wang, Steven M Drucker, Maneesh Agrawala, and Michael F Cohen. The cartoon animation filter. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 1169–1173. ACM, 2006. 7.3

[229] Webots. Cyberbotics, 2018. `https://www.cyberbotics.com/#webots`. 2.1

[230] Christian Weichel, Manfred Lau, and Hans Gellersen. Enclosed: a component-centric interface for designing prototype enclosures. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*, pages 215–218. ACM, 2013. 3.3, 12

[231] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005. 7.8.1

[232] *The Free Encyclopedia*. Wikipedia, 2017. `https://en.wikipedia.org/`. 3, 4, 7.7.3

[233] Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. Printed optics: 3d printing of embedded optical elements for interactive devices. In *Proceedings of the 25th an-

nual ACM symposium on User interface software and technology, pages 589–598. ACM, 2012. 3.2

[234] Randall H Wilson. On geometric assembly planning. Technical report, DTIC Document, 1992. 3.3, 3.7

[235] Jan D Wolter. On the automatic generation of assembly plans. In *Computer-Aided Mechanical Assembly Planning*, pages 263–288. Springer, 1991. 3.3

[236] Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D Goodman, and Pat Hanrahan. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Transactions on Graphics (TOG)*, 31(4):56, 2012. 3.3

[237] Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30, no. 4, July 2011, article no. 86*, 2011. 3.3, 3.7, 6

[238] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. A survey of crowdsourcing systems. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 766–773. IEEE, 2011. 7.3

[239] Mehmet Ersin Yumer, Siddhartha Chaudhuri, Jessica K Hodgins, and Levent Burak Kara. Semantic shape editing using deformation handles. *ACM Transactions on Graphics (TOG)*, 34(4):86, 2015. 1.3.4, 7.2, 7.3, 7.7.2, 7.7.2

[240] Allan Zhou, Dylan Hadfield-Menell, Anusha Nagabandi, and Anca D Dragan. Expressive robot motion timing. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 22–31. ACM, 2017. 7.3, 7.6.3