Crafting Chaos:

Computational Design of Contraptions with Complex Behavior

Robin Roussel

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

October 7, 2020

I, Robin Roussel, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

The 2010s saw the democratization of digital fabrication technologies. Although this phenomenon made fabrication more accessible, physical assemblies displaying a complex behavior are still difficult to design. While many methods support the creation of complex shapes and assemblies, managing a complex behavior is often assumed to be a tedious aspect of the design process. As a result, the complex parts of the behavior are either deemed negligible (when possible) or managed directly by the software, without offering much fine-grained user control.

This thesis argues that efficient methods can support designers seeking complex behaviors by increasing their level of control over these behaviors. To demonstrate this, I study two types of artistic devices that are particularly challenging to design: drawing machines, and chain reaction contraptions. These artifacts' complex behavior can change dramatically even as their components are moved by a small amount. The first case study aims to facilitate the exploration and progressive refinement of complex patterns generated by drawing machines under drawing-level user-defined constraints. The approach was evaluated with a user study, and several machines drawing the expected pattern were fabricated. In the second case study, I propose an algorithm to optimize the layout of complex chain reaction contraptions described by a causal graph of events in order to make them robust to uncertainty. Several machines optimized with this method were successfully assembled and run.

This thesis makes the following contributions: (1) support complex behavior specifications; (2) enable users to easily explore design variations that respect these specifications; and (3) optimize the layout of a physical assembly to maximize the probability of real-life success.

Impact statement

This thesis makes two contributions in the field of computational design. Both aim to support the creation of physical assemblies that present an intentionally complex behavior. This focus on user-controlled behavior complexity is largely unexplored and opens promising new research directions, both in computer graphics and human-computer interaction.

The first contribution is a novel method providing a form of guided exploration inside the space of motion patterns produced by a mechanical assembly. This work continues previous efforts in the field to make mechanical design accessible to novice users. It could be integrated into computer-aided design software to provide new intuitive modes of interaction. Examples of applications that would benefit from this approach include toys, assembly line tools and other machines that perform a repetitive complex motion.

The second contribution is a novel method to increase the robustness of physical assemblies whose behavior can be formulated as a causal graph of events. Design and fabrication errors are a significant source of frustration, as well as wasted time and material. Integrating ways to accommodate errors into design tools can significantly help users (both novice and expert) reduce the number of product iterations. This is particularly relevant to situations where the context of use of an assembly is highly variable, for instance when repairing or augmenting existing artifacts (which can be critical when resources are scarce, e.g., because of environmental concerns or during humanitarian crises), or when designing for children or people with disabilities.

Acknowledgements

First of all, I am indebted to my academic advisors, without whom this thesis would never have been possible: Professors Niloy Mitra, Marie-Paule Cani and Jean-Claude Léon. They have taught me countless things, constantly pushed me to challenge myself, allowed me to be part of a vibrant community, and made me a better researcher.

I would also like to thank all the people who have helped me during these years, either through thoughtful advice, support, companionship, or by contributing to a stimulating and friendly work environment. Starting chronologically with my teammates and colleagues at Inria Grenoble (and later, at LIX): thanks to Aarohi Johal, Antoine Begault, Camille Schreck, Damien Rohmer, Estelle Charleroy, Even Entem, Geoffrey Guingo, Grégoire Nieto, Guillaume Cordonnier, Maxime Garcia, Pauline Olivier, Pierre Ecormier-Nocca, Pierre-Luc Manteaux, Rémi Ronfard, Stefanie Hahmann, Thomas Buffet, Tibor Stanko, Ulysse Vimont and Youna Le Vaou. Then, continuing with my labmates at UCL: thanks to Aron Monszpart, Carlo Innamorati, Dan Koschier, Eric-Tuan Le, James Hennessey, Lingjie Liu, Mohamed Sayed, Moos Hueting, Paul Guerrero, Philipp Henzler, Pradyumna Reddy, Stratos Skordos, Tom Kelly, Tuanfeng Yang Wang and Yu-Shiang Wong. Additionally, I am grateful to Professors Tobias Ritschel and Simon Julier for their feedback and advice, and to Sarah Turnbull and Dawn Bailey for their incredible job on the administrative side.

I also had the chance to do an internship at Adobe's Creative Intelligence Lab, under the supervision of Amanda Ghassaei and Matt Fisher. Although the research conducted there did not end up in this thesis, I am incredibly grateful to Amanda for these few months in a different and exciting environment. Thanks also to the Adobe

labmates I was fortunate to spend time with: Emma Frid, Germán Leiva, Karren Yang, Megha Nawhal, Nathan Keil, Savvas Petridis and Stefanie Claudino Daffara.

Lastly, on a personal note, I would like to thank my parents for their unwavering support; Alex Sin, who has been my rock during these years, always reminding me to be kind to myself; and Sean Cham, for his tremendous help and support during the writing of this thesis.

The work presented in this thesis was funded by the ERC Starting Grant Smart-Geometry (335373), the ERC Advanced Grant Expressive (291184), the ERC PoC Grant SemanticCity (825706), a Google Faculty Award, a Royal Society Advanced Newton Fellowship, and gifts from Adobe.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem

The 2010s saw the democratization of digital fabrication technologies. Computer-controlled devices such as 3D printers and laser cutters significantly simplified previous fabrication workflows, opening new possibilities for both novice and expert users. Designing fabricable and functional objects, however, remains a challenging task. *Computational design* aims to address this problem by augmenting the design capabilities of end users with computational assistance. To this end, researchers have identified various roadblocks and sources of frustration in existing design workflows, and have proposed new methods to help users achieve a target appearance or functionality. For instance, researchers have tackled problems involving a high number of assembly components (e.g., in puzzles [132]) and a high level of coupling (e.g., in mechanical assemblies [26]), as well as behaviors with a complex dependency on shape and material distribution (e.g., spinning [4] and sound [13]).

Though many such methods offer considerable control over the geometry of the product, physical behavior specifications are usually more limited. Design interfaces that offer some form of behavior control often only support simple inputs, such as motion paths [26] or sparse target assembly poses [93]. This is because managing a complex physical behavior is often assumed to be a tedious task. As a result, complex physical phenomena are either deemed negligible (when possible) or directly managed by the software, without offering much fine-grained control to

the user. Although some systems provide more interactive forms of control, such as *guided exploration* [105], behavior constraints are nowhere near as common as their geometric counterpart. For instance, in computer-aided design (CAD) software, although the features of a shape representation can be constrained in various ways (e.g., contact and orthogonality), no existing method allows users to similarly constrain the features of a simulated behavior representation. Additionally, very few systems improve physical control by accommodating unpredictable factors such as fabrication errors [137], and none of them supports complex physical behaviors.

In this thesis, I argue that efficient methods can support designers seeking complex behaviors by *increasing* their level of control over these behaviors. To support this claim, I present two case studies in which individuals aim to design complex machines for artistic or entertainment purposes: drawing machines and chain reaction contraptions. I chose to investigate these specific design problems for three reasons. First, they are prime examples of intentional behavior complexity. Although their outcomes differ (static drawing in one case; audiovisual performance in the other), both rely on the complex behavior of a machine to captivate the audience, and both are difficult to control because small changes in the initial conditions can result in large and hard-to-predict behavior variations.[1] Second, solving these design problems is relevant because drawing machines and chain reaction contraptions are popular among various communities of makers, video creators, educators, and artists. For example, at the time of writing, the digital model sharing platform Thingiverse[2] contains dozens of drawing machines and Spirographs; the most popular chain reaction builders on the video sharing platform YouTube[3] (i.e., Hevesh5, Joseph's Machines and DoodleChaos) each total hundreds of millions of views; at least three chain reaction contraption student contests take place every year in the USA;[4] and contemporary artists (such as Olafur Eliasson [72], Arthur Ganson [10] and James Nolan Gandy [128]) are still creating pieces involving

[1] Which is why I informally describe these machines as being "chaotic" from a user perspective.

[2] `https://www.thingiverse.com/`

[3] `https://www.youtube.com/`

[4] The Rube Goldberg Machine Contest (Purdue University), the Friday After Thanksgiving Chain Reaction (MIT Museum), and the Chain Reaction Contraption Contest (Carnegie Science Center).

either type of machine. Lastly, although aiming for complexity at the expense of efficiency seemingly runs counter to the philosophy of engineering, the inventiveness of these devices can still be a source of inspiration for more practical tools. For instance, traditional Japanese automata known as *karakuri puppets* have inspired the practice of *karakuri kaizen* in assembly lines, a form of low cost automation in which workers are assisted by chain reaction devices without any reliance on electrical power [139]. These contraptions are robust to power cuts and reduce the energy and environmental cost of factories [64].

To summarize, this thesis describes efficient computational design methods to help users create physical assemblies displaying an intentionally complex physical behavior, by providing a better level of control over this behavior, with a focus on machines designed with an artistic intent.

1.2 Definitions

In this section, I define and clarify the terms used throughout the thesis. The order matters: except for the last two, each definition builds upon the previous ones.

Physical assembly. A physical assembly is a real-world artifact composed of parts arranged in a specific configuration to achieve a purpose. The parts may be interconnected with various types of joints, or simply in contact or close proximity.

Virtual assembly. A virtual assembly is a representation of a physical assembly used in CAD software. A virtual assembly abstracts and approximates its physical counterpart in various ways. For instance, the geometry may be discretized as a 3D mesh. Additionally, the virtual assembly may include components that already exist in the world and have been measured or 3D scanned to build the virtual model.

Physical variables. Physical variables fluctuate in the (simulated) physical world. Examples include location, velocity, force, temperature and pressure.

Constructive variables. Constructive variables fluctuate in the design environment. Examples include length, angle, relative position and orientation, as well as nominal values of physical variables. Constructive variables that can be directly controlled are also called “design parameters” (see Control).

Behavior. The behavior of a design characterizes how input variables are mapped to output variables. Behavior can be *physical* or *constructive*.

Physical behavior refers to the relation between physical variables. For instance, it describes how an input rotation is converted into the motion path of an end-effector in a mechanical assembly. This behavior is governed by the laws of physics (including causality) and, in the case of electro-mechanical devices, by the software controlling the actuators. It can be simulated in a virtual environment, up to a certain accuracy. It is then called "simulated behavior".

Constructive behavior refers to the relation between constructive variables. For instance, it describes how changing a dimension of a part requires changing its position in order to avoid interpenetration with another part. This behavior is governed by the parametrization of the model, as well as the objectives and constraints enforced by the system.

Critically, both types of behavior are interdependent: physical behavior depends on design parameters, while constructive behavior may involve constraints and objectives based on physical considerations. However, while physical and constructive variables may be viewed as part of the same high-dimensional space [50], they do not usually change at the same time.

Further, not all aspects of the behavior are equally important to the end user. Just like they may care about the outer appearance of an object but not its internal structure, they may also care about the motion of an end-effector but not of some intermediate part. Any aspect that is important to the user is called a "behavior of interest".

Control. Control is the ability to *purposefully* set, change or constrain variables or behaviors. Purpose matters: exposing variables that users do not understand does not yield control. Control can be *physical* or *constructive*.

Physical control is the ability to obtain the desired physical assembly or behavior. It depends on several factors. First, different fabrication methods require different practical skills and physical abilities. Computer numerical control (CNC) machines have been successful notably because their requirements are relatively low. Second, physical control depends on the sensitivity of the artifact's physical behavior to

approximations and external perturbations. For instance, measurement errors may prevent design augmentations to correctly fit a target object. Similarly, physically-based simulations are often deemed accurate enough to ensure physical control of the behavior of interest, but the case study in Chapter 4 shows that additional computational assistance may be needed. On the other hand, mechanical assemblies and robots are typically expected to offer a high degree of physical behavior control.
Constructive control is the ability to obtain the desired virtual assembly or simulated behavior in the design environment. The level of constructive control, as perceived by the end user, notably depends on the ability of the system to match their skill and needs. For instance, automated systems may expose partial or simplified constructive controls to help novice users, but make advanced users feel constrained.

Complexity. Complexity describes the time or effort required to achieve the desired result *as a function* of the number of variables and their interactions. Complexity can be *physical* or *constructive*.[5]
Physical complexity is measured in time or effort spent fabricating and operating the artifact. It is a function of the artifact's *probability* of successfully achieving target specifications, which depends on the degree of physical control.
Constructive complexity is measured in time or effort spent by the end user designing the artifact. It is a function of the representation and parametrization of the artifact and its simulated behavior, as well as the designer's technical skills and cognitive abilities (including working memory). While many shape modeling tools support the creation of intricate shapes and structures, most computational design methods assume that complex behavior specifications are a source of frustration or a limitation (especially for novice users). Based on this assumption, a common strategy to reduce constructive complexity is to hide part of the artifact and behavior from the end user, thus exposing fewer design parameters, while automatically managing the other constructive variables.

Computational design. Computational design refers to the use of computation to support or carry out design tasks. Compared to CAD paradigms, design parameters

[5] A related distinction has been made between "functional" and "structural" complexity in the literature [15].

can be directly controlled by an algorithm to achieve the best performance under constraints. In particular, a computational design method is said to be "fabrication-aware" when fabricability objectives or constraints are taken into account. The works presented in this thesis are instances of computational design.

Computational fabrication. Computational fabrication refers to the use of computation to support or carry out the fabrication process. This kind of approach typically involves digital fabrication technologies such as 3D printers and laser cutters. The current literature remains ambiguous about the exact difference between "computational fabrication" and "fabrication-aware computational design". A possible distinction could be that computational fabrication methods focus more on taking advantage (or working around the limitations) of a specific fabrication technology.

1.3 Contributions

This thesis presents two case studies in which computational assistance improves designers' level of control over the intentionally complex physical behavior of a physical assembly. I conducted this research with the help and advice of the co-authors indicated in the references below.[6] In the first study, the space of feasible drawing machine designs is so constrained that it is difficult to predict what physical behavior can be achieved. I present a novel method for interactive guided exploration directly in behavior space. In the second study, errors and approximations have a significant impact on the behavior of chain reaction contraptions. I present a novel algorithm to increase the robustness of the design, and therefore improve physical behavior control. The main contributions can be summarized as follows.

(C1) Representation: support new types of complex physical behavior specifications.

(C2) Exploration: introduce a new method enabling users to easily explore design variations that respect their physical behavior specifications.

[6] In this thesis, the first person "I" refers to my own claims, as well as my personal work in terms of analysis and presentation. The collective "we/our" either refers to the work I have conducted with other people, or holds the impersonal role commonly found in academic writing.

(C3) Optimization: propose a new method to minimize the chance of real-life deviation from physical behavior specifications.

These technical contributions and much of the text in this thesis were published in three peer-reviewed articles:

- Robin Roussel, Marie-Paule Cani, Jean-Claude Léon, and Niloy J. Mitra. SPIROU: constrained exploration for mechanical motion design. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*, SCF '17, pages 7:1–7:11, New York, NY, USA, 2017. ACM [111]

- Robin Roussel, Marie-Paule Cani, Jean-Claude Léon, and Niloy J. Mitra. Exploratory design of mechanical devices with motion constraints. *Computers & Graphics*, 74:244–256, 2018 [112]

- Robin Roussel, Marie-Paule Cani, Jean-Claude Léon, and Niloy J. Mitra. Designing chain reaction contraptions from causal graphs. *ACM Transactions on Graphics*, 38(4):43:1–43:13, 2019 [113]

1.4 Organization

The remainder of this thesis is divided into 4 chapters.

Chapter 2 analyzes related works through the lens of interactions and methods giving users various levels of control over assemblies and their complex behavior.

In Chapter 3, the goal is to enable users to easily explore and fine-tune the variety of patterns produced by drawing machines. I describe a novel drawing-centric method that provides an enhanced representation (C1) enabling the user to select feature points (such as intersection points) and prescribe geometric specifications on them (such as: staying at the same position). These visual preferences, along with the feasibility constraints of the mechanism, are combined to let users interactively explore a range of desirable valid patterns (C2). Additionally, several examples have been fabricated to validate the method. This work has been presented at SCF 2017 and published as the first and second papers mentioned above [111, 112].

Chapter 4 describes the second case study: generalizing from continuous motion to a wider range of behaviors, I present a method to ease the design of large-scale chain reaction contraptions. While the physical behavior considered in the previous case study was a single trajectory (albeit complex) performed by a mechanical assembly, here a disconnected collection of objects roll, topple, hit or fall into each other. This behavior is described by a causal graph of events (C1) that specifies the order in which events are expected to happen, including complex synchronizations between parallel branches. The contraption layout is optimized to maximize robustness (C3) to the various sources of uncertainty introduced in the process (including modeling approximations and human assembly errors). Several complex examples have been realized (and successfully run) to validate the method. This work has been presented at SIGGRAPH 2019 and published as the last paper mentioned above [113].

In Chapter 5, I summarize the case studies and discuss three main directions for future work.

Chapter 2

Related work

In this thesis, I investigate computational design methods that increase user control over the complex behavior of assemblies. To provide context, I analyze three types of computational assistance that aim to reduce constructive complexity: design assisted by behavior previsualization (Section 2.2), guided design space exploration (Section 2.3), and inverse design in the specific case of moving assemblies (Section 2.4). Each type offers various modes of *interaction* and levels of *constructive control* over the assembly, its simulated behavior, or both. While some of these methods also aim to increase *physical control* to some extent, I describe the main advances in this direction in a section dedicated to the relatively new problem of accommodating uncertainty in computational design and fabrication (Section 2.5).

Sections 2.2–2.5 show how each type of approach addresses specific sources of complexity (physical or constructive, at assembly or behavior level), but may be limited by other sources. This analysis, which contextualizes the general problem stated in Section 1.1, is prefaced with a description of the classification methods used and insights found in previous surveys (Section 2.1). Moreover, although the case studies in this thesis investigate different aspects of the general problem statement, each study adds its own specific challenges and contributions. Since comparisons are easier with a full context, I discuss in detail the papers most relevant to each study in a dedicated in-chapter section titled “Closely related works” (respectively Sections 3.2 and 4.2).

2.1 Previous surveys

The survey presented in this chapter analyzes works through the lens of *user interaction* and *control*. By contrast, previous surveys in computational design have analyzed the literature in terms of *applications* and *methods*. While early reports have focused on specific topics such as material appearance [57] and topology optimization [32], the increasing breadth of research directions has allowed later surveys to adopt a wider scope.

Medeiros e Sá et al. [91] proposed an analysis of *functional fabrication* technologies, which they define as the design and manufacture of physical objects with functionalities exploiting the capabilities of digital fabrication technologies. Works are clustered according to the intended physical behavior of the objects: articulation, elastic deformation, structural stability, balance, aerodynamics, appearance and acoustics. Among other insights, the survey aptly observes the tension between two concurrent research goals: (i) letting users explore the design space freely without restricting them to prescribed shapes, and (ii) letting the system control the design parameters to ensure a prescribed functionality.

Bermano et al. [8] later took a similar but more general perspective: *fabrication-aware design*. While strongly motivated by the challenges of digital fabrication, this topic is broader in scope and includes alternative manufacturing contexts such as rod structures, wire sheets and architectural structures. The authors first note that the capabilities of digital fabrication technologies exceed what human designers are able to understand and specify exhaustively, a challenge that calls for new design tools supporting partial specifications. Their classification is based on two axes: (i) goal of the fabrication process or artifact usage, and (ii) shape and attribute representations used by each method. The first axis divides works into several types of objectives: appearance, deformation and motion, high-level objectives, domain-specific objectives and process-specific objectives. One limitation of this grouping is that some of the categories, such as "high-level" and "domain-specific" objectives, are not precisely defined and difficult to distinguish from the other categories. This does not, however, invalidate the general insight of the survey: different tasks usually

require different geometric and physical representations.

Bickel et al. [14] adopted a different angle: *stylized fabrication* methods. This term refers to design and manufacturing methods used to create an abstract or stylized representation of a digital shape. The survey classifies techniques according to the different phenomena that are abstracted, modified or considered when generating a stylized physical object: shapes, materials, lighting and shadow, decompositions, and "printing the unprintable". Among other observations, it is noted that finding the trade-off between automation and artistic control is particularly important in this setting. Moreover, as many of these designs involve a complex manual assembly sequence, the presented works can significantly benefit from intuitive assembly instructions and error accommodation techniques.

Some other surveys, while not directly dedicated to computational design and fabrication, are still relevant because they explore the notion of *functionality* of a given artifact, which has a significant importance in design. One of the first ideas explored in this regard is that the *structure* of an object (i.e., the arrangement and relations of its parts) correlates strongly with its functionality, and therefore that analyzing and preserving structure automatically may in turn help analyze and preserve functionality. This relationship is notably mentioned in an in-depth analysis of structure-aware shape processing by Mitra et al. [96]. Functionality inferred from geometry and structure is also used in data-driven analysis and processing of shape collections, as shown in a survey by Xu et al. [152]. Lastly, Hu et al. focus entirely on functionality representations for shape analysis [56], going further than previous surveys by considering *interactions* besides geometry.[1]

[1]While the survey proposes its own definition of functionality, it is worth noting that several frameworks have already been proposed in design science literature to analyze how *function* relates to specific attributes [53], such as *behavior* and *structure* [46]. Among other differences, the design science view of behavior considers responses to stimuli independently from a specific source, whereas Hu et al. only consider interactions between entities. This is not surprising since in shape analysis, dynamic data is captured from real life or simulations, in which shapes are already fully formed and interacting with each other. On the other hand, in design, the abstract behavior can be considered separately from the function (which involves a context of use). This suggests that the applicability of shape analysis frameworks to computational design may depend on their ability to infer a behavior model that is independent from context.

2.2 Design assisted by behavior previsualization

Traditional 3D design interfaces provide constructive control over the entire structure and shape parameters of the artifact. At this level of control, computational assistance typically consists in providing intuitive shape modeling tools, simulation feedback, or a combination of both. Shape modeling is a rich area of research that is outside the scope of this thesis, although some of the contributions mentioned here do belong to this domain [89, 115]. Simulations allow designers to analyze and understand how components interact and how their parameters influence the physical behavior of the artifact. It is a staple of CAD software: to cite a few commercial examples, SolidWorks, Autodesk Inventor and Solid Edge all provide motion and structural analysis modules. It is common, however, for this type of software to require a high degree of user expertise, and to run simulations below interactive speeds, which limits the number of design iterations. Researchers have investigated solutions to both problems: (i) integrating simulation visualization in intuitive shape modeling interfaces, and (ii) finding ways to visualize simulated behaviors at interactive speed without sacrificing too much accuracy.

Shape modeling with simulation feedback. In shape modeling methods, the main source of constructive complexity is assumed to be the geometry and structure of the artifact. Masry and Lipson [89] proposed a sketch-based interface where 3D objects are reconstructed from a 2D input, before applying finite element analysis to visualize deformation under a given load. Saul et al. [115] also used sketches, to design chairs that can be fabricated from laser-cut sheet materials. The system allows users to test the design by showing human models of various sizes sitting in the chair to check its ergonomics, and running a fast rigid-body simulation to check its balance. Schulz et al. [116] took a data-driven shape modeling approach where users pick components from a preexisting collection. While these components are automatically aligned and connected to each other, the user can still freely change their shape parameters to personalize the designs. Simulation is used to highlight any unstable part of the assembly. Garg et al. [44] described an interface to design reconfigurable assemblies which looks similar to traditional animation software

(featuring a timeline, keyframes, etc.). These familiar controls are augmented with a useful collision detection indicated both in the timeline and on the 3D model. Ion et al. [58] took advantage of additive manufacturing to invent a new type of machine: metamaterial mechanisms. These cell-based structures are printed as a single entity and are able to reproduce the behavior of more complex mechanical assemblies such as pliers, door latches and switches. An intuitive interface allows the user to fill a 3D grid with the different types of cells, and visualize the object's behavior simulated by a finite element solver. Lastly, Oh et al. [101] implemented a CAD system for exploratory design and fabrication of mechanical papercraft. As in the method by Schulz et al. [116], novice users choose mechanisms from a dataset and customize the various parts. The difference lies in the fact that their system simulates mechanical motion. Interestingly, while a number of inverse motion design systems have been published before this paper [157, 26] (described in Section 2.4), the authors emphasize the importance of avoiding too much automation to allow experimental open-ended design and let users understand mechanical interactions.

Visualizing simulated behavior. Compared to the issue of shape complexity tackled in the previous paragraph, here the artifact's physical behavior is difficult to either simulate or visualize all at once, and needs to be displayed quickly in an intuitive manner. Although not technically part of a design interface, the method by Mitra et al. [97] to create "how things works" visualizations of mechanical assemblies is particularly useful to help users understand how parts move and relate to each other. Moreover, this motion analysis is performed automatically on the 3D model without the need, common in CAD software, to tediously set up all motion constraints. In the specific context of kinetic art, Furuta et al. [42] proposed intuitive ways to visualize the behavior of interacting rigid bodies to help design mobile sculptures and chain reaction contraptions. As this work is particularly close to our own, I provide a more thorough analysis and comparison in Section 4.2. Another increasingly common way to obtain interactive behavior visualization is to generate experimental data *offline*, i.e., before the interactive session, to build a model of the behavior as a function of the various design parameters. These experiments can be manual [144, 86, 98, 20, 41],

or rely on accurate simulations [126, 119, 118, 142], which allows to generate larger amounts of data in parallel. Two main directions have been explored to build the behavior model: locally interpolating between sample points in design space stored in efficient data structures (such a spatial trees), or globally regressing the behavior function over the design space (e.g., using neural networks). Schulz et al. [119] implemented the former type of method to interactively visualize various physical phenomena (stress, deformation, heat distribution) within a CAD interface. The latter type of method was used in a series of works centered on the design of free-form aerodynamic objects, starting with Umetani et al. [144] on paper planes, and followed by Martin et al. [86] on kites, Nakamura et al. [98] on bamboo-copters and Fukusato et al. [41] on boomerangs. While these works aimed to display the motion of flying objects interactively, Umetani and Bickel [142] parametrized 3D shapes as PolyCubes to learn and display specific aerodynamic properties, such as the surface pressure field, velocity stream lines and drag coefficient. Importantly, the speed gained by evaluating a (usually smooth) behavior function instead of running a simulation is often exploited not only for interactive visualization, but also to provide additional forms of assistance, such as guided design exploration [126, 118] (described in Section 2.3) or even design optimization [144, 98, 119, 20, 41] (described in Section 2.4). Both case studies presented in this thesis rely on the smooth approximation of a behavior function.

With the assistance of behavior previsualization, direct constructive control over the geometry gives users a more intimate understanding of *why* a given design works, instead of asking them to trust a solution coming out of a black box. This comprehension can make users more adaptable when faced with unexpected factors that jeopardize the functionality of the object (e.g., fabrication defaults), rather than having to redesign the object. However, this point assumes that the simulation is accurate enough to account for the various sources of physical complexity. In Chapter 4, I present a method that improves physical control over assemblies undergoing complex sequences of events. Additionally, more automated systems shine when user understanding is unnecessary or not even possible. When direct constructive

control becomes too tedious (e.g., because of fabricability constraints, or unpredictable behavior), more computational guidance needs to be provided to keep the design experience productive.

2.3 Guided design space exploration

The need for guided design methods arises when two conditions are met: (i) some aspect of the artifact is deemed too complex for users to control or predict entirely (structure, physical behavior, fabricability constraints, or any relation between them), and (ii) the *value* of a specific design is difficult to quantify, and better left to the judgment of the end user.[2] Regarding the latter condition, the textbook example in computer graphics is the assessment of aesthetic qualities [136, 153, 35, 34, 105]; alternatively, it could be unclear what users value the most about a design [118] (e.g., mass, durability, stability etc.). Guided exploration methods allow users to explore the subset of designs that respect constraints. Therefore, whenever they make a change, they do not have to wonder whether this new configuration is feasible. In other words, they trade a small amount of constructive control for a significant reduction of constructive complexity. Depending on the nature of the interaction, guided exploration methods fall into three categories: (i) discrete suggestions, (ii) subspace-based and (iii) model-based exploration.

Discrete suggestions. Such interfaces are typically the least intrusive as they still offer traditional shape modeling tools, while providing alternative solutions or suggesting edits to help achieve the desired behavior. Umetani et al. [143] proposed a furniture modeling system that automatically suggests alternative designs when dysfunctional configurations are detected, in order to ensure the stability and durability of the assemblies. Thomaszewski et al. [138] also let users browse alternative designs, but for a different task: converting animated characters into fabricable linkages by replacing motor joints with rods. Lastly, rather than suggesting different

[2]Compared to inverse design, the problems tackled in this section lack a single, well-defined optimization objective. However, this does not mean that the process involves no optimization at all: while the question "what is the best design?" may be impossible to answer in the general case, the question "what is the design closest to the current one that respects all the constraints?" is often answered repeatedly during an exploration session [153, 35, 34, 70, 105]. This is also the case in Chapter 3.

designs, Schulz et al. [117] highlighted the deformation directions on a selected part that would best improve the performance of a robot for the desired ground locomotion task.

Subspace-based exploration. Subspace-based exploration takes a more active role in changing the way users control a design. This type of method addresses two sources of constructive complexity: (i) an excessive number of design parameters [126, 118, 156], and (ii) the impossibility for users to predict what specific design is achievable under the various feasibility constraints [153, 35, 105]. In this type of approach, each specific design is considered as a point in a high-dimensional space (called "design space" or "shape space"). The problem consists in building a low-dimensional design subspace that is easy to explore while respecting various functionality and fabricability constraints. In the context of architectural design, several works have focused on the *rationalization* of freeform surfaces (understood as the discretization of these surfaces into flat polygons that can be fabricated and assembled). Specifically, these methods allow users to explore the shape space of meshes that already respect rationalization constraints, without needing to enforce them *a posteriori*. Yang et al. [153] showed how to explore a local approximation of the subspace respecting such non-linear constraints. The subspace is shown as a clickable 2D heat map. The more the current design is modified, the larger the distance gets from the initial point in this subspace, and the more constraints tend to be violated; therefore, this type of method usually features a *reprojection* step, where an optimization is used to find the closest design lying on the constraint-preserving manifold in shape space. Deng et al. [35] observed that such constraint-based deformation methods often lacked local control ("local" on the mesh, not in shape space), and presented a method to build a local subspace approximation that minimizes mesh deformations far from the region of interest. This algorithm, as well as another by Deng et al. [34], also allows model-based exploration, as described below. Besides these geometric approaches, researchers have investigated ways to expose subspaces enforcing physically based constraints. Shugrina et al. [126] proposed a general framework where the design space is sampled, and computationally expen-

sive validity tests are performed offline, so that the subspace of valid designs can be approximated by a spatial tree structure. At interaction time, predefined sliders are exposed to the user, featuring dynamically updated validity ranges which are computed from the tree. This particular interaction inspired the dynamic bounds featured in our own work (Chapter 3). Schulz et al. [118] tackled a different problem: exploring the Pareto optimal front of a design space in the case of multiple optimization objectives. The advantage is that design variations can be directly explored in the low-dimensional space defined by these performance metrics, instead of the potentially high-dimensional design space. Zhu et al. [156] later proposed a similar type of interaction to explore the space of foldable carton patterns according to three high-level properties: material efficiency, ease of folding and stability.

Model-based exploration . Such systems let users modify the 3D model directly, and immediately propagate changes to the rest of the model to maintain constraints. They address similar sources of complexity as subspace-based methods, with the difference that feasible design changes are easier for users to predict. As a result, some works prefer to use the word "editing" rather than "exploration", although it may remain difficult to predict exactly how the requested deformation is going to affect the entire design. This type of interaction is also called "handle-based" when the user initiates the deformation by directly moving vertices on the mesh. The method by Deng et al. [35], described earlier, allows subspace-based exploration given a handle-based deformation, but also features a global optimization to apply this deformation while respecting constraints and minimizing non-local changes. Later, Deng et al. [34] proposed a constrained optimization framework that is fast enough to be continuously run along the deformation path obtained by moving the handle, effectively making the design travel on the constraint-preserving shape space manifold. Interestingly, an alternative form of subspace-based exploration is provided, not by changing the shape space parameters, but by changing the respective weights of the soft constraints (as well as by adding or removing such constraints). Koo et al. [70] proposed another constrained optimization framework to map functional specifications (e.g., "cover", "fit inside") to a specific design, in order

to model "works-like" prototypes of furniture. Here again, the optimization is fast enough to change a parameter and witness the other parameters change in real time. Pérez et al. [105] introduced a comprehensive design system to create Kirchhoff-Plateau surfaces. Their interface features both subspace-based and handle-based exploration. At a conceptual level, the aesthetic qualities of such surfaces and the highly constrained space of valid designs make their problem very close to our work on drawing machines; therefore, I describe and compare it more in more detail in Section 3.2. Lastly, in the context of assemblies performing a desired motion, some works allowed users to deform a part while automatically adapting the others to enforce the target behavior, such as Umetani et al. [144] with paper planes, Zhang et al. [155] with mechanisms retargeted to custom shapes, and Geilinger et al. [45] as well as Ha et al. [50] for various robotic tasks. More rarely, some systems let users change the geometry of both parts *and* motion paths, such as Bächer et al. [3] with planar linkages.

Guided exploration is useful to give users constructive control even when the design space is heavily constrained. However, in most existing works this user control remains primarily at the level of the geometry rather than the simulated behavior (as opposed to some inverse design methods presented next). In Chapter 3, I present a guided exploration method that is completely behavior-centric, and allows users to define constraints directly on the simulated behavior representation.

2.4 Inverse motion design

Finding the right assembly configuration to perform a desired physical behavior is an example of *inverse problem*: given the result, the goal is to determine the cause. The main source of constructive complexity here is the relationship between design parameters and simulated behavior. While inverse design problems have been investigated in many contexts (e.g., sound [13] or elasticity [87]), a significant number of works have focused on artifacts that produce or undergo a specific motion. Since it is also the case for the case studies of this thesis, in this section I restrict the scope to inverse motion design. Compared to the previous sections, here users

are allowed to specify part of the assembly, part of the motion, or both. However, the workflow is more unidirectional in the sense that the system provides a single result directly from the user's input. This type of approach significantly reduces the constructive complexity for novice users. However, even within this context, various levels of constructive control may be deemed desirable for users. There are four main categories of motion specification: (i) motion paths, (ii) motion poses, (iii) motion routines and (iv) predefined motion.

Specifying motion paths. At this level of control, users typically provide an *external* assembly, as well as a number of simple motion paths tracing out the trajectory of some end-effectors in this assembly. The goal of the design algorithm is to choose the right underlying mechanical elements and optimize their shape parameters in order to produce the target motion. Research in this area has mainly focused on the *kinematics* of mechanical assemblies. Zhu et al. [157] introduced one of the first mechanical toy modeling frameworks, in which the user defines the rotation and translation of some character parts. The algorithm fills an underlying box with the mechanical elements required to drive this motion, connects them together, and runs an optimization to find the best design parameter values. Coros et al. [26] proposed an improved method that notably accepts more complex end-effector motion curves drawn by the user. Since some aspects of this approach influenced our own work on drawing machines, I describe and compare it in more detail in Section 3. Ceylan et al. [19] followed on these works with a system that accepts a different kind of input: motion capture sequences. While the number of motion paths to follow is higher than in previous methods (one for each bone of the captured model), each individual path is approximated to become planar, cyclic, and realizable by a small combination of gears, pulleys, and four-bar linkages. The resulting assembly produces a motion that is more a stylized interpretation than an exact reproduction. In a similar vein, Jeong et al. [61] created a augmented reality environment that tracks the motion of hand-held characters drawings, finds the best matching mechanism, and projects it back on a board for editing via a tangible interface. In the specific case of drawing machines, Takashi and Okuno [135] proposed a method which, compared

to previous works, does not require the input curve to be continuous and closed. On the other hand, their cam-based assembly only produces a coarse approximation of the input. While this work is close to ours in terms of application, we implemented a form of guided design exploration rather than inverse design. Meanwhile, some contributions added flexibility to the mechanical assemblies: Megaro et al. [95] proposed a framework to design compliant mechanisms; Song et al. [130] focused on wind-up toys; Ion et al. [59] followed up on their metamaterial mechanisms contribution [58] by introducing a computational design tool that finds the cell configuration reproducing an input motion path. Overall, while such methods allow the user to specify the motion of parts somewhat precisely, they have entire control over the underlying mechanical assembly. Moreover, these motion paths are usually simple enough to be drawn by hand in one go. One possible limiting factor is the necessity to devise the right curve metric as motion paths get increasingly complex, which may significantly impact the optimization landscape.

Specifying motion poses or configurations. In some other works, the motion is specified more sparsely than trajectories: with one or more target assembly configurations. For instance, some authors have proposed solutions to the problem of automatic linkage synthesis given target poses. Megaro et al. [93] proposed an interface allowing users to sketch planar assembly parts in two extremal configurations. Their system then generates a linkage moving the parts back and forth between these two poses. Later, Nishida et al. [99] tackled the harder problem of automatically creating a linkage for 2.5D multi-body objects transitioning between more than two poses. While these examples still focused on kinematics, other researchers have considered mechanisms involving a range of dynamic effects. Here, besides generating the assembly, the complexity may lie in simulating or controlling the physical behavior of interest. Such assemblies may be actuated by the tension in a cable network, such as the animated plush toys designed by Bern et al. [9], the kinematic chains and trees of Megaro et al. [92] and the kinematic wire characters of Xu et al. [151]. While Li et al. [79] also worked on cable-driven mechanisms, they considered the problem of an end-effector applying a specific force and displacement at a number

of points, rather than reaching a target pose. In the context of elastic materials and springs, Ma et al. [83] proposed a tool to design soft pneumatic objects able to reach several target deformed shapes. Chen et al. [20] described a simulation scheme that is fast and accurate enough to optimize the geometry of a jumping mechanism so that it reaches a target configuration by flying over an obstacle. Meanwhile, Bharaj et al. [12] and Takahashi et al. [134] focused on spring mechanisms with several stable configurations. Overall, few of these dynamic systems support non-trivial user interactions: Bern et al. [9] proposed an interactive posing system supported by their fast simulation, while Takahashi et al. [134] included a form of guided exploration. Lastly, works with the least amount of user control allow specifying one pose, and some characteristics of a second pose. For instance, Yuan et al. [154], in their work on transformable characters, let users define the initial configuration of a skeleton, and the outer boundary of its final configuration. Another example is the original problem of *ballistic shadow art* explored by Chen et al. [23]: given an initial configuration of rigid bodies and light source in 3D space, and a 2D shape, the goal is to find the initial velocities of the bodies such that, at some point in time, their combined shadows outline the desired shape. This method was only validated in simulation, and did not lead to any fabricated prototype, notably because the sources of error and unpredictability were deemed to strong too reliably reproduce this behavior in real life.

Specifying motion routines or global characteristics. Another way to specify the motion of an artifact is to describe its behavior at a higher level. This can mean defining global constraints (e.g., wheels having a similar velocity [155]), weighing specific characteristics of the motion (e.g., smoothness and effort [11]), or composing pre-defined motion sequences [117]. Almost all the examples in this paragraph focus on a single goal: locomotion. Indeed, this is a highly complex task that can be realized in a wide variety of ways but is generally deemed too difficult for the user to control precisely. In the context of passive mechanical assemblies, Bharaj et al. [11] proposed an interface allowing users to design walking automata by selecting leg mechanisms from a database, placing them on a body, and tweaking the weight

of various objectives such as walking a long distance of walking more smoothly. Later, Zhang et al. [155] demonstrated how to *retarget* a mechanical template to a user-specified shape. Their interface supports global motion constraints but is much more focused on mechanism and shape editing. As mentioned earlier, they notably allow model-based guided editing. Meanwhile, other researchers have investigated problems in robotics. Megaro et al. [94] described an interactive design system to create robotic creatures that walk. Their core contribution addresses the problem of controller design, i.e., computing time-varying motor values for each actuator in order to achieve a specific gait. The user can interactively adjust the gait type, footstep pattern and morphology of the robot. Schulz et al. [117] proposed a detailed interface allowing users to explore how both geometry and motion affect the performance of a robot with a ground locomotion task. While the user is let free to select various gait sequences and customize the geometry, the system can also optimize the geometry to achieve a specific gait direction. Lastly, Geilinger et al. [45] used a drag-and-drop interface similar to the one by Bharaj et al. [11], but for robots with an arbitrary arrangements of legs and wheels. Their system can optimize the geometry to reach a specific speed or travel through sparse targets, and also include a model-based guided editing tool.

Predefined motion. This section would not be complete without works aiming to achieve a type of motion that is completely built into the problem, and cannot be controlled by the designer. Such works have little in common except that the only input is the geometry of the object(s). Bächer et al. [4] showed how to turn arbitrary 3D shapes into spinnable objects by adapting the internal mass distribution. Li et al. [78] tackled a problem in the domain of reconfigurable objects: making pieces of furniture *foldable* by segmenting the shape into units linked by pivot joints, so that the resulting configuration is as flat as possible. Lastly, Xu et al. [150] proposed a method to turn a pair of 2D shapes into meshing gears. While the target motion is relatively simple (rotation of each part), the key behavior that is sought here is the optimal *transmission function* between the two gears.

Compared to guided exploration, inverse design methods reduce constructive

complexity by running a constrained optimization on the design. While many of these works give users significant constructive control in terms of simulated behavior specifications, these inputs are often kept relatively simple. However, for more complex inputs, it may be difficult for users to predict what is feasible or not. In Chapter 3 I show how a more interactive guided exploration approach allows users to progressively design intentionally complex motion paths. Moreover, although in inverse design simulations are tightly integrated within the design loop, existing systems still typically neglect the influence of physical errors and approximations. In Chapter 4 I present a method to achieve a chain reaction of events that is robust to such uncertainties.

2.5 Accommodating uncertainty

One of the most appealing applications of digital fabrication technologies is *personal fabrication*, i.e., the ability to create customized objects by accommodating variations between users. Very few design systems, however, are meant to accommodate *unintended* or *unpredictable* variations of the design or its context of use. As described in Chapter 4, all steps of a design-and-fabrication workflow are susceptible to uncertainties: physical measurement errors, modeling and simulation approximations, fabrication errors, etc. These sources of physical complexity can lead to many design-and-fabrication iterations which are frustrating for the end user.

The most direct way to deal with uncertainty is to attempt to reduce it. Since "uncertainty" can refer to sources as diverse as approximations, errors and random variation, there is value in finding a common measure: this is the problem of *uncertainty quantification* [90]. A related issue is to determine which input variables are most responsible for the variation of a model's response (in order to focus the uncertainty reduction efforts on these): this is *sensitivity analysis* [54]. While uncertainty can then be reduced by investing resources into more accurate measurements, models and fabrication, another approach is to computationally determine which values of the unknown inputs result in a model's response that best matches experimental results, a technique known as *calibration* [66]. Nevertheless, both of these

uncertainty reduction approaches require a level of physical control and technical skills that might not be available to all users. It is then important to increase the robustness of a design to uncertainties.

In the context of engineering, *robust parameter design* is a methodology introduced by Taguchi [110] to improve the quality of products and processes. Taguchi proposed to distinguish "control factors" (which in Chapter 4 correspond to the nominal values of layout parameters) from hard-to-control "noise factors" (corresponding to actual layout parameter values, dimensions, physical properties, non-physical simulation parameters, etc.). Then, the optimal combination of values of the control factors is chosen so that the target response is robust to the variation of the noise factors. To find this combination, physical experiments are run to measure the influence of each control factor on the correlation between noise factor variation and output variation. This approach assumes that noise factors can be easily controlled in an experimental setting. In the context of Chapter 4, however, most noise factors cannot be physically changed (because the objects are assumed to already exist, and because some factors are non-physical). Even if such experiments were simulated, Taguchi's method scales poorly to high numbers of control and noise factors. While our method primarily increases the robustness of the outcome to deviations from nominal values of the layout parameters, we assume that doing so also makes the outcome more robust to the uncertainty of other variables (notably the objects' dimensions). Moreover, the proposed extension to complex causal chains (Section 4.6) helps the method scale to a large number of layout parameters by conducting a preliminary sensitivity analysis for each critical event in the chain.

Closer to end users, some works in human-computer interaction involve both hardware and software to allow design changes *a posteriori*. Teibrich et al. [137] built a system composed of a 3D printer, a 3D scanner, a mill and a user interface to patch an already printed object by removing parts and adding new ones. Given the original and modified 3D models, their algorithm automatically computes how to patch the 3D print. Compared to typical 3D printing workflows, where new designs are fabricated from scratch every time, this approach significantly reduces the waste

generated across design iterations. Groeger et al. [48] proposed a method to embed heating elements in parts of a design so that the 3D printed object can transition on demand from a solid into a deformable state and back. Both approaches allow not only fixing design errors, but also efficiently repurposing existing 3D prints. Another type of approach consists in modifying the design to increase its robustness against errors. Kim et al. [67] focused on characterizing measurement errors. They conducted two studies to understand how users make erroneous measurements, even when given precise instructions. Moreover, they proposed strategies to accommodate such errors by making designs more robust, which is very similar in spirit to our work on chain reaction contraptions. I compare this contribution to our own in Section 4.2. While all of these works present interesting ways to accommodate errors, and thus reduce physical complexity, they remain primarily focused on static objects. In Chapter 4, I present the first method to accommodate errors in assemblies displaying long (and codependent) motion sequences.

Chapter 3

Designing complex mechanical motion with guided exploration

3.1 Introduction

Toy of the Year in 1967, the Spirograph is an easy-to-use family of interlocking gears allowing to draw a great variety of intricate curves (see Figures 3.1 and 3.2). It is probably the most popular example of drawing machine[1] among the many types invented over the last few centuries (see Figure 3.3). With the recent democratization of personal fabrication tools, a wider range of artists and makers can now create their own drawing machines. The simplicity of the mechanical parts involved makes them relatively easy to fabricate, opening the door to new fascinating patterns.

Designing such machines, however, is particularly challenging. Many of these mechanical devices transform an input rotation into a more complex cyclic output by combining oscillations of different periods and amplitudes. To produce a closed end-effector curve, the radii of mating gears (or equivalently, the number of teeth) need to have rational ratios. It is easy to enforce this constraint by restricting radii to positive integers; the size of the pattern can still be controlled by a global scaling factor. The downside is that the design space becomes much more complex to explore: as the period is governed by modular arithmetic between radii, the visual output can radically change from one value to the next (see Figure 3.2). In other

[1] See for instance the various academic works inspired by Spirograph patterns over the last few decades [36, 51, 81, 39].

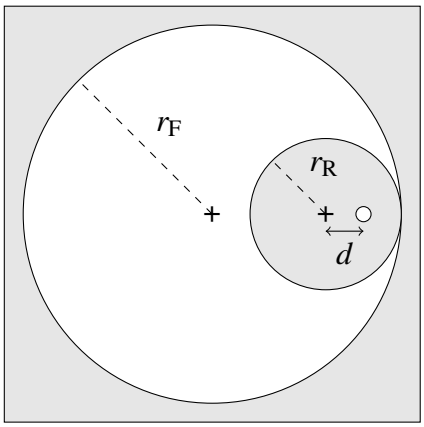


Figure 3.1: Model of a basic Spirograph. In this version, the moving gear rolls inside the fixed one (the teeth are not represented for simplicity). There are three design parameters: r_F is the radius of the fixed gear, r_R the radius of the rolling gear, and d is the distance from the center of the rolling gear to the pen hole. The geometric constraints and equations of motion are detailed in Section A.2.1.

words, the design parameters become *less predictive*. Furthermore, the number of design parameters obviously increases with the number of parts. While this greatly enriches the space of possible curves, manually refining a design becomes difficult with as little as three continuous parameters. Indeed, nonlinearities make the influence of each control hard to grasp, and each one possibly influences the bounds of the others, making the space harder to explore.

In this chapter, we propose a guided exploration method to design complex mechanical trajectories by interacting directly with the output pattern. In contrast to previous works [3], we focus on: (i) highly structured curves, which would be tedious to edit point by point, and (ii) allowing the continuous exploration of local design variations, rather than recomputing a new solution after each curve edit. With model-based exploration, modifications made in one place of the pattern may result in unexpected changes somewhere else. Our method, on the other hand, allows users to define visual preferences and explore the resulting constrained subspace.

Our exploration workflow consists in a coarse-to-fine definition of visual preferences that progressively refine the choice of curves (see Figure 3.4). First, as an entry point into the design space, the user draws a coarse sketch that defines the global properties (e.g., order of rotational symmetry) and appearance of the desired pattern. After selecting an initial curve among suggestions proposed by the system,

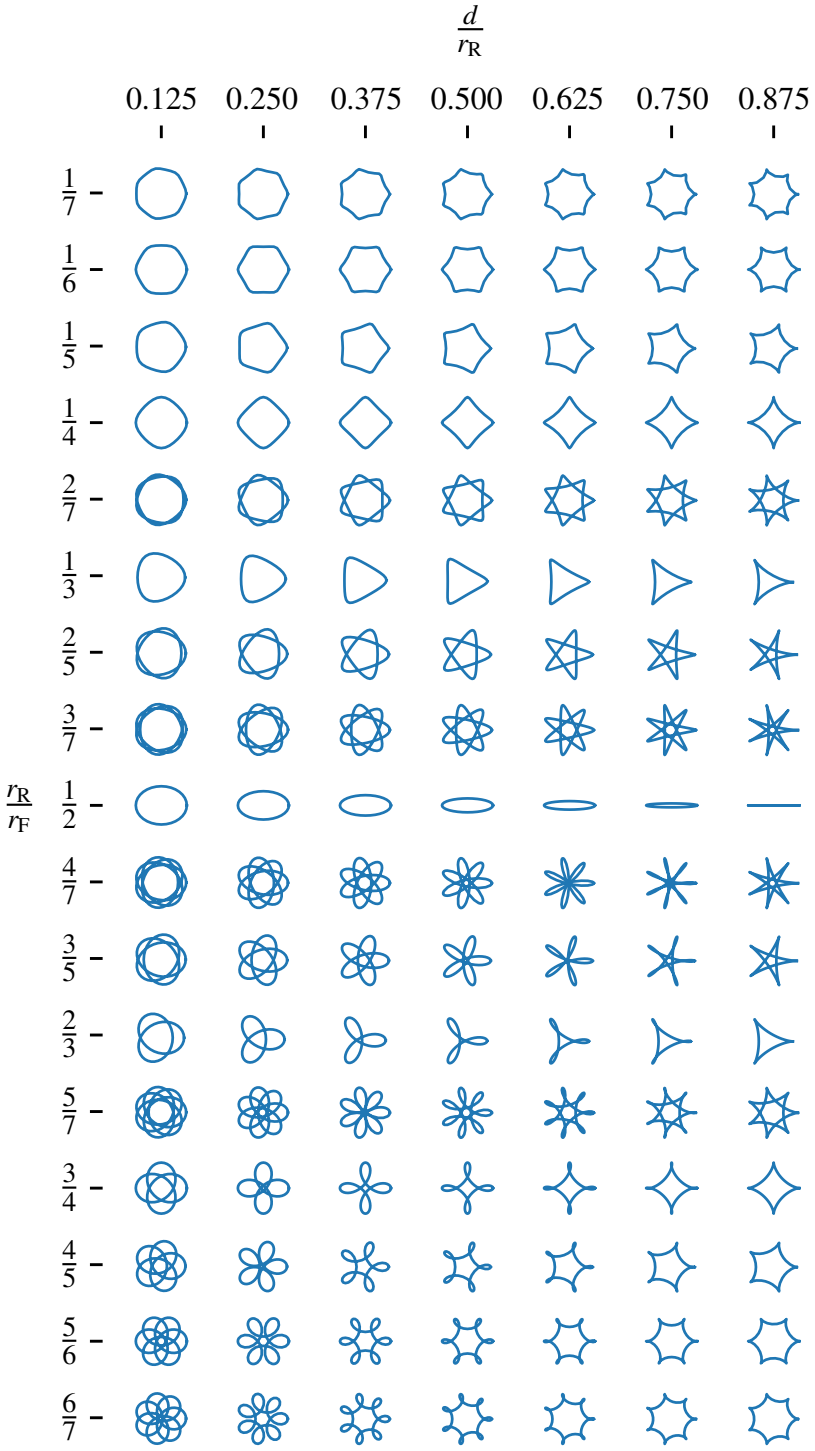


Figure 3.2: Spirograph patterns. The set of finite curves produced by the Spirograph model in Figure 3.1 can be explored by sampling the reduced parameters $\frac{d}{r_R}$ (continuous) and $\frac{r_R}{r_F}$ (discrete) inside an open unit square (excluding the degenerate case $\frac{d}{r_R} = 0$, which forms a perfect circle, and ignoring the global scale parameter). See the curve equations and constraints in Section A.2.1. The values of $\frac{r_R}{r_F}$ must be rational to produce finite curves, and here correspond to the Farey sequence F_7 without the first and last terms (see footnote 3 in Section 3.4).

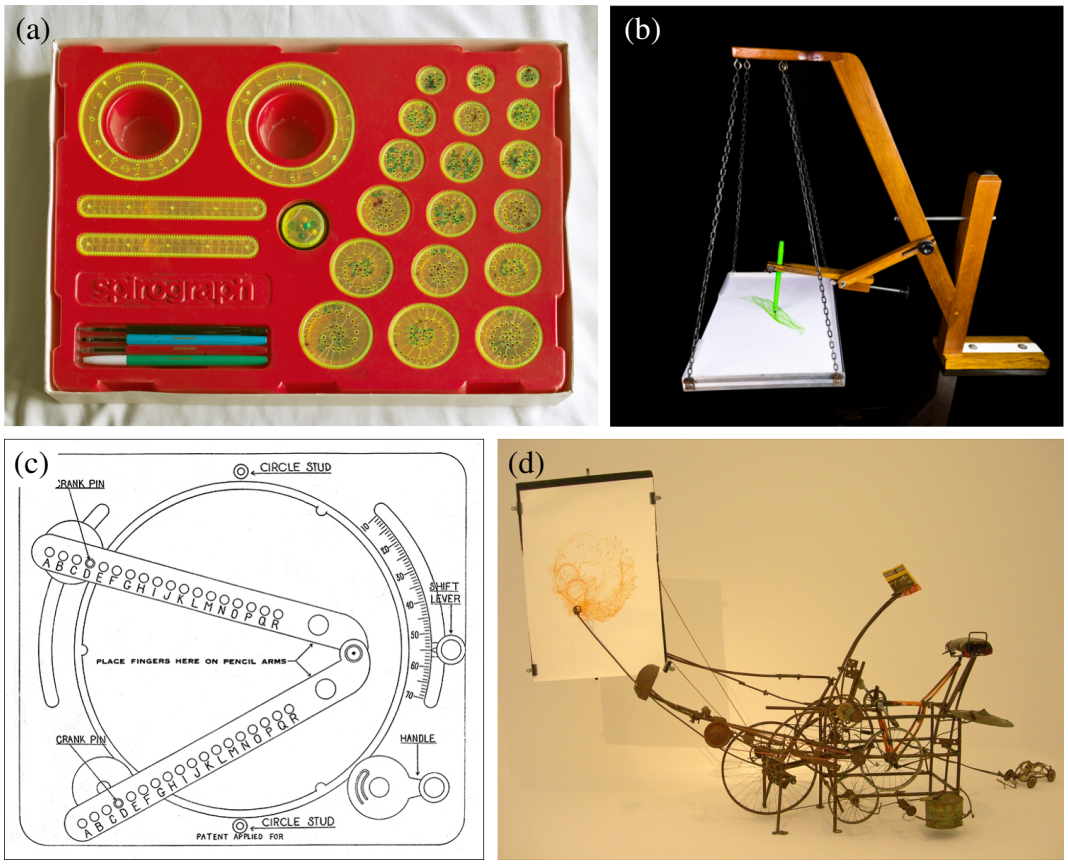


Figure 3.3: Examples of drawing machines. (a) Spirograph, (b) Harmonograph, (c) Hoot-Nanny (aka Magic Designer), (d) Jean Tinguely's Cyclograveur [17]. A historical archive is available at DrawingMachines.org [43]. Modern commercial examples include Joe Freedman's Cycloid Drawing Machine [127] and Bruce Shapiro's Sisyphus [131].

changes can be made via sliders within a domain that respects the feasibility constraints of the corresponding mechanism. When one slider is moved, the bounds of the others are automatically updated. As a key interaction, the user can define *visual preferences* directly on the drawing. These take the form of special points on the curve that can be constrained according to their geometric properties. The user can then explore local variations closest to these specifications via new handles that are automatically generated. Once the user is satisfied, the shape of the mechanical parts is automatically generated and exported for laser cutting fabrication.

Technically, we enable the above key interaction with a novel dynamic reparametrization method that locally samples the high dimensional configuration space of a given mechanism, measures the closeness of each sample point to the

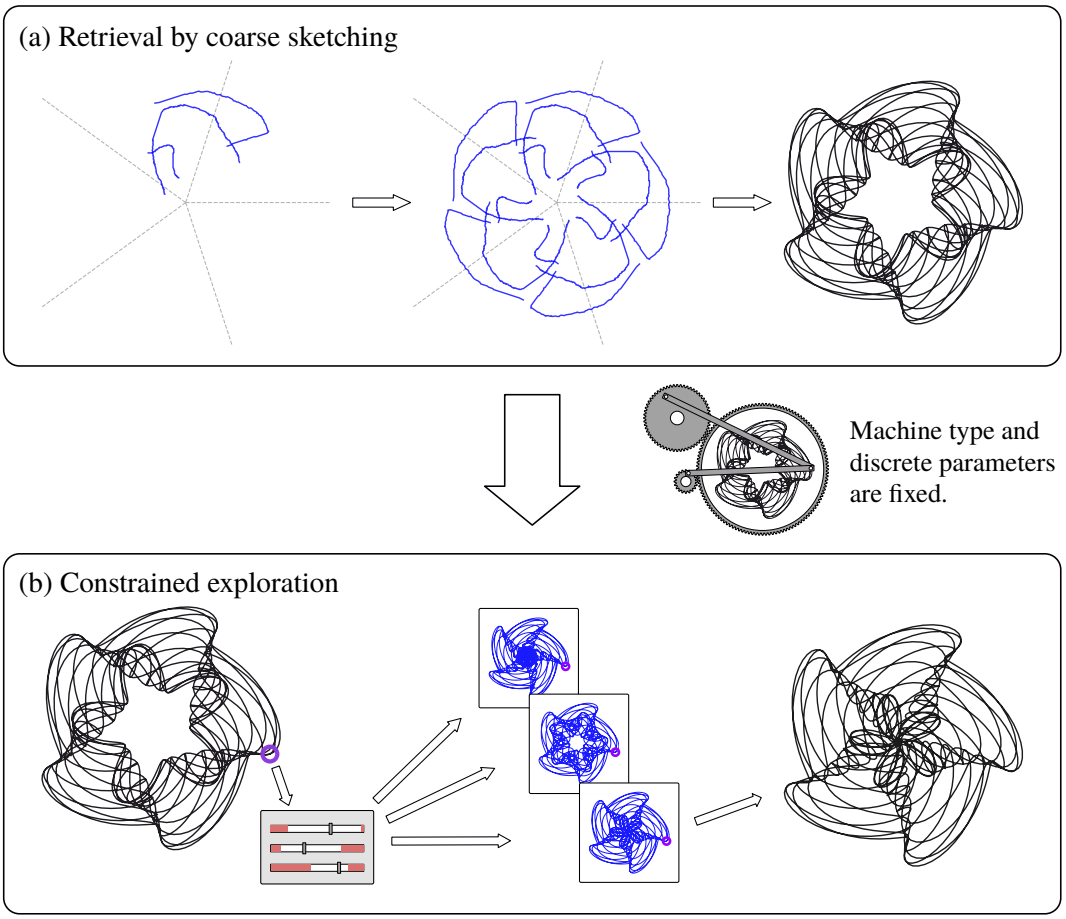


Figure 3.4: Overview of our design workflow. (a) The user first selects a mechanically feasible drawing by providing a rough sketch, and is then able to (b) interactively explore local alternatives by defining visual constraints directly on the pattern (here, the cusp position). The resulting machine is automatically exported to laser cutter profiles for fabrication.

user-defined preferences, approximates the closest subspace, and exposes new design parameters to navigate this subspace.

We evaluated the effectiveness of our design tool on several test scenarios, conducted a user study, and fabricated several physical prototypes able to draw patterns created by users. Overall, we found that dynamic reparametrization allowed users to reliably make meaningful fine scale adjustments to their pattern designs.

3.2 Closely related works

Drawing machines have a long history in mathematics [103, 140, 148], art [17, 25, 102] and crafts [18, 29]. Before computers, tools such as Suardi's geometric pen

(early 19th century) [1] were used to accurately draw a variety of curves. Simulating drawing machines is now relatively easy with a computer-controlled two-axis machine or a robotic arm (using inverse kinematics [60]). Such modern machines, however, can be expensive or require expertise to fabricate and operate (compared to laser cut or 3D printed machines). Their versatility comes at the cost of speed, compactness and power efficiency, especially when the target application requires the same complex motion to be repeated at a high frequency.[2] Therefore, being able to determine the design parameters of a relatively simple mechanism that realizes a target end-effector trajectory is valuable. One of the most fundamental results in this regard is Kempe's universality theorem [65], which states that for any arbitrary algebraic plane curve, a linkage can be constructed that draws the curve. The constructive method proposed by Kempe, however, produces mechanisms with so many links that they are impossible to fabricate in practice (see Figure 3.5). There have been many endeavors since then; for instance, Liu et al. [82] recently proposed a method to reproduce trigonometric plane curves with either Scotch yoke mechanisms or serial chains. Works in this field, however, usually aim to reproduce pre-existing curves for evaluation purposes, rather than to propose a design interface.

More broadly, the aesthetic features and qualities of drawings made with machines have been studied from a variety of lenses: analytical [63], empirical [123, 124, 133], and philosophical [100]. Although such studies differ from our work in terms of goal and methodology (describing and characterizing the space of feasible curves rather than proposing a tool to explore it), they highlight the importance of exploring the space of possible drawings (rather than seeking the best drawing according to some metric). For instance, O'Dowd argues: "Drawing, being located in the arts, fundamentally inverts a task-focused convention. Drawing is not a task to be solved but a space to be explored." [100]

In the context of computational design research, Coros et al. [26] made an

[2] An example of such application is the Risley prism scanner [77]. Similar to a pen moved by a pair of gears, an electromagnetic beam is steered by two wedge prisms (called a Risley prism pair) that rotate relative to each other. The motion pattern of the beam going through this pair is a trochoid [85], i.e., exactly the sort of curve that can be drawn with a Spirograph [51]. The space covering properties of such a curve can be used for accurate scanning and target tracking [24, 77].

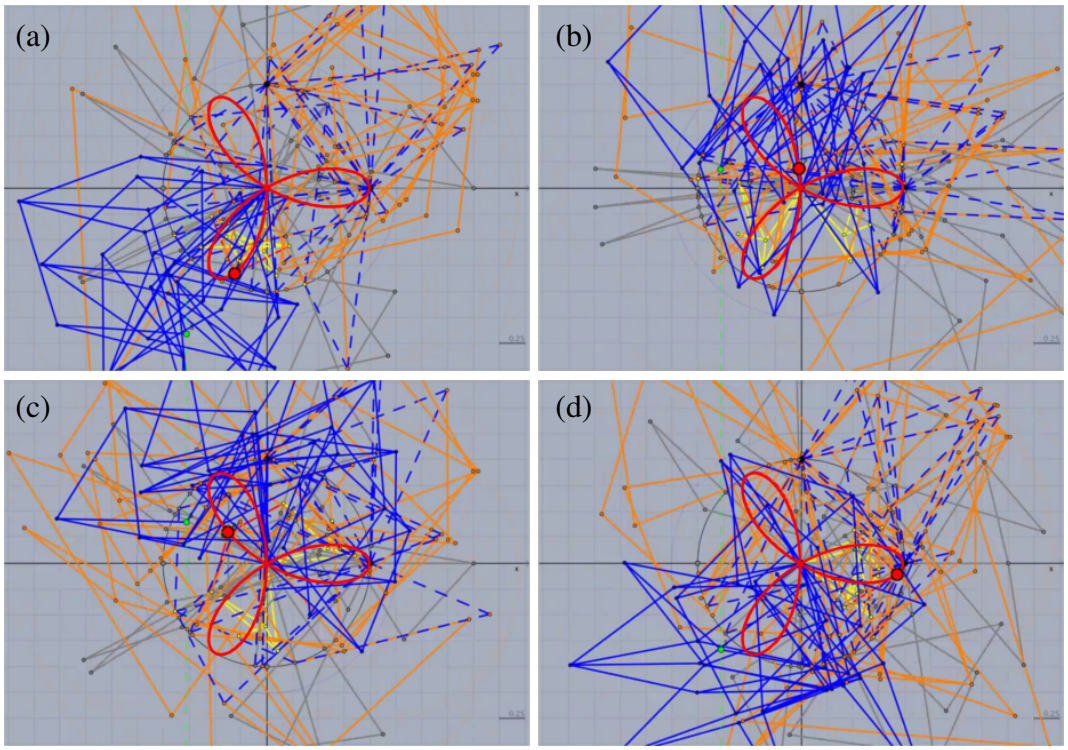


Figure 3.5: Example of Kempe linkage. Four snapshots of a simulated Kempe linkage whose end effector (red vertex) draws a trifolium curve. While such a curve can be drawn with a simple Spirograph (see Figure 3.2), Kempe's general method leads to a structure composed of dozens of links. This specific linkage was constructed using Kobel's implementation [69].

influential contribution in the form of a sketch-based interface allowing users to customize the motion of mechanical characters. The process of matching an input end-effector motion to an assembly configuration is twofold. First, a library of parametrized mechanisms is sparsely sampled and simulated offline to obtain a representative database of motion curves. At the beginning of a design session, the user is asked to compare curves from the database. This allows building a feature-based curve distance function that reflects the user's visual perception. Then, when the user sketches the desired motion of a part, the database is queried using this distance function to find the best matching mechanism, along with a set of initial parameter values. In a second time, assuming that the retrieved curve already matches the input sketch reasonably well, a gradient descent optimization is performed on a point-based curve distance function to find the best continuous parameter values. Once this optimization has been performed for each motion sketch given by the user,

individual mechanisms are connected together with gear trains in a semi-automated fashion, and a support structure is automatically added to finalize the design.

This work was an important source of inspiration for our own method. We query a database of sampled parametrized mechanisms with an input sketch in a similar fashion. However, our approach differs in several ways. First, we auto-complete the sketch by symmetrizing the strokes and connecting them with a fitting spline. Second, we effectively reduce the search space by performing a Fourier analysis on the input curve, and computing design parameter values from the Fourier coefficients. Third, we do not require a initial curve distance calibration, and instead use a symmetrized Procrustes distance that we found sufficient for our needs. Besides this curve matching step, the remainder of our workflow achieves a different type of interaction (as described in Chapter 2): guided exploration rather than inverse design. This is mainly due to the fact that the design space of our drawing machines, although aesthetically diverse (see e.g. Figure 3.15), is so constrained that it is hard for a novice user to know *a priori* whether a specific motion curve is achievable.

For this reason, our design problem also shares interesting similarities with the problem of designing Kirchhoff-Plateau surfaces (KPS) tackled by Pérez et al. [105]. KPS are defined by the authors as networks of thin elastic rods embedded in pre-stretched membranes. The competing tensions of the membranes and rods create, at equilibrium, a 3D shape composed of minimal-surface patches. Because of this complex physical phenomenon, the space of feasible designs is extremely constrained and hard to predict. In other words, as in our work, one cannot expect that any given shape can be precisely approximated by a reasonably simple KPS (or machine drawing in our case). This problem justifies a similar computer-assisted design paradigm. The authors propose a mix of guided exploration and inverse design tools; for brevity, we will focus on the former because it is the closest to our approach. The user first draws an initial rod network that is simulated to previsualize the resulting KPS. A 2D constraint-preserving design space can then be explored to discover feasible shape variations. This space is obtained by computing the dominant modes of the *sensitivity matrix*, which is the Jacobian of the deformed

(equilibrium) configuration variables with regard to the design parameters. The user can impose additional constraints on the deformed configuration, which are incorporated by multiplying the sensitivity matrix by a projection matrix derived from the constraint functions.

The essential difference with our own constrained exploration problem lies in *what* the constraints are applied to. Pérez et al.'s formulation, as other works before [153, 35, 34], defines constraints on vertices of the discretized shape, and assumes that the topology is fixed. In our case, however, constraints are defined on Points of Interests (specifically curvature maxima and intersection points), which can appear or disappear, and merge or split, as the user changes the design. In other words, the problem of *finding correspondences* between constrained points from one configuration to the next, which is trivial in these works, requires a tracking algorithm which may not always return a solution in ours. Consequently, constraint functions are not defined over the entire design space, and thus not differentiable everywhere, which makes constraint Jacobian based approaches difficult. However, we found the Weighted Principal Component Analysis to be a reliable approach in our scenario, as it is more robust against disappearing Points of Interest (at the cost of a higher number of sample points per subspace computation).

3.3 Overview and definitions

Mechanical drawing machines are typically made of gears and bars arranged so that an end-effector can trace out intricate 2D patterns. Such a device physically realizes an algebraic expression connecting the machine part parameters to the output drawing. This tight coupling between the parameters and the resultant pattern variations makes the designers' task of exploring the design space very challenging. Specifically, while on one hand modifying a single parameter may cause several simultaneous changes (e.g., twisting and scaling), on the other hand a single desired change often requires synchronous manipulation of multiple parameters. Our goal is to decorrelate these variations. Rather than trying to find the best possible separation (which tends to be subjective or context-dependent), our goal is to allow users to

define their own visual constraints or invariants in the drawing space, so that other variations can be explored independently.

There are two main technical challenges to tackle: first, mechanisms are often described by a relatively high number of parameters (3-8 in our examples), both continuous and discrete, and whose valid domain is implicitly defined by a set of non-linear constraints; second, mapping invariants in the drawing to a corresponding parameter subspace cannot be done analytically in the general case, as the relation between parameter changes and drawing changes is very complex.

We address these challenges with a two-step workflow (see Figure 3.4). The first step, described in Section 3.4, consists in selecting an appropriate machine by defining global pattern characteristics and providing a coarse sketch. This step notably allows to assign and fix all discrete parameters. During the second step, described in Section 3.5, local continuous variations can be explored while dynamically specifying visual invariants. Our key contribution is twofold: identify a set of recurring geometric regularities involving relevant feature points that can be tracked as the drawing changes (Section 3.5.1), and a novel local approximation method that allows to explore the subspace where such regularities appear (Section 3.5.2).

We evaluate our method in several ways (see Section 3.6). First, we demonstrate a number of cases where our invariants allow meaningful changes in the drawing (Section 3.6.2). Second, we validated the feasibility of our drawing machines by fabricating several prototypes (Section 3.6.3). Lastly, we conducted a user study to assess the ability of invariant-based parameterization to efficiently help navigating the configuration space (Section 3.6.4).

Let us now define the notation used in the rest of the chapter. Each type of machine is described by:

- A set of design parameters $\mathbf{x} = (\mathbf{x}^d, \mathbf{x}^c)$ (the concatenation of discrete and continuous parameters), evolving in a design space D implicitly bounded by a system of algebraic constraints $\mathbf{C}(\mathbf{x})$ (typically nonlinear).

- A simulator that works out the trajectory $\gamma(\mathbf{x}, t),\ t \in [0, \mathcal{T}]$ (where $\mathcal{T}$ is the period) traced by the end point. It also provides a time series of the positions

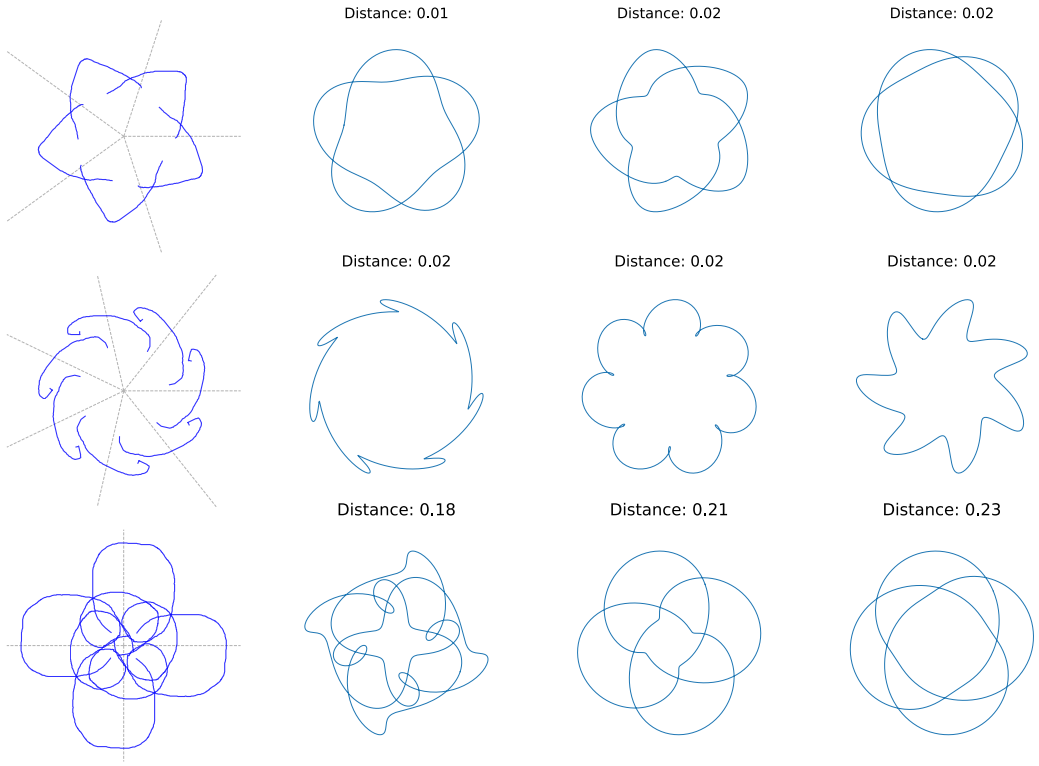


Figure 3.6: Pattern retrieval results. Starting from an automatically symmetrized user sketch (left), we retrieve the best matching patterns in the database—lower value indicates a better candidate.

and orientations of each machine part for visual inspection.

- A representation (or schema) S of the mechanism geometry at different levels of detail (coarse for visual checking, detailed for export and fabrication).

In this chapter, "configuration space" or "parameter space" refers to D, while "curve space" or "drawing space" denotes the Euclidean 2D space of the trajectory.

3.4 Pattern retrieval

The first step of the design workflow is an inverse problem: finding the parameter combination solution of

$$\min_{\mathbf{x} \in D'} \; d(\gamma(\mathbf{x}), \hat{\gamma}) \tag{3.1}$$

where $\hat{\gamma}$ is a spline fitted to the user's sketch (Section 3.4.1), D' is a subspace of D computed from the features of $\hat{\gamma}$ (Section 3.4.2), and d is a measure of dissimilarity between curves (Section 3.4.3). Since the patterns produced by drawing machines

are most often abstract, intricate, and generally tedious to sketch precisely, the result may only coarsely match the user's intent. Our goal in this step is to find a value for the discrete parameters (i.e. the radii), so that the remaining continuous space can be explored.

Once $\hat{\gamma}$ and D' are computed, a set of N_b best matching candidates $\{\gamma_i\}$ is retrieved via brute force comparison of $\hat{\gamma}$ to drawings sampled in D'. Redundant drawings are avoided (to some extent) by making sure that the radii are coprimes (i.e., they do not share a divisor other that 1). This is enforced by sampling values from the Farey sequence[3] (without the first term), which can be computed with linear complexity in the number of terms [114]. Continuous parameters, on the other hand, are naively sampled (within feasible bounds) using grid search. The N_b best matches are presented to the user, who can thus choose the best $\mathbf{x}$ (see Figure 3.6).

3.4.1 Sketching and spline fitting

The user first draws a rough sketch. Construction lines can be used to pre-set the order of rotational symmetry: the pen strokes are then automatically symmetrized (see Figure 3.7).

Our input is therefore a sequence of N_s disconnected, noisy strokes (represented as polylines) that we want to turn into a smooth closed spline. Obviously, if $N_s = 1$ we can fit the spline directly. Otherwise, our goal is to obtain the shortest closed path that runs through all strokes. This amounts to a variant of the traveling salesman problem (TSP), where part of the path is already fixed. We start with the integer program formulation of Dantzig et al. [30], and make the following modification. Let the $2N_s$ ends of each stroke be the vertices V of a complete undirected graph. The set of edges E in this graph corresponds to the connections between each end. In order to force the connection between two ends of the same stroke, we define the

[3] The Farey sequence F_n for any positive integer n is the ascending series of irreducible fractions between 0 and 1 whose denominators do not exceed n. Thus the rational number a/b belongs to F_n if $0 \leq a \leq b \leq n$ and $(a, b) = 1$ [52].

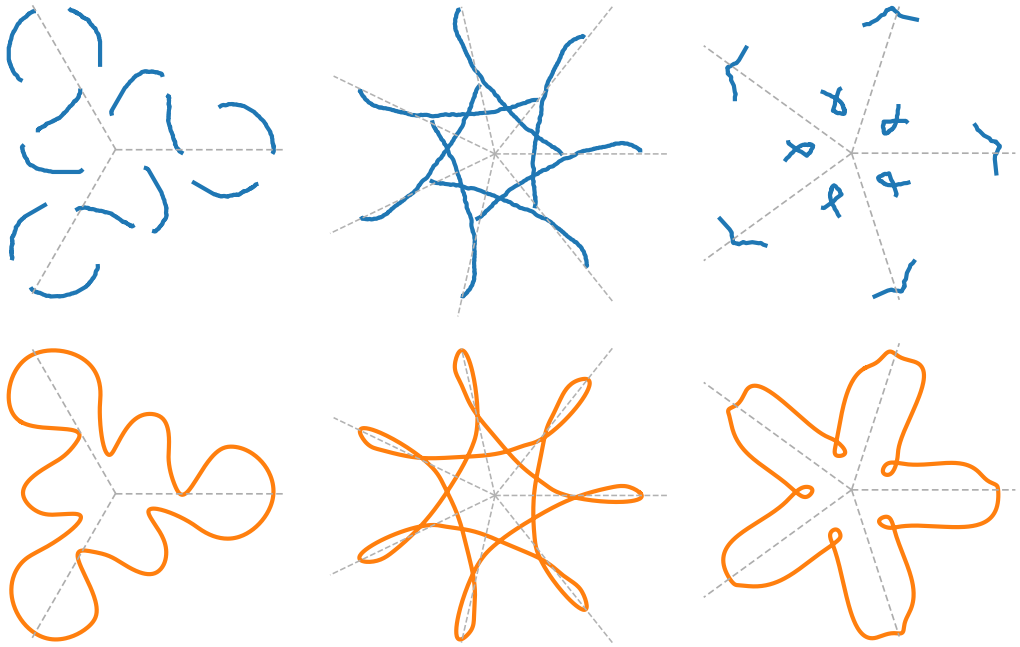


Figure 3.7: Examples of splines fitted to input sketches. Each sketch required only 1–4 strokes that were automatically symmetrized.

coefficient of each edge as

$$c_{ij} = \begin{cases} -k & \text{if } i \text{ and } j \text{ belong to the same stroke,} \\ d_{ij} & \text{otherwise,} \end{cases} \tag{3.2}$$

where d_{ij} is the Euclidean distance between the vertices i and j, and k is a positive constant. For k high enough, it is always more advantageous for the solver to connect the two ends of a given stroke together, while still having the freedom of choosing the connection order. We use the Gurobi solver [49] for this optimization, which relies on a linear-programming based branch-and-bound algorithm. We obtain an ordering of the strokes as well as the points within these strokes (given by the order of the ends). Lastly, we concatenate the strokes and fit a closed cubic B-spline $\hat{\gamma}$ with a user-defined smoothing factor.

We note that while the TSP is NP-hard, in practice the number of strokes tends to be small; therefore, in all of our examples, the processing time of the entire step was around 0.01s.

3.4.2 Reducing the search space

To prevent the grid search from becoming prohibitively expensive, we prune the search space with a number of empirically determined rules. First of all, the degree or rotational symmetry, if there is such a symmetry, is always equal to the reduced radius r_1 of the leading gear G_1 ("reduced" meaning that all radii have been divided to become coprimes). Second, the frequency spectrum of the pattern is a useful source of information regarding the other radii. Indeed, each pattern is a result of the combination of periodic movements produced by the gears. Our strategy starts by computing the discrete Fourier transform (DFT) of a target curve to discover the dominant frequencies, before translating them in terms of gear radii.

First, we sample N_f equally spaced spline parameter values in $[0, 1]$ and compute the corresponding points along the target spline $\hat{\gamma}$. Since the data points are 2D, we can write them as complex numbers $z_n \in \mathbb{C}$, which yields the following relatively simple formula for the components of the DFT $\mathbf{Z} = \mathcal{F}\{\mathbf{z}\}$:

$$Z_k = \sum_{n=0}^{N_f-1} z_n \, e^{-\frac{2\pi i}{N_f} kn} \qquad \forall k \in [0\,..\,N_f - 1], \tag{3.3}$$

where $[l\,..\,m]$ denotes an integer range from l to m. The DFT is, by definition, a sampling of the discrete-time Fourier transform (DTFT), which is itself a continuous function of frequency. Each value Z_k of the DFT corresponds to a frequency f_k given by

$$f_k = \frac{f_s}{N_f} k \qquad \forall k \in \left[-\frac{N_f}{2}\,..\,\frac{N_f}{2} - 1\right], \tag{3.4}$$

where f_s is the signal sampling frequency. The different interval for k compared to Equation 3.3 is not a problem since $\mathbf{Z}$ is periodic over this range. Moreover, by construction, our input curve spans exactly one period $\mathcal{T}$. Therefore, $f_s = N_f/\mathcal{T}$ and Equation 3.4 becomes

$$f_k = \frac{k}{\mathcal{T}} \qquad \forall k \in \left[-\frac{N_f}{2}\,..\,\frac{N_f}{2} - 1\right]. \tag{3.5}$$

Our goal is to find the equation of motion closest to the input curve; therefore, $\mathcal{T}$ is

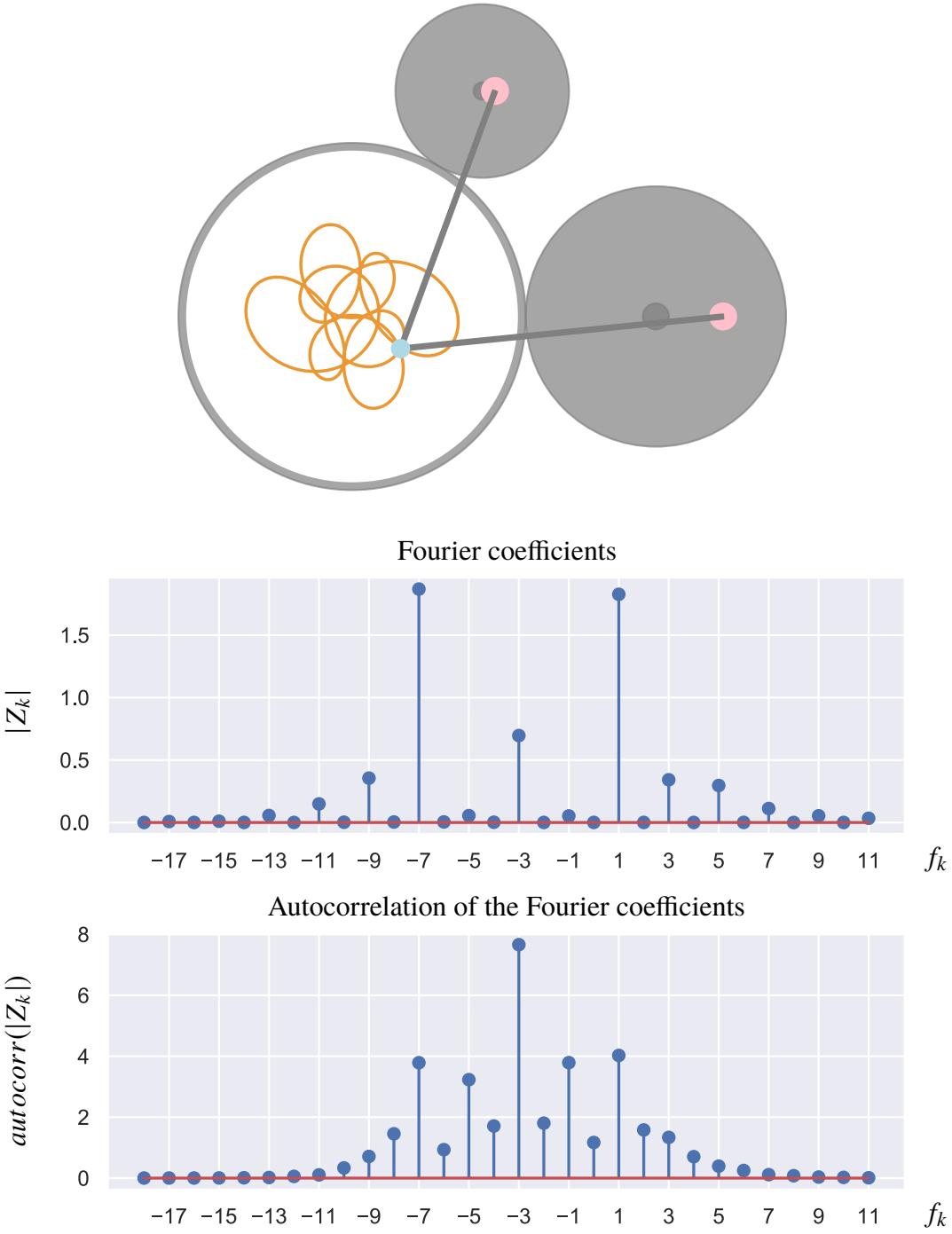


Figure 3.8: Fourier analysis of a Hoot-Nanny drawing. The radii are $(4, 3, 2)$, and the highest peaks of Fourier spectrum happen at frequencies $(-3, 1, 3)$. One can easily check the validity of Equation 3.6.

a priori unknown. As we will see, however, we do not need to find its actual value.

The middle graph in Figure 3.8 illustrates the $(f_k, |Z_k|)$ mapping, also called frequency spectrum, for a given curve (where we arbitrarily set $\mathcal{T} = 1$). As we

can see, for such a clean curve only a few frequencies dominate, symmetrically distributed around the fundamental frequency $\overline{f}_1$. Two of the simplest machines (Spirograph and Cycloid Drawing Machine) present a second dominant frequency $\overline{f}_2$, while the Hoot-Nanny even displays a third dominant frequency $\overline{f}_3$ (both $\overline{f}_2$ and $\overline{f}_3$ being accompanied by their harmonics). We postulate that the number of such frequencies increases by one for each new pair of mating gears in the system (unless they cancel each other out). In practice, however, $\overline{f}_3$ may be a multiple of $\overline{f}_2$, in which case it is indistinguishable from $\overline{f}_2$'s harmonics; this is a case where our analysis would only partially reduce the search space.

To compute the radii of the N_G gears from the frequencies $\overline{f}_i$, we have experimentally found that the radius of gear G_i satisfied

$$r_i = \frac{\mathrm{lcm}(n_1, \ldots, n_{N_G})}{n_i}$$
$$\text{with} \quad n_i = \begin{cases} \mathcal{T}|\overline{f}_i| & \text{if } i = 1, \\ \mathcal{T}|\overline{f}_i - \overline{f}_1| & \forall i \in [2\,..\,N_G], \end{cases} \tag{3.6}$$

where lcm() is the least common multiple, and n_i is the number of rotations of G_i over one period. From the frequency formula (Equation 3.5), it is clear that $\mathcal{T}$ cancels out; therefore, we can set $\mathcal{T} = 1$ from the start, yielding integer frequencies.

Equation 3.6 can be verified in the case of the epitrochoid (i.e., the curve produced by a Spirograph with one gear rolling outside the other [51]), whose Cartesian equation of motion is

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = r_{\mathrm{R}}(q+1) \begin{bmatrix} \cos(t) \\ \sin(t) \end{bmatrix} - d \begin{bmatrix} \cos\left((q+1)\,t\right) \\ \sin\left((q+1)\,t\right) \end{bmatrix} \quad t \in [0, \mathcal{T}] \tag{3.7}$$

with $q = \frac{r_F}{r_R}$, where r_F and r_R respectively denote the radii of the fixed and rolling gears, and d is the distance from the pen hole to the center of the gear. Since Equation 3.7 can be read as a Fourier series, and $q + 1 > 1$ for all radii, we find

$$\overline{f}_1 = \frac{1}{2\pi} \quad \text{and} \quad \overline{f}_2 = \frac{q+1}{2\pi}. \tag{3.8}$$

Taking the ratio,

$$\begin{aligned} \frac{\overline{f}_1}{\overline{f}_2} = \frac{1}{q+1} &\iff q = \frac{\overline{f}_2 - \overline{f}_1}{\overline{f}_1} \\ &\iff r_F \overline{f}_1 = r_R(\overline{f}_2 - \overline{f}_1). \end{aligned} \tag{3.9}$$

Since r_F and r_R are supposed coprimes,

$$r_F \overline{f}_1 = r_R(\overline{f}_2 - \overline{f}_1) = \operatorname{lcm}(\overline{f}_1, \overline{f}_2 - \overline{f}_1), \tag{3.10}$$

which is indeed a specific case of Equation 3.6. The proof for other machines involves much more complex equations of motion, but we can intuitively understand why it still works: different linkages only affect the amplitude of the oscillations, and not their frequencies; the latter only depend on the radii of mating gears. Thus, for instance, the Hoot-Nanny frequency spectrum can be seen as a sum of epitrochoids.

The challenge, then, is to determine the $\overline{f}_i$ from the spectrum of $\hat{\gamma}$. First, we can safely set the f_0 peak (constant component) to 0, effectively centering the curve. Second, while $\overline{f}_1$ is the highest peak for the (Elliptic) Spirograph, this is not necessarily true for other machines (see Figure 3.8). The spectrum, however, always tends to be symmetric; therefore, we perform an autocorrelation of the Fourier peaks (i.e., a discrete convolution of the peaks with themselves) to make the fundamental detection more robust. If the machine has only two gears, $\overline{f}_2$ is simply the next maximal peak. The case of the Hoot-Nanny is more complex, as we saw that its frequency spectrum appears to be the sum of its two gear matings spectra considered separately. This has two consequences. First, $\overline{f}_2$ and $\overline{f}_3$ can only be distinguished if they are coprimes (which happens when the corresponding radii are also coprimes); otherwise their harmonics overlap. While all three radii are constrained to share no common divisor, any pair of them taken separately still can, which is why $\overline{f}_2$ and $\overline{f}_3$ are not necessarily coprimes. Second, their order in the spectrum remains ambiguous no matter what: the next highest peak after $\overline{f}_1$ can correspond to r_2 or r_3, while the Hoot-Nanny curves are sensitive to the order of r_2 and r_3.

From a practical point of view however, the goal of this step is only to reduce

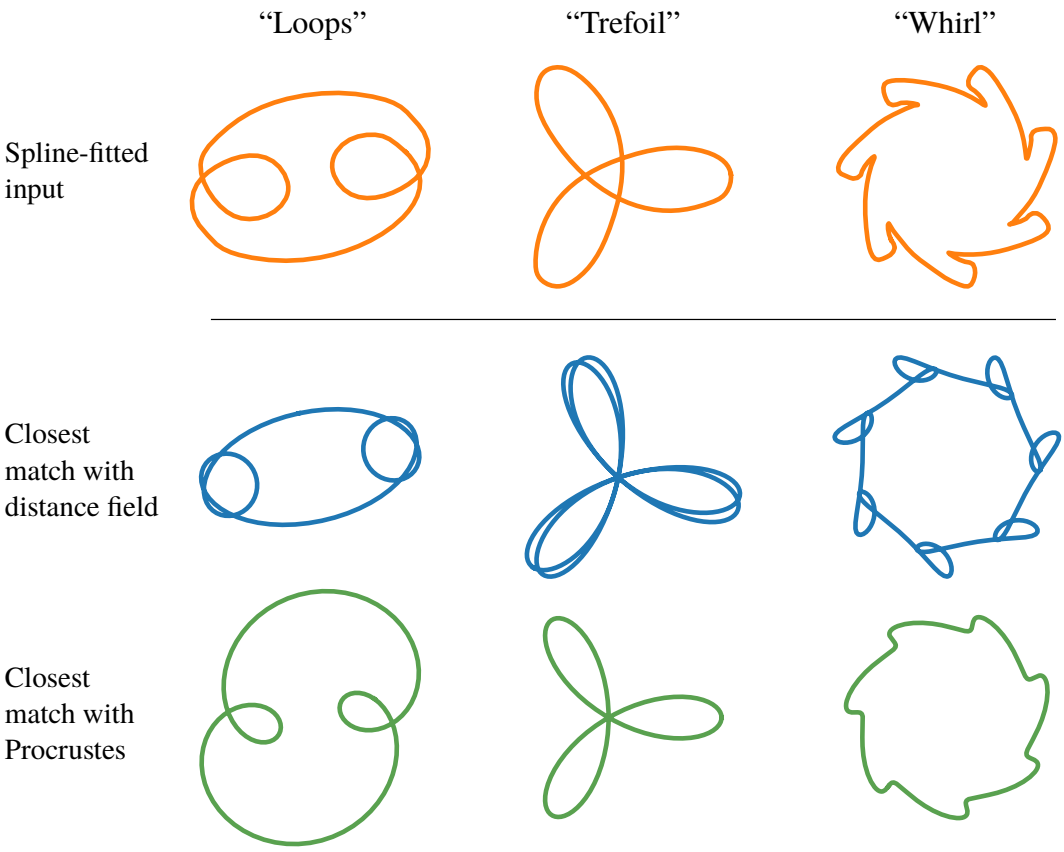


Figure 3.9: Comparison of curve distances. For a given input spline (top row), we show the best matching curve respectively obtained with the distance field metric (middle) and the Procrustes distance (bottom).

the search space, not to completely determine the radii. If the order of r_2 and r_3 is uncertain, we test both combinations; if the amplitude of a peak is too low, we leave the corresponding radius undetermined and sample it along the continuous parameters during grid search. Any pruning of the search space is good to take, since the complexity is combinatorial in the number of parameters. Using a sampling density of 4 values per dimension for all machines in our dataset, our experiments showed that curve retrieval was faster by an order of magnitude when at least one parameter value was fixed.

3.4.3 Curve dissimilarity measure

Let us consider two curves γ_A and γ_B discretized as polylines $\{\mathbf{p}_i^A\}$ and $\{\mathbf{p}_i^B\}$ with N_A and N_B vertices respectively. In the first published version of our method [111], curves were compared by normalizing and aligning them, then computing their

respective distance fields F_A and F_B defined as

$$\forall \mathbf{p} \in \mathbb{R}^2 \quad F_*(\mathbf{p}) = \inf_{1 \leq i \leq N_*} \|\mathbf{p} - \mathbf{p}_i^*\|, \tag{3.11}$$

and finally computing the symmetric distance

$$d_F(\gamma_A, \gamma_B) = \max \left\{ \frac{1}{N_A} \sum_{i=1}^{N_A} F_B(\mathbf{p}_i^A), \frac{1}{N_B} \sum_{i=1}^{N_B} F_A(\mathbf{p}_i^B) \right\}. \tag{3.12}$$

In words, each side of the max tells us, on average, how close a point from one curve is to the other curve. Intuitively, d_F quantifies how curves differ in terms of the "density" of strokes in a region of space. From a practical point of view, this metric accepts a wide range of inputs, as each curve could be a collection of polylines, or even a binary image obtained from a picture. On the other hand, this metric is not as precise as e.g. Procrustes analysis [37] or the discrete Fréchet distance [38], two classic curve dissimilarity measures. The former requires $N_A = N_B$, but inherently normalizes and registers the curves and has linear complexity in N_A and N_B, while the latter has (sub-)quadratic complexity. Having the same number of vertices is not difficult in our updated method, since we can resample $\hat{\gamma}$ to match the size of γ. Therefore, we adopt a symmetrized Procrustes distance as our new measure:

$$d_P(\gamma_A, \gamma_B) = \min \left\{ \sum_{i=1}^{N} \|\mathbf{p}_i^A - \mathbf{p}_i^B\|^2, \sum_{i=1}^{N} \|\mathbf{p}_i^A - \mathbf{p}_{N-i+1}^B\|^2 \right\}, \tag{3.13}$$

where $N = N_A = N_B$. Here, one curve is compared to the other with its vertices taken in increasing, then decreasing order. This is because the regular Procrustes distance depends on the order of the sampling, and we do not know if $\hat{\gamma}$ and γ are drawn in the same direction.

Three comparisons between the distance field metric and the Procrustes distance are given Figure 3.9. Both metrics were used to explore the same dataset of candidate curves and retrieve the best matching one. We observe that although the distance field metric is able to capture the general aspect of the curve, it is insensitive to other aspects such as the arc length because of its image based nature. Thus, for

Table 3.1: Sampling and computation times for different curve distances. Data is shown for the three examples in Figure 3.9. There were 12248 samples before pruning.

Drawing	# samples	Sampling time (s)	Time per sample (s)	
			Distance field	Procrustes distance
Loops	376	0.85	0.0100	0.0006
Trefoil	360	1.04	0.0154	0.0007
Whirl	600	0.94	0.0225	0.0008
		Average	**0.0159**	**0.0007**

example, the trefoil knot in the middle is evaluated as similar despite looping twice. Furthermore, our timing measurements (see Table 3.1) show that the Procrustes distance between curves at a resolution of 1024 points is more than 20 times faster than the distance field metric between curves rasterized at a resolution of 512 pixels.

3.5 Constrained exploration

Once the the discrete parameters of the machine are fixed, the user can focus on fine tuning the continuous parameters. We note that an intuitive system should allow the user to edit different features of the drawing as independently as needed. This is not always possible: the smaller the number of degrees of freedom, the harder it is to prevent several changes from happening at the same time. For instance, a drawing machine with a single continuous parameter would not benefit from our system. Conversely, as the number of parameters increases, so does the extent to which modifications can be decorrelated. However, the exact combination of parameters that allows a constrained change is generally complex to determine, as it requires to either solve a system of non-linear equations, or to resort to manual trial-and-error. Hence, our goal is to efficiently identify, abstract, and expose the space of valid machine configurations subject to the specified constraints.

We allow the user to specify *visual preferences* as geometric properties that should stay fixed when a change is made. Note that this is different from handle-based deformation as the user indicates what *shouldn't change* during editing, rather than a specific target change. Then, our system computes a new parameter space that incorporates the previous machine-specific and global constraints with the new

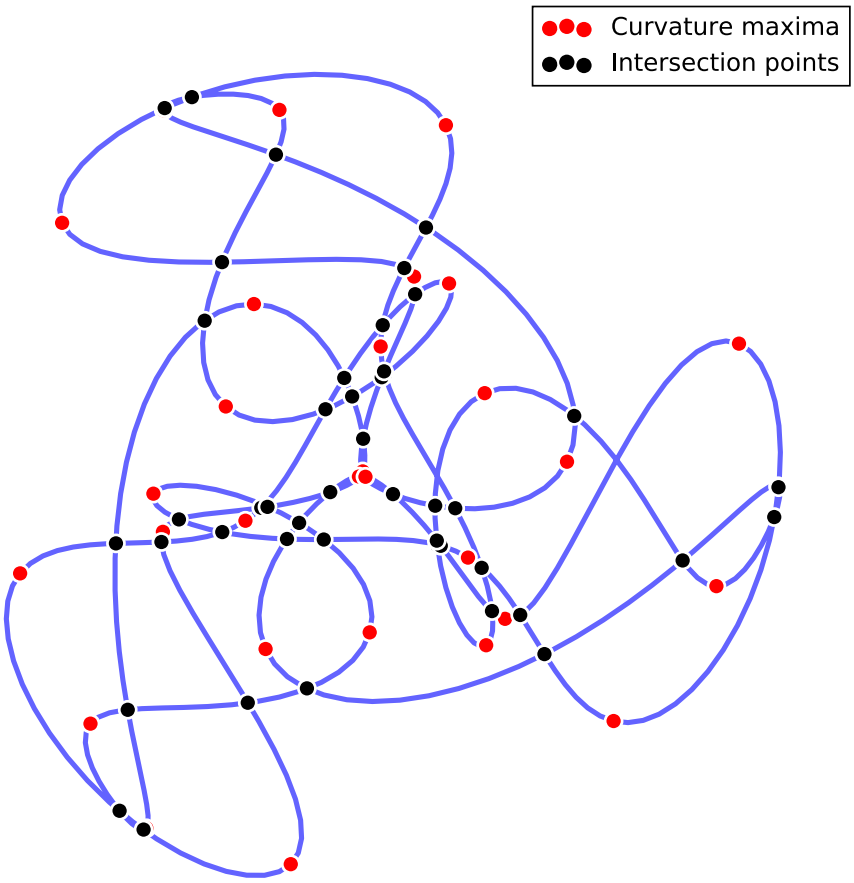


Figure 3.10: Example of Points of Interest in a drawing.

shape invariant(s). The resulting space can be explored via sliders, whose bounds are dynamically updated after each modification. The user can subsequently add more invariants, which further constrain the solution space until no remaining degree of freedom is left.

We first introduce the shape invariants that are supported and how they are dynamically computed and tracked (Section 3.5.1). Then, we propose a local reparametrization method that enables the user to intuitively explore the resulting invariant space in the form of desirable pattern variations (Section 3.5.2).

3.5.1 Pattern invariants

The curves generated by drawing machines are often highly structured and can be described at several levels of detail. If we attempt to decompose such a shape, the smallest discernible element is the point. Among all curve points, some have particular properties that make them stand out, such as intersection points and

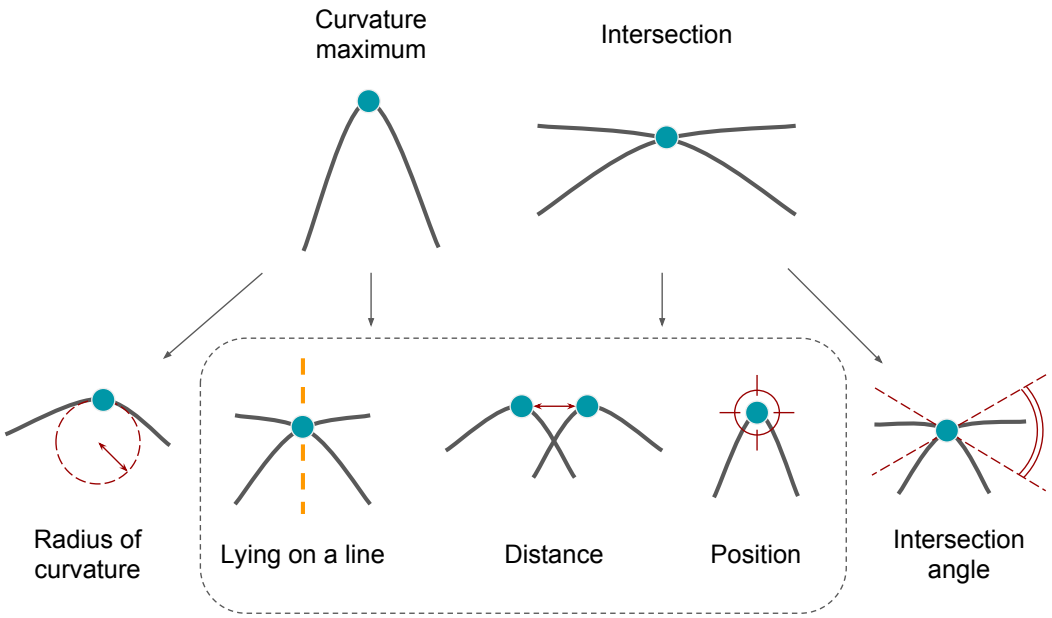


Figure 3.11: Types of Points of Interest (PoI) and associated invariants supported by our system.

curvature maxima (see Figure 3.10). We call them *Points of Interest* (PoI). Such points have generic attributes, such as Euclidean coordinates in curve space, and one (or two) associated time (or arclength) values. They also display properties which are specific to their type, such as the angle made by curve tangents at an intersection point, or the value of the curvature at the maximum (see Figure 3.11).

Next, we define *Relations of Interest* (RoI), as relations that hold either between a PoI and an external object (e.g., a PoI lying on a geometric primitive), or between a group of PoIs (e.g., the distance between two PoIs). Any relation that can be expressed as an algebraic equation involving one or more features of one or more PoIs can be implemented in the system. While higher-level entities could also be considered, such as edges between PoIs, cells formed by edges, or even envelopes formed by sequences of PoIs, we currently only support PoIs and RoIs, as they are easier to compute and track in the parameter space. We note that the computation of the invariant subspace is agnostic of the nature of the features defined by the user.

Selection and computation. During the interactive session, the PoIs closest to the mouse pointer are highlighted. Selecting one (or two) of them opens a menu allowing the user to choose a feature to freeze. For generality, we compute the PoIs on a

discretized curve output by the simulation, instead of solving for them analytically. Curvature maxima are straightforward to obtain, as discrete curvature on polyline vertices is easy to compute. Finding the self-intersections of a polygon, however, is more involved, as the naive algorithm (testing every pair of segments) has quadratic complexity in the number of sides. This problem has been extensively studied and several methods (essentially sweep-line based) have been proposed [31]. We use an implementation of the Bentley-Ottmann algorithm [7], whose complexity is $O((n + k)\log(n))$ with n line segments and k crossings.

Tracking. Key to our approach, PoIs must be *tracked* as continuous parameter values change: in other words, when considering two patterns relatively close in the parameter space, we need to establish correspondences across the PoIs allowing us to quantify how much a specific PoI property has changed between two curves, and therefore, to build an invariant space.

Given two drawings γ and γ' and a reference PoI $\hat{\boldsymbol{\pi}}$ on γ, a naive criterion for such correspondence is to superimpose both drawings and take the closest PoI $\boldsymbol{\pi}'$ on γ'. In some configurations however, several PoIs can overlap each other, leading to ambiguities. This search can be made more robust by considering proximity in terms of the arc length (see Figure 3.12):

$$\hat{\boldsymbol{\pi}}' \coloneqq \arg\min_{\boldsymbol{\pi}'_i} \left\| \boldsymbol{\Lambda}(\hat{\boldsymbol{\pi}}) - \boldsymbol{\Lambda}(\boldsymbol{\pi}'_i) \right\|, \tag{3.14}$$

where $\boldsymbol{\Lambda}(\boldsymbol{\pi})$ gives the arc length (or pair of arc lengths) of $\boldsymbol{\pi}$ and i indexes the PoIs on γ'. This is especially useful for drawing machines where the tracer needs to make a full turn before coming close again to the same area. Moreover, it should be noted that the matching PoI does not always exist: some intersections or curvature maxima are only present in a limited range of parameter values. Therefore, we define a distance threshold σ_{PoI} between the reference PoI and its match, and discard curves for which this limit is exceeded. This threshold can also be used to make the search more efficient: indeed, candidate PoIs in other curves need only be computed in the circle of radius σ_{PoI} centered at the reference PoI.

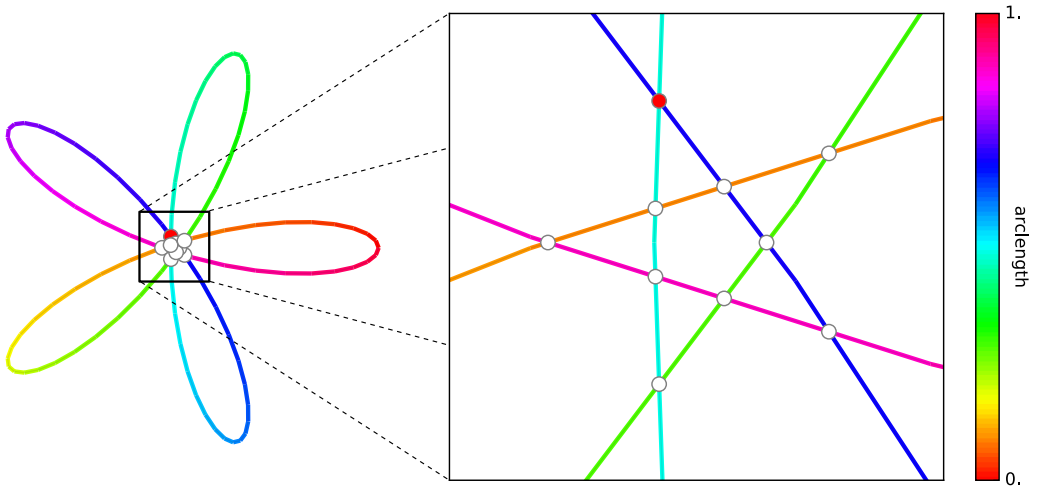


Figure 3.12: Using unambiguous features to discriminate between Points of Interest. Although the red intersection point is close to the others, we can still differentiate it using the pair of arc lengths values at the crossing.

3.5.2 Exploring the invariant space

Once the desired pattern invariants have been selected by the user, the challenge is to explore the resulting constrained parameter space. In the general case, the invariant space is difficult to determine analytically. Therefore, we opt for a sample-based local linear approximation (see Figure 3.13 for an illustration).

In terms of interaction, our algorithm aims to expose new sliders that allow interactive exploration. Since our approximation is only linear, regular re-projections and re-approximations of the invariant subspace are required. We perform them each time a slider is released after a move, which is preferable to continuous updates for two reasons: it ensures interactivity and allows *reversible* changes, i.e., give the user the ability to come back continuously to a previous design while keeping the button pressed. We now describe our approach for subspace approximation.

We consider an n-dimensional continuous parameter space implicitly bounded by several machine-intrinsic constraints, containing a point $\mathbf{x}_0$ associated with the initial drawing γ_0. For simplicity, we assume a single user-defined invariant expressed by

$$d_j^F \left(F(\hat{\boldsymbol{\pi}}^0), F(\hat{\boldsymbol{\pi}}^j) \right) = 0 \tag{3.15}$$

where j indexes the drawing γ_j associated to a neighboring point $\mathbf{x}_j$, F is the

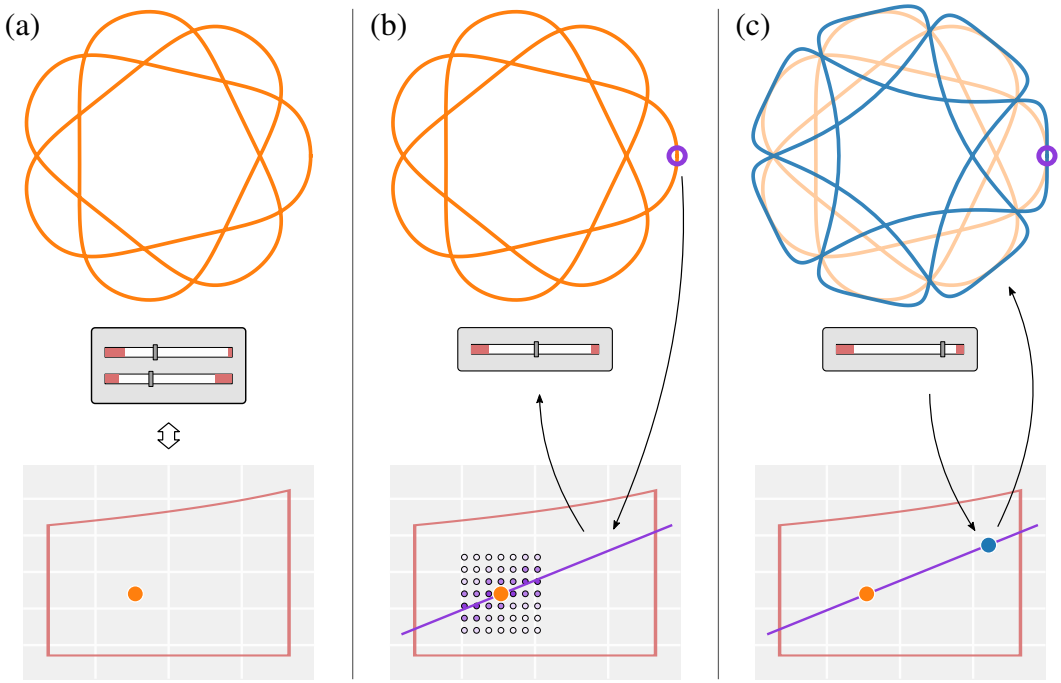


Figure 3.13: Illustrating the invariant space with two continuous parameters. (a) The drawing is controlled by a set of sliders (top) corresponding to the coordinates of a single point in the design space (bottom). (b) The user identifies a PoI directly on the curve and specifies its desired invariant (here: position). Our system then locally samples the parameter space, evaluates an invariance score on them (shown as the dots' color), and performs a linear regression given these scores (purple line). It then exposes a new parametrization (and corresponding slider) allowing the exploration of this subspace. (c) When the user changes the position of the new slider, the corresponding point moves in the subspace approximation, showing a new constraint-respecting curve (in blue).

feature of interest (real- or vector-valued), and d_j^F is the Euclidean distance in the corresponding feature space.

First, we sample neighboring points $\mathbf{x}_j$ within the feasible continuous parameter domain, taking them on a grid whose resolution is adapted dimension-wise to the length of the feasible range. We instantiate the associated drawings γ_j, and track the corresponding PoI $\hat{\boldsymbol{\pi}}^j$. We define the *invariance score* as

$$\mathcal{S}_j := \exp(-d_j^F). \tag{3.16}$$

We will use these scores as weights for the regression of the solution space. Before that, we filter the samples to keep only a fraction of the highest weights. We

assume, given the locality of the neighborhood, that the resulting domain is convex and not disjoint.

Then, to perform the regression, we apply a Weighted Principal Component Analysis (WPCA) to our data centered on the starting point. The algorithm is based on a method by Delchambre [33] that uses a direct decomposition of the weighted covariance matrix to compute principal vectors, followed by a weighted least squares optimization to compute principal components. Since the weights are our invariance scores, this algorithm provides a basis of vectors ordered by decreasing contribution to the invariant space. A local basis can therefore be taken as the first m Principal Components, where m is the dimensionality of the invariant subspace. It is important to note that m cannot simply be deduced from the number of algebraic constraints, which are not necessarily independent. In other words, some constraints may be redundant, either between themselves or with the intrinsic constraints of the mechanism.

In order to determine the dimensionality of the resultant space, we first make sure that it is not reduced to a singleton by checking the number of samples with a sufficiently high invariance score (superior to $\sigma_{\text{inv}} = 0.9$). If less than two points are found, we consider that the system is over-constrained and invite the user to remove one invariant. The WPCA gives us the proportion of variance explained by each Principal Component. Defining v^1_{rel} as the highest relative variance in the set, we keep all components whose proportion is superior to $\sigma_{\text{var}} = 0.1 v^1_{\text{rel}}$. Each axis of the resulting subspace is mapped to a slider shown to the user. If no component is filtered out, we consider that all the invariants were redundant with the intrinsic constraints, and hence keep the original parameterization.

Next, we compute the bounds of the resultant solution space. Since the approximation is local, we do not need to allow too wide an amplitude around the starting position. Since each Principal Component is normalized, we put coarse bounds at -2 and 2. Even then, the intrinsic constraints may impose tighter bounds along some dimensions, which depend on the value of the other parameters; therefore, they need to be re-computed every time a slider is moved. We formulate this as a sequence

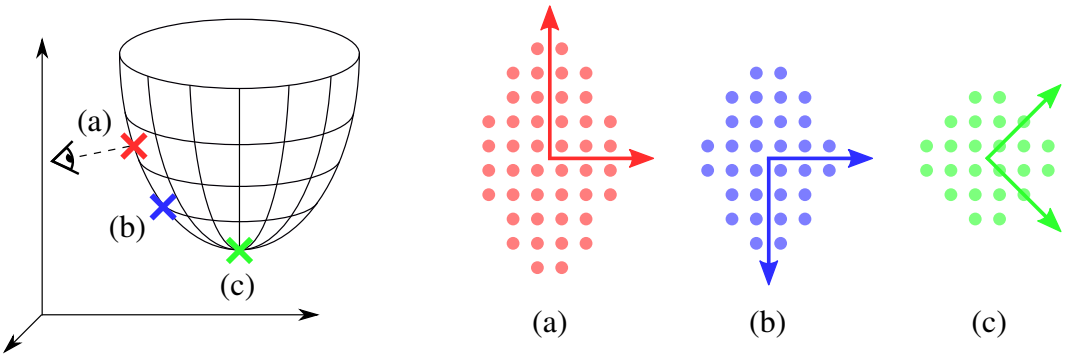


Figure 3.14: Ensuring temporal consistency. Left: constraint-preserving 2D surface in a 3D parameter space, with three successive positions (i.e., designs) on this surface. Right: for each position, we show the additional points sampled around it that sufficiently respect the constraint, and the corresponding first two PCA axes. As illustrated, the Principal Components may flip during a slider move from (a) to (b), or rotate when the user explores a near position (c). To remain as consistent as possible with the original principal directions, we flip or rotate (within the subspace of interest) the axes of the new linear approximation.

of non-linear constrained optimization problems: for each parameter, with the other parameters held fixed, we successively find its minimal and maximal values. Please note that this optimization only uses the intrinsic constraints of the system, which do not require a simulation or the evaluation of PoIs (see Appendix A for details). Further, since we assumed that the local neighborhood was convex and connected, we expect a single range of possible values within the coarse bounds.

We are now ready to present the user with a set of sliders that can be moved while respecting the invariants. Once a slider is released, we update our model accordingly. First, we project the current position back onto the solution space, by finding the point closest to this position that maximizes the invariance score. Then, we re-compute a local approximation of the solution space, following the procedure that has just been described.

In addition, we make the system more intuitive to use by ensuring that the sliders have a temporally consistent *visual effect* on the drawing. Indeed, re-approximating the invariant subspace may typically result in the Principal Components flipping or rotating (see Figure 3.14). Flipping can be easily resolved by comparing the old and new principal directions pairwise—since their order is preserved—and flipping

them back if necessary. Rotation of principal directions, which typically happens when the spread is symmetrical (Figure 3.14c), can be avoided by projecting the previous local basis onto the new one, and normalizing the resulting vectors. This ensures a consistent behavior of the sliders throughout the exploration.

3.6 Case study results

We evaluated our method in three ways: first, we collected non-trivial examples of constrained exploration enabled by our method (Section 3.6.2); second, we ensured that it produces mechanically functional machines by fabricating prototypes (Section 3.6.3); third, we conducted a user study to assess the intuitiveness of our reparametrization compared to the base parametrization of our most complex drawing machine (Section 3.6.4).

3.6.1 Implementation

Our framework was implemented in Python 3.5. For each type of drawing machine (described next), we manually wrote a parametric model including equations of motion, period length computation and physical validity constraints. Most computations, including optimization, are done with NumPy and SciPy, while WPCA provides the Weighted Principal Component Analysis algorithm. The graphical interface of each module was implemented using Matplotlib. Code, demonstration videos and additional details (including package versions) are available online.[4]

Our database of mechanisms contains four parametric models whose specifics are given in Appendix A. Table 3.2 summarizes the main characteristics of these machines. While the Spirograph, the Cycloid Drawing Machine and the Hoot-Nanny are motivated by existing drawing machines, the elliptic Spirograph was designed by the authors to experiment with non-circular gears.

3.6.2 Constrained exploration results

We demonstrate examples of curve invariants for each row in Figure 3.15. Let us discuss each of these experiments.

[4] https://geometry.cs.ucl.ac.uk/projects/2018/drawing-machines/

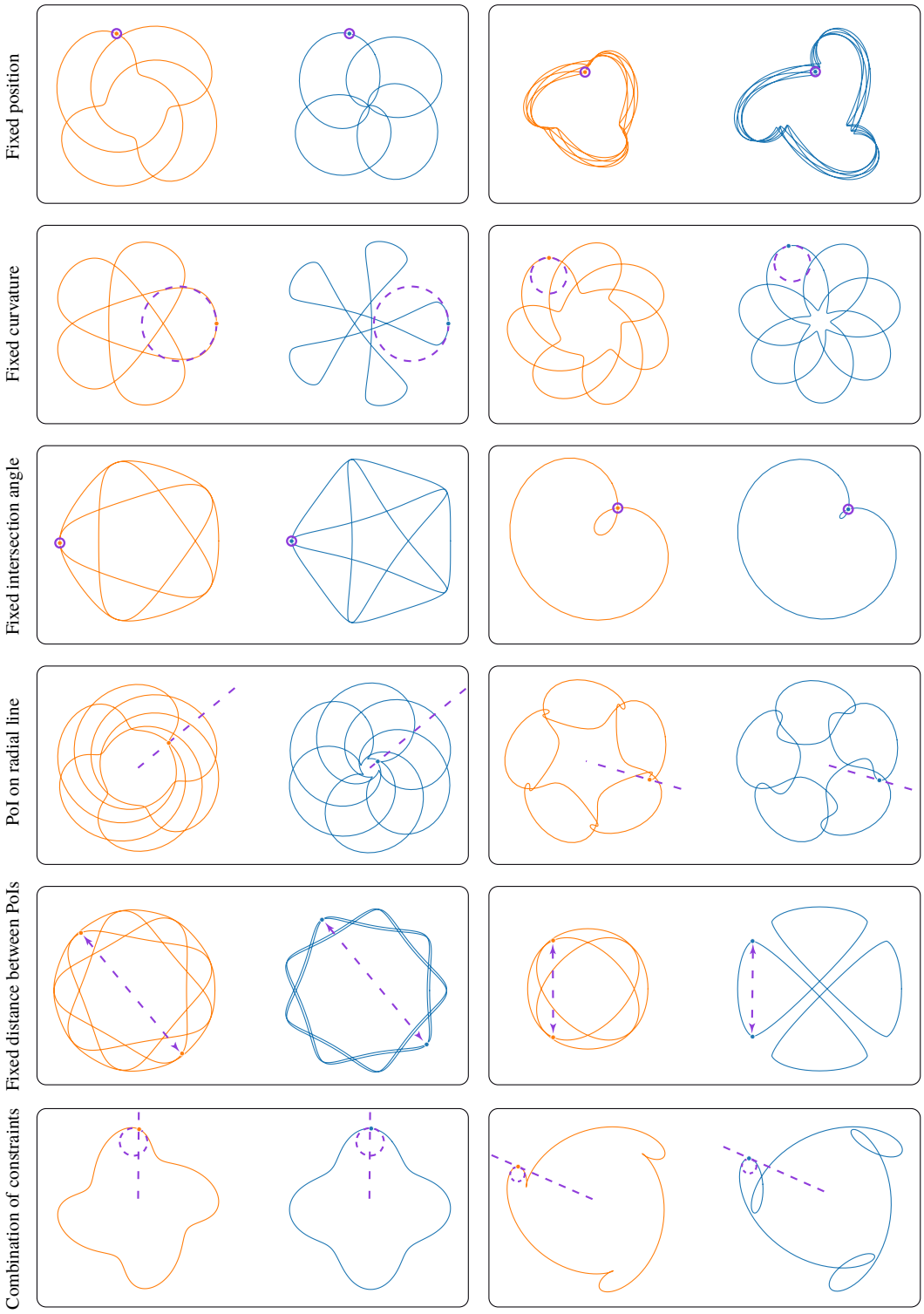


Figure 3.15: Examples of constrained variations obtained with our system. (Original curves in yellow, modified in blue.) Our generated sliders allow significant visual changes to the curves, while respecting the visual constraints (one per row, two at the bottom).

Table 3.2: Drawing machines implemented in our system.

Name	Number of exposed parameters (discrete + continuous)
Spirograph (S)	2 + 1
Elliptic Spirograph (ES)	2 + 2
Cycloid Drawing Machine (CDM)	2 + 4
Hoot-Nanny (HN)	3 + 5

- *Fixed point.* The user fixed the location of the selected PoI. On the left (CDM), the interior boundary was pulled in, while keeping the external arc fixed. On the right (HN), the cusp point is fixed, while increasing the symmetric lobes.
- *Fixed curvature.* The user fixed the curvature at the selected PoI. In the left example (ES), the center was pulled in, while maintaining the PoI's curvature. In the right example (CDM), the central part was reduced and rotated, while maintaining the PoI's curvature.
- *Fixed intersection angle.* The user fixed the angle between tangents at the selected intersection point. In the left example (ES), the center was pulled in while preserving tangency between the curve segments (i.e., zero angle). In the right example (CDM), the loop size was changed, while keeping the inter-curve intersection angle (and symmetry).
- *Moving along radial line.* The user restricted the movement of the PoI along a radial line. On the left (CDM), the center was closed in while keeping the global orientation. On the right (HN), the central part was pulled in and the curvature at the cusp was changed, while keeping the original orientation.
- *Fixed distance between 2 PoIs.* The user fixed the distance between 2 selected PoIs. In the left example (ES), the external boundary size was maintained, while pulling the petals closer together. In the right example (ES), the size of the petals was held fixed, while pulling them apart.
- *Multiple specifications.* In these examples, multiple constraints were specified on selected PoIs. On the left (CDM), the asymmetry was changed while keeping the global orientation and curvature of petals. On the right (HN), the petals were made more ornamental while preserving their curvature and

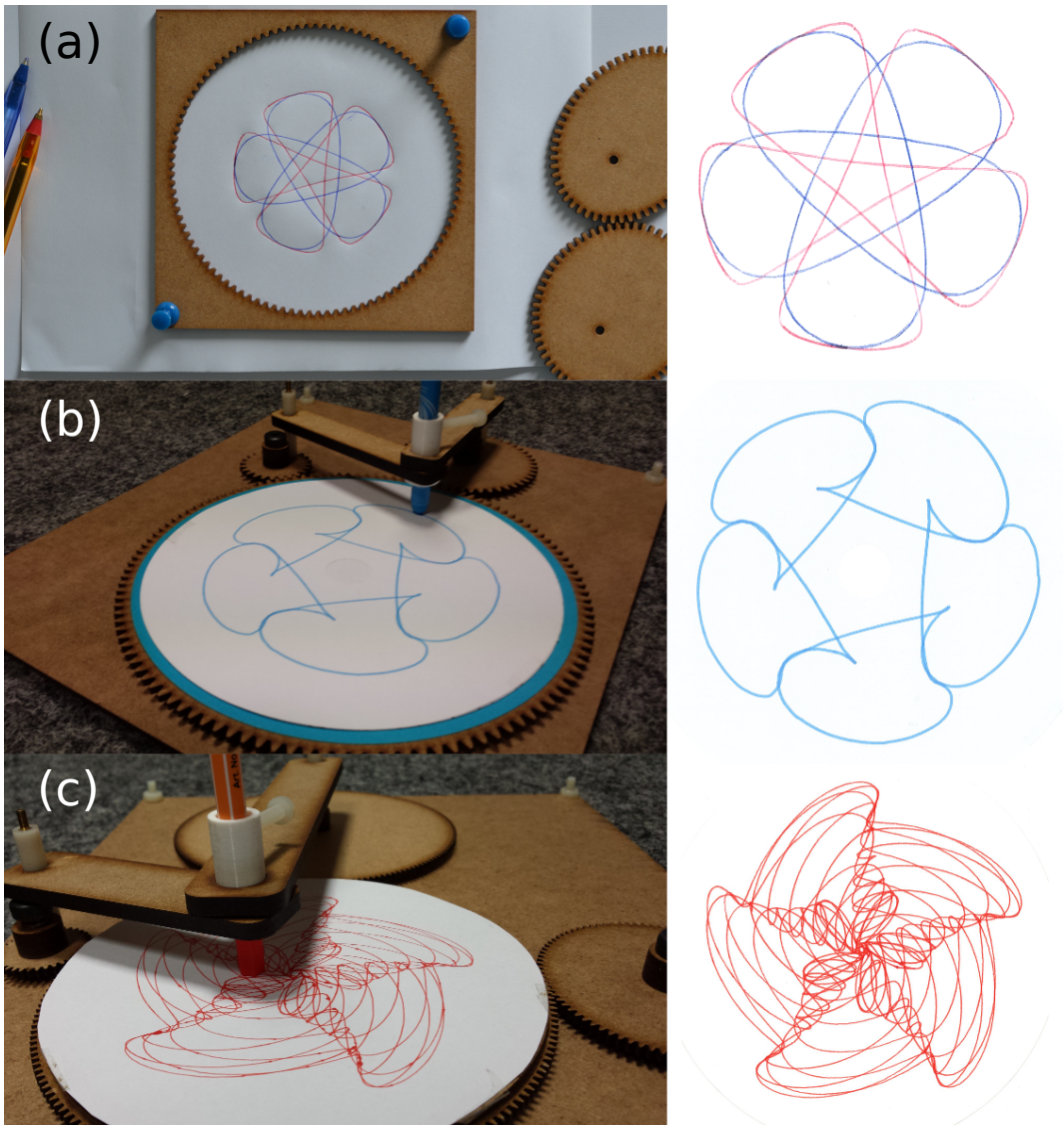


Figure 3.16: Examples of fabricated prototypes. (a) Elliptic Spirograph with two curves drawn using a different elliptic gear; (b) and (c) Instances of the Hoot-Nanny. Scans of the drawn patterns on the right.

restricting movement along radial line.

While the examples above were kept voluntarily simple to emphasize the effect of a given constraint, the videos available online (see link in Section 3.6.1) show more complex examples.[5] Each video also compares the slider changes made in the chosen design space to the corresponding changes in terms of base parameters.

[5]In particular, the video `session12.mp4` shows an example of constrained exploration with an asymmetric curve.

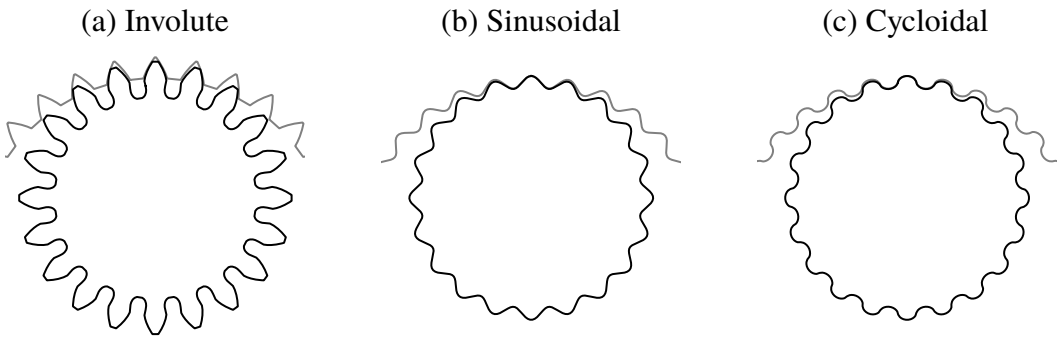


Figure 3.17: Different possible gear profiles. Profile (a), though more complex to generate, maximizes torque transmission.

3.6.3 Precise modeling and fabrication

We fabricated two types of machines (see Figure 3.16):

- the elliptic Spirograph, an easy to fabricate two-parts mechanism that we used to validate the first invariants;
- the Hoot-Nanny, which demonstrates our ability to manage devices with a wider range of parts and connectors.

Our general principle during the fabrication process was to laser-cut the precision-critical, horizontal parts, and to 3D-print the remaining custom connectors, which notably ensure the transmission of movement and support the different layers of flat components. While the vector files given to the laser cutter are automatically generated by a script, the 3D-printed components were designed by hand using CAD software, requiring to adjust tolerances to help the machine run smoothly.

One challenge encountered during fabrication was the design of gear profiles. Such profiles are usually not represented in CAD software, as they would unnecessarily make the geometric model more complex; moreover, these pieces are traditionally manufactured with normalized shaper cutters. Laser cutters, on the other hand, require a precise geometric model as input. Therefore, we implemented a procedural generation of involute gear profiles (which optimize the transmission of torques, see Figure 3.17), for both circular and elliptical gears. The latter, which is less common, was derived from a method by Bair [5].

Pictures of some of the fabricated examples are given Figure 3.4 and 3.16. A

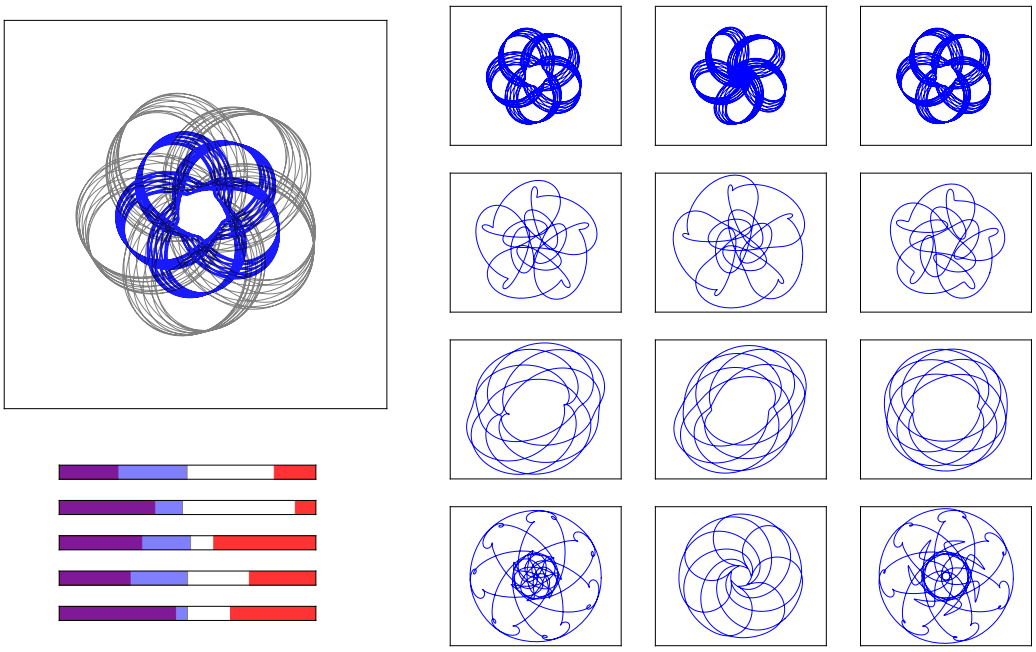


Figure 3.18: User study. Left: interface for a given subtask (target pattern in grey). Right: summary sheet presented to the user in order to rate the results (each column is respectively the target pattern, and results of subtask 1 and 2 in an arbitrary order). Each pattern was generated with the Hoot-Nanny model.

video demonstration of their usage is available online (see link in Section 3.6.1).

3.6.4 User study

We conducted a user study with 8 participants to validate the efficiency of a form of guided exploration based on physical behavior constraints. We chose to focus on an important premise of our method—the fact that defining visual preferences can help navigating the configuration space more easily—rather than trying to evaluate the entire pipeline. This choice allowed to focus on the core contribution of constrained exploration, and made user sessions reasonably short in time and easier to compare.

We defined the following protocol. Each user session was divided into four pattern-editing tasks, where each pattern was generated by our most complex machine (the Hoot-Nanny). In each of these tasks, the candidate was asked to transform an initial curve A into a target curve B, using sliders, in less than two minutes. The set of target patterns was the same for all users, while initial patterns were randomly generated for each new session. The editing operation had to be performed twice: once with the basic machine parameters (subtask 1), and once with parameters

corresponding to a predefined visual invariant (subtask 2). The interface was kept minimal, has shown in Figure 3.18 left. In order to focus solely on the efficiency of the parametrization, we designed both subtasks to be as close as possible interaction-wise. First, the same number of sliders was exposed each time (despite our method allowing to reduce this number), and the order in which the subtasks successively appeared was randomized. Second, the predefined invariant was *not shown* to the user. Lastly, we presented the re-projection and re-approximation process as a little "helper" which could be called by pressing the spacebar, triggering a change in the curve and in the behavior of the sliders. This "helper" had a negligible effect in the base case: a dummy waiting time was triggered (inferior to the time required by the true "helper"), and a tiny perturbation was added to the sliders. This managed to make both versions completely indistinguishable for all users. At the end of the session, candidates were presented with a table displaying their results (see Figure 3.18 right). For each task, they were asked to rate the similarity with the target pattern between 0 and 5.

Results are given for two metrics (number of slider moves and perceived dissimilarity) in Figure 3.19. On average, for two tasks out of four (T1 and T4), candidates were able to reach a final result perceptually closer to the target curve with a smaller number of slider interactions. For the two remaining tasks (T2 and T3), the difference between the base and invariant-space parametrizations is not statistically strong enough to decide on a superior method. Additional metrics, namely the time spent on the task and the Euclidean distance travelled in the parameter space, are provided in Appendix B, Figure B.1. Overall, these results suggest that our parametrization was more efficient for the specific type of pattern editing task tested in this study. We note, however, that this study only partially validates the efficiency of our entire method, as candidates were not allowed to choose their own invariants (which would have required a longer familiarization time). Therefore, the intuitiveness of the Points of Interests and associated invariants has not been assessed. Moreover, reaching a specific target does not exactly correspond to the exploration scenario we envisioned for this method; it is, however, easier to evaluate quantitatively.

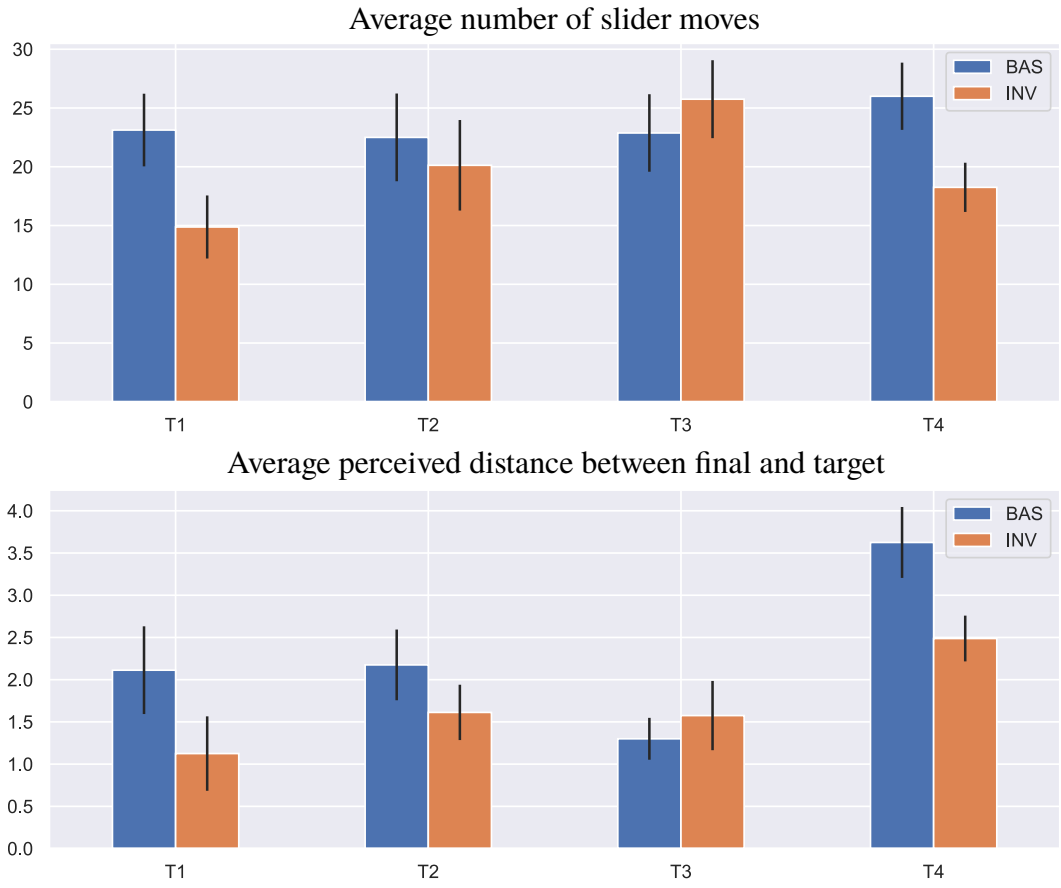


Figure 3.19: User study results. Lower values mean better results. Black bars show the standard error of the mean. "BAS" and "INV" respectively denote the base and invariant-space parametrizations.

3.7 Summary of the case study

I have presented a framework allowing a pattern-centric exploration of fabricatable drawing machines. Users can indirectly select a parametrized machine using high-level scribbles, and then refine the retrieved drawing by specifying constraints on dynamically computed feature points. The main idea of the method is to locally sample the design space and regress to the subspace that best preserves user-specified constraints on Points of Interest in the drawing. The subspace is linearly approximated using a weighted PCA, before exposing these parameters via sliders allowing the user to explore the valid region. This mode of exploration was used on several classical drawing machines to obtain intuitive pattern variations, and a few prototypes were subsequently fabricated to demonstrate that the patterns remained physically feasible.

I argue that the results presented above support two of the main contributions claimed in Chapter 1: first, this method effectively allows a new type of design specification formulated directly at the level of the complex physical behavior representation of an assembly, in the form of geometric constraints applied to Points of Interest in a machine-made drawing (C1). Second, design variations respecting these constraints can easily be explored thanks to a fast approximation of the invariant space that provides parameters to navigate this space with a slider-based interface (C2).

The fabricated prototypes presented in this chapter were able to reproduce the virtual drawings with sufficient accuracy. Physical drawings, however, are likely to degrade as the number of gears and bars increases, due to the aggregation of fabrication uncertainties. The next chapter explores this problem in depth in the context of chain reaction contraptions, and introduces a new method to increase the robustness of these assemblies to various sources of uncertainty.

Chapter 4

Designing chain reaction contraptions from causal graphs

4.1 Introduction

Chain reaction contraptions[1] achieve simple functions from intentionally complex sequences of events (see Figure 4.1). Sitting at the intersection of entertainment, art and engineering, they are featured in movies [55], exhibited as artworks [10] and used for educational purposes in classrooms, science fairs and competitions [55, 68]. A particularly compelling aspect of these setups is the careful management of risk: a chain of events is all the more captivating when it looks like it could fail at multiple points. Contraption builders can spend days trying to assemble these sophisticated structures in a reasonably predictable way,[2] often relying on a rich community knowledge including rules of thumb and specific procedures to try to minimize risks of failure [107, 109]. Despite these efforts, the physical realization of such chains of events remains a delicate art, involving a tedious and very time consuming trial-and-error design process.

Authors of chain reactions face two main challenges. First, small variations at one step may result in wider unintended deviations further down the line (aka

[1] Also known as "Rube Goldberg machines" in the USA, "Heath Robinson contraptions" in the UK or "Pythagorean devices" in Japan [149].

[2] Among the most famous recorded examples of chain reaction contraptions, Honda's advertisement "Cog" required 606 takes over four days [76], while the music video for OK Go's song "This Too Shall Pass" required more than sixty takes over two days [141].

Figure 4.1: Chain reaction contraptions. In a now-famous cartoon series, Rube Goldberg invented complicated gadgets performing simple tasks in convoluted ways (top) [88]. Nowadays, people create physical Rube Goldberg machines in competitions and for entertainment (bottom). These machines are fun and exciting as they delicately balance apparent unpredictability and careful risk management.

the butterfly effect). Limitations in our spatial cognitive abilities prevent us from considering all possible outcomes of a sequence of physical events [120]. As a consequence, long chains of events—even individually simple ones—can easily fail due to a single unwanted side effect. For instance, dominoes arranged along tight, highly curved paths can fall onto each other in an unexpected order. Moreover, orchestrating complex sequences may require carefully synchronizing several simpler sub-chains that run in parallel, or at least being able to robustly predict the completion order of these sub-chains. In other words, a target *causality* between events is often sought, such as a lid being removed from a cup *so that* a ball can fall in it. This kind of effect is essential to make contraptions more visually engaging, as they make potential failures points more obvious to the spectator.

Chain reaction contraptions are an example of real life designs where authoring and assembly are several orders of magnitude longer than the final execution. Hence, despite the efforts of many passionate practitioners, these machines are often limited to linear chains and lack non-trivial causal dependencies involving the synchronization of parallel branches. In this chapter, we investigate the use of computational design to simplify and accelerate the realization of chain reaction contraptions, notably by making designs robust to uncertainties introduced by measurements, modeling and manual assembly.

Developing a computational design tool for such machines is quite different from design problems already tackled in computer graphics research. While the creation of objects and assemblies from target motion has already been investigated (see the survey by Bermano et al. [8]), artifacts were most often fabricated by connecting 3D printed or laser cut parts. By contrast, we face two extra challenges: first, chain reaction contraptions are fully assembled by hand, and each placement error may jeopardize the whole execution. Second, only pre-existing and possibly imperfect physical objects are used as components. The designed layout therefore needs to account for their variability and approximately known features. While the quantification and management of uncertainty is discussed in scientific and industrial contexts [90, 110], the prevalence of intentional risk-taking in chain reaction

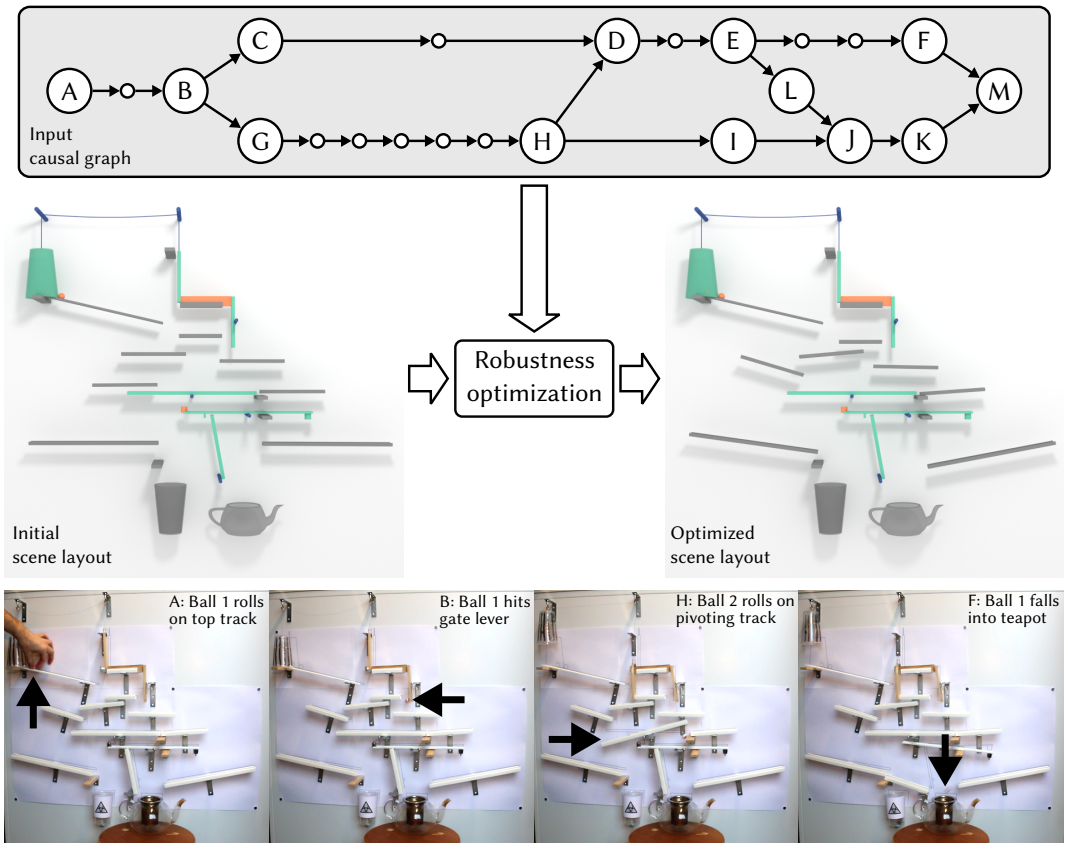


Figure 4.2: System overview. Our system takes as input an initial scene layout associated with a causal graph of expected events. It then combines simulation, search and learning to build a success probability as a function of layout parameters, and optimizes the layout for robustness against the uncertainty inherent to manual assembly. The optimized layout is then exported as a guide sheet and used to successfully assemble complex chain reactions in the physical world.

contraptions, as well as the usually limited resources available to those who build them, makes them a challenging case study.

The key idea of this chapter is to build a simulation-based *success probability* for the intended scenario, parametrized by the layout parameters of the assembly. The input design is subsequently optimized under this estimated probability to improve the robustness to perturbations of the machine layout. More precisely, we start with (i) an initial set of primitive objects (e.g., ball, track) arranged in a coarse scene layout provided by the user; (ii) a set of predefined events (e.g., "rolling on", "falling") arranged in a *causal graph* specifying their expected event order as in Figure 4.2; and (iii) a limit range for each layout parameter. Note that the initial layout does not need to yield a successful run; instead, we expect to find such successful layouts in

the provided design space. Efficiently computing a parametric probability of success from such input requires solving two challenges: first, exploring the potentially high-dimensional design space to find enough successful instances; and second, building an estimator that is accurate near the relevant regions of the design space.

We combine efficient search and machine learning techniques to address both issues. We tackle the first challenge using an adaptive sampling algorithm that progressively trades exploration of the design space for exploitation of the discovered successful regions. We formulate the second challenge as a binary success/failure classification task, where features are layout parameters and labels are derived from simulations run under the supervision of the causal graph. The success probability parametrized by the layout is therefore expressed as the probability of belonging to the "success" class, as provided by the classifier; it is further refined with an *active learning* technique. Simulations are run with a fast deterministic rigid body engine [27], as we posit that a relatively coarse model is sufficient to approximate the layout with the highest probability of success. Additionally, we use sensitivity analysis to identify events holding a critical role in the sequence and map their individual probability of success to the relevant design parameters; this allows our method to scale to a high number of dimensions. Once the parametric success probability estimator is built, we increase the robustness of the layout by identifying and optimizing weak points where the design is likely to fail. Note that our optimization takes place in a space with voids, i.e., containing physically impossible configurations preventing any meaningful measure of success.

We evaluate our framework on contraption examples of increasing complexity, both quantitatively (by computing an integral robustness metric and comparing the output of our method against several baselines) and qualitatively (by building these examples in real life). Our results show that we consistently generate robust designs even in high dimensional configuration spaces. In summary, our contributions are (i) a general methodology to optimize chain reaction contraptions; (ii) a general simulation-based measure of robustness to layout uncertainty; (iii) a divide-and-conquer method to efficiently compute this function for complex chains of events.

4.2 Closely related works

Understanding causal relationships in the behavior of a complex artifact is key to understanding its functionality. Causality models have been used more broadly in a variety of applications, such as representing storyboards in narrative design: pioneering work from Kalra and Barr [62] used directed graphs to analyze and model time and events in computer animation. While our causal graph is inspired from their event graph, our goal is to exploit it to create a real world contraption. Chains of events were also studied for video games and computational narratology, e.g., with the goal of finding a consistent causal order among events [108]. In our setting, however, the causal chain is fully specified by the user. In the context of mechanical assemblies, researchers have investigated how representations can help analyze and understand causal relations in mechanisms [97, 120], while more recently, functional graphs have also been used for reconstruction [80]. By contrast, our work uses the causal graph to build a measure of robustness that is subsequently used to optimize the design. In the very specific context of dominoes, researchers have investigated analytical approaches. For example, dynamic analysis of domino runs (speed of propagation [145], magnification power [146], stable states, etc.) has been applied to other fields with networks of chain reactions [2]. Our work differs mainly in that our approach is not analytical, but empirical (based on simulations).

In the context of design interfaces, Furuta et al. [42] proposed a system to support the creation of kinetic art pieces such as mobile sculptures and chain reaction devices. A rigid body simulation runs continuously while the user adds objects to the scene and adjusts design parameter values. Simulation results are visualized in various ways. First, the loci of some vertices of the moving objects are displayed. Second, a wireframe model of each moving object is added when the object abruptly changes direction. Lastly, collision points are shown on static objects. If the assembly is supposed to reach a static equilibrium instead, the visualization can be switched to show motion arrows instead of trajectories, so that the user can work on minimizing the length of each arrow. While this system is quite useful to design chain reaction contraptions, it does not take uncertainties into account (which could

be shown, for example, as motion cones [6]). Moreover, compared to our work, this is only a visualization tool, and no automatic design improvements are performed.

To our knowledge, the only other system that performs automatic design modifications to accommodate uncertainties was proposed by Kim et al. [67] with the specific goal of reducing the impact of measurement errors. An increasingly common application of personal fabrication, called *augmented fabrication* [84], consists in 3D printing pieces that modify or improve the functionality of preexisting objects. While systems have been proposed to help design such augmentations [21, 22], precisely measuring objects remains a challenge for novice users. Kim et al. conducted two studies to characterize the sources and types and measurement errors. Interestingly, they reported that participants still made errors even when given precise instructions. The authors argued the need to accommodate errors besides trying to minimize them, and proposed two modeling strategies to build more robust objects. First, modular joints or clamps can be inserted at measurement-sensitive locations to allow replacing only parts of the object, which reduces the printing time and material waste. In the second strategy, flexible buffers can be added to accommodate measurement errors in the order of millimeters. This approach efficiently increases the robustness of 3D printed objects. The main difference in our work is that we focus on robustifying an assembly of components to reliably obtain an input chain of events. Moreover, our approach aims to increase robustness against a greater variety of uncertainties.

4.3 Concepts and definitions

Let us introduce some key concepts with an illustrative example (see Figure 4.3). Consider the case of a ball initially at rest on a tilted plank. The ball starts rolling on this plank, gains momentum, leaves the plank, and hits the head of a row of dominoes, which all gradually topple until the last one finally stops.

We have just described a *scenario*, the central component of our framework. A scenario is a triplet $\mathbb{S} = \{S, G, D\}$ consisting of a *scene* S, a *causal graph* G and a *design space* D.

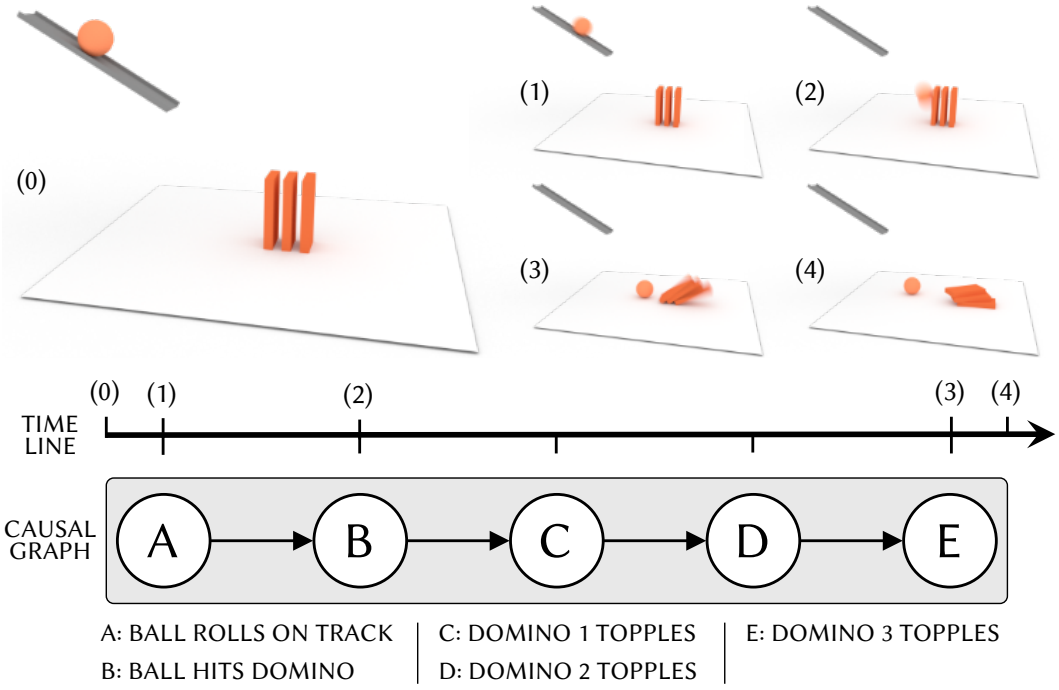


Figure 4.3: SIMPLE scenario. A ball rolls on a track and triggers the fall of a sequence of dominoes. A succession of snapshots taken from the simulation (top) is matched with the events of this scenario's *causal graph* (bottom). (Ticks along the timeline are uniformly spread for clearer visualization.)

Scene. A scene S is a collection of m 3D objects $\{o_i\}_{i=1}^{m}$ laid out in space and organized as a scene graph in which a child object's transform (i.e., position and orientation) is defined in the local frame of reference of its parent. This graph is useful for objects whose initial position is more intuitively described relative to others (e.g., a ball resting on a track). For the sake of convenience, we assume that each scene is made of a small number of *primitives* (in this example, ball, track, dominoes) arbitrarily repeated, combined, and constrained to form an initial setup. Figure 4.4 shows all the primitives implemented in our system. Each object o_i is built by selecting a primitive and fixing a set of constructive variables $\Theta_i = \{\theta_i^j\}$, including both shape (e.g., length, width) and physical (e.g., mass) parameters.

Causal graph. A causal graph G organizes a collection of *events* expected to happen during the simulation. An event $e = (c_e, s_e)$ has a specific definition in our framework: it is an entity characterized by a *condition* c_e and a *state* s_e. The event condition is a Boolean function of time and one or more objects $c_e(t, o_i, \ldots)$

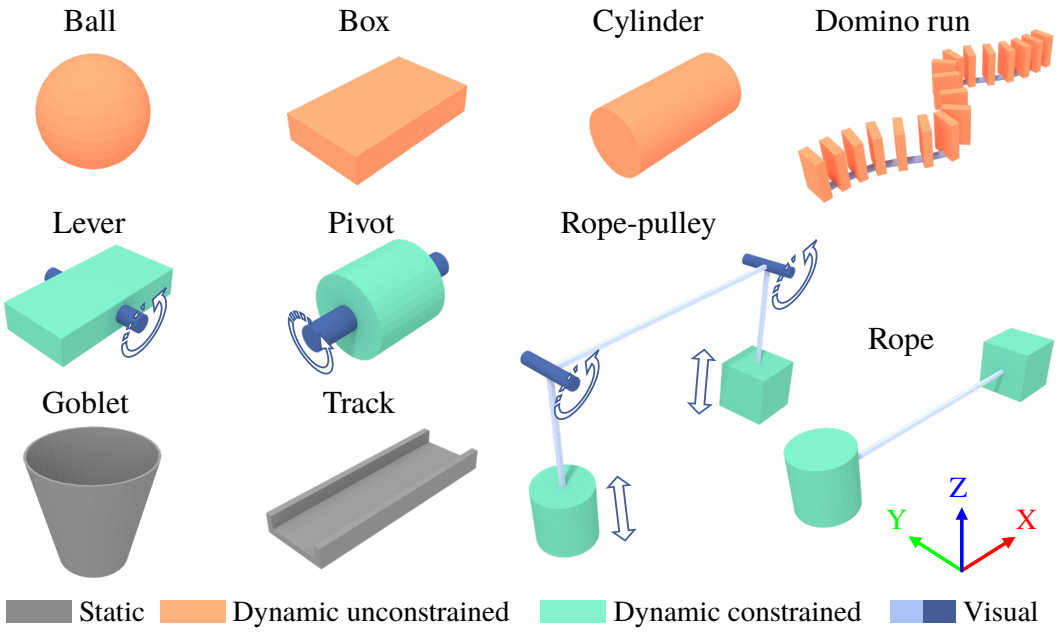


Figure 4.4: Primitive types. The above primitive types are available to the user in our implementation. The color hues correspond to the different types of behavior in the physically-based simulation. Arrows indicate the motion type allowed by the constraint. Please see Appendix C for details.

that evaluates one or more statements about the transform, velocity and/or geometric relationship of these objects at time t. The event state $s_e(t)$ is one of $\{\texttt{asleep}, \texttt{awake}, \texttt{success}, \texttt{failure}\}$. Let $(t_k)_{k\geq 0}$ be the sequence of simulation times. Any event but the first starts with $s_e(t_0) = \texttt{asleep}$, and is triggered `awake` at some time t_e by the `success` of all of its predecessor(s). The condition $c_e(t_k)$ is only evaluated while $s_e(t_{k-1}) = \texttt{awake}$. Since we cannot wait indefinitely for the event to happen, we introduce a timeout duration $t^{\max}$ such that

$$s_e(t_k) \leftarrow \begin{cases} \texttt{success} & \text{if } c_e(t_k) = 1 \text{ and } t_e \leq t_k < t_e + t^{\max}, \\ \texttt{failure} & \text{if } c_e(t_k) = 0 \text{ and } t_k \geq t_e + t^{\max}. \end{cases}$$

The timeout $t^{\max}$ is manually set to match the longest expected time between two events (2s in our experiments). Figure 4.5 shows the events currently supported in our system. We note that our formulation induces a discrepancy with the intuitive definition of some events: the act of falling, for instance, is not instantaneous—it lasts a certain amount of time. In our system, however, the switch `success` or

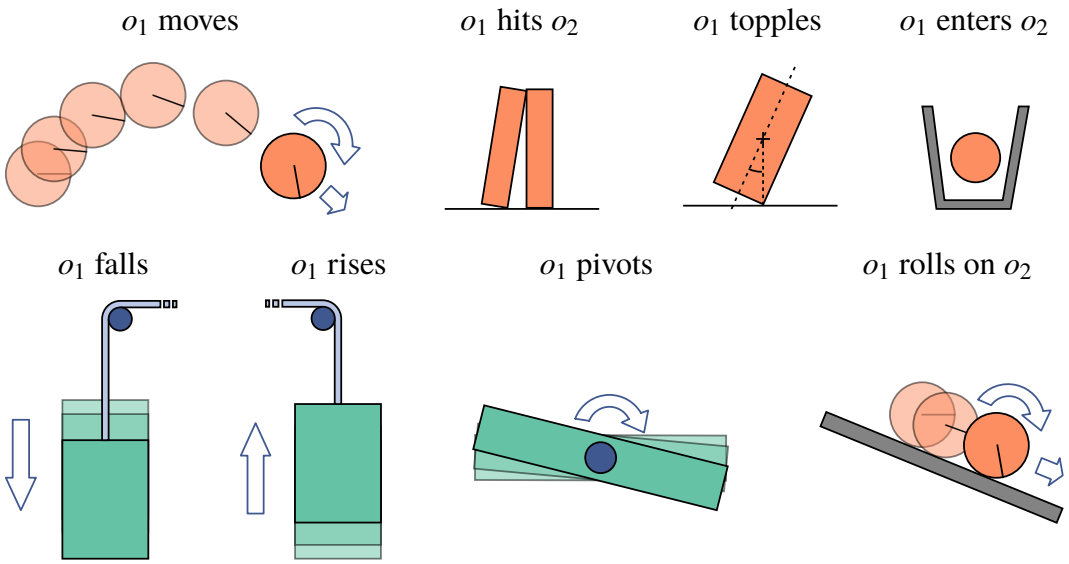


Figure 4.5: Event types. The above events are supported in our implementation. Each event's condition $c_e(t)$ is a function of the spatial transform (and corresponding time derivative) of the target object(s) at time t. Events may also have a negated version (e.g., "o_1 stops" being equivalent to "o_1 does not move"). See Appendix C for details.

`failure` is immediate; hence, in such cases, `success` merely means that the event has started. In practice, we found this formulation expressive enough for our needs.

Events are tied together as nodes of the causal graph, which is a directed acyclic graph with a single root node (i.e., only one starting event) and one or more terminal branches. Using a graph rather than a single timeline allows to account for events happening in parallel in more complex scenarios (see Figure 4.6). Each edge (e_i, e_j) enforces a temporal ordering of the two events it connects. Hence, for instance, if the ball was to fall on the last domino instead of the first, the causal graph would be violated because the intermediate expected events have not happened. We note that such a causal graph is not necessarily a tree: two branches may converge, signifying that *all* parent events need to happen *before* the current one. The entire scenario reaches its *termination* when either (i) the last event of each branch has been reached (global `success`), or (ii) at least one event has timed out (global `failure`).

Design space. The design space D contains different realizations, or *instances*, of a scenario $\mathbb{S}$. In this work, we assume that all the primitives' shape and physical parameters $\{\Theta_i\}$ (defined above) are fixed in advance by measuring and modeling

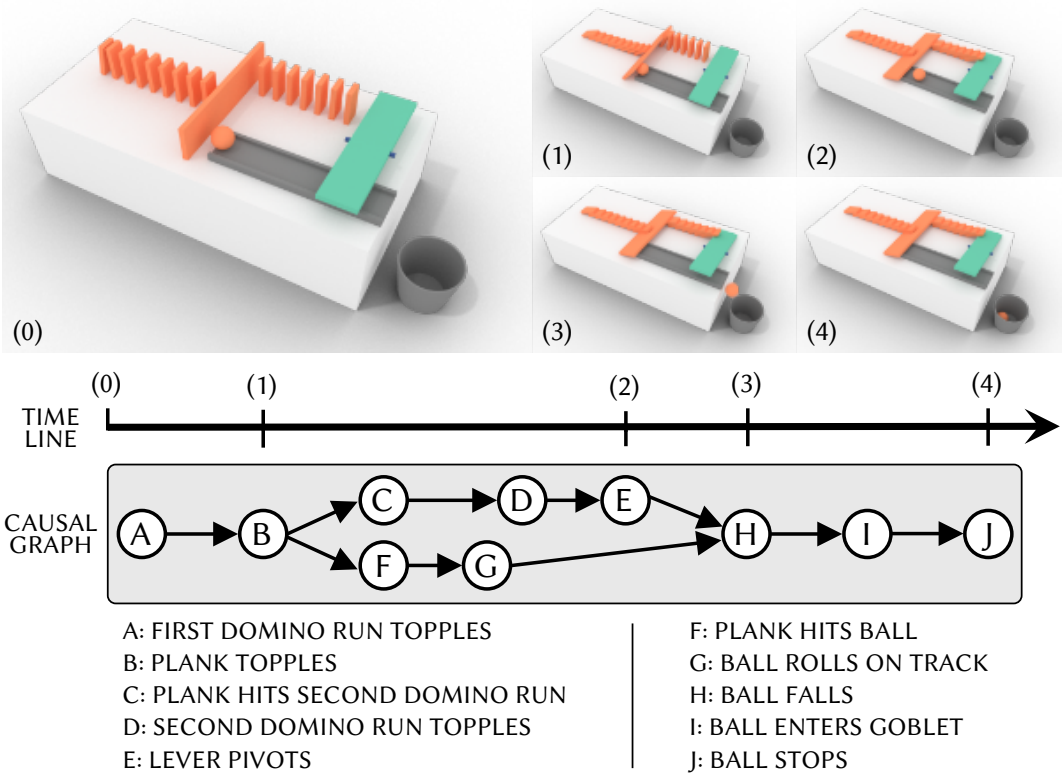


Figure 4.6: **BRANCHING scenario.** A first domino run (top left of the view) topples and hits a plank, which in turns triggers two parallel branches: on one side, a second domino run topples and falls on a lever, which pivots; on the other, a ball rolls on a track, passes below the now-raised lever, and falls into a goblet. As in Figure 4.3, snapshots are matched with events from the causal graph. The two arrows pointing towards event H mean that both E and G need to have happened for H to happen; i.e., the ball can only fall if it started rolling and the lever was raised before the ball reached it.

preexisting objects. We also assume that all the objects are initally at rest. The design space D is composed of the remaining degrees of freedom, i.e., the layout parameters (translation and rotation) of each object relative to its parent in the scene graph. In other words, a scenario instance $\mathbf{x} \in D$ corresponds to a specific initial configuration of the scene S. Therefore, in the general case, $D = SE(3)^m$; in practice however, some layout parameters can be frozen (e.g., if an object's center always lies in a given plane). This results in d parameters, each in a user-defined range $[a, b]$. We automatically normalize each parameter range so that $D = [0, 1]^d$.

Using a deterministic physically-based simulator, we can further structure the design space as follows (see Figure 4.7): first, some regions are forbidden a priori

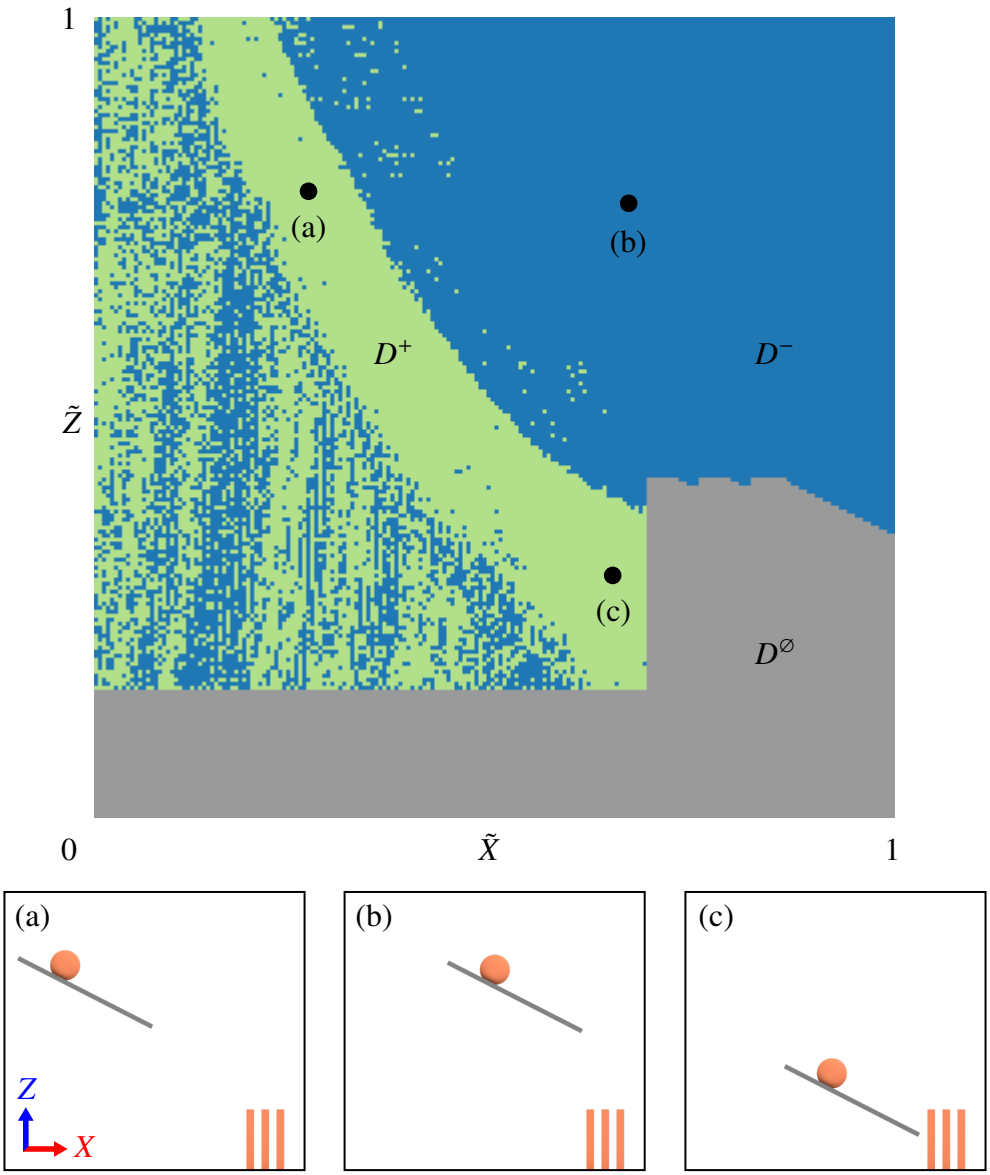


Figure 4.7: Design space. Simulated outcome of the SIMPLE scenario (see Figure 4.3) across a slice of its design space D, divided into success (D^+, green), failure (D^-, blue) and impossible ($D^\varnothing$, grey) regions. Points (a-c) represent three different initial configurations of the scene, only changing the Cartesian coordinates of the center of the track X and Z (normalized as $\tilde{X}$ and $\tilde{Z}$ in D). For (a) and (c), the ball hits the top left edge of the first domino, triggering the fall of the entire domino run, while for (b) the ball misses the run entirely. Region $D^\varnothing$ is a clear cutout of the track intersecting either the floor or the dominoes. Under the relatively smooth central component of D^+, we observe an extremely complex pattern of successes and failures corresponding to the various ways in which the events can unfold, mainly depending on how the ball hits the first domino. This pattern could be an intrinsic property of the scenario, an artifact of the simulation, or a combination of both.

(i.e., before simulation) because they are not physically feasible: typically when two distinct rigid bodies intersect each other at t_0. They form the *impossible region* $D^{\varnothing}$. Second, the physically feasible space is divided between the *success region* D^{+} and the *failure region* D^{-}, which have no explicit representation in the general case, but can be approximated by sampling scenario instances and simulating them under the supervision of the causal graph. Thus, $D = D^{\varnothing} \cup D^{+} \cup D^{-}$, with some of these regions potentially disconnected.[3] It is important to note that this global partitioning of the design space merely *indicates* rather than *explains* the validity and outcome of a scenario instance.[4] In other words, the physical space and time of the simulation are condensed into a single datum. Thus, two points may be neighbors in D^{-}, and yet have failed for different reasons (e.g., dominoes not toppling, or toppling in the wrong order). This may partly explain the complex pattern seen in Figure 4.7 as a *superposition*, in the same space, of different causes of failure. Likewise, two neighboring points in D^{+} may result in visibly different object behaviors, and yet, still lead to a successful outcome. Both versions should be equally acceptable as long as the causal graph is sufficiently detailed by the user. In this case, finding a satisfactory layout could simply be a matter of randomly sampling the design space until a successful instance is found. When *uncertainty* is taken into account, however, a successful scenario instance may become less desirable than its neighbor.

Uncertainty and robustness. To partition the design space as shown in Figure 4.7, points were sampled along a dense grid and labeled as impossible, global `failure` or `success` according to the response of a computer model. The inputs of this model are a combination of the objects' design parameters (kept fixed during this sampling) and the layout parameters. Model predictions of physical phenomena are inherently uncertain. This uncertainty enters models in the form of approximations, errors and unavoidable variability. For instance, in our scenario:

[3] Although, ultimately, we are only interested in approximating D^{+}, keeping $D^{\varnothing}$ and D^{-} distinct is useful to separate soft from and hard constraints in the optimization step of our method (see Section 4.7).

[4] The explanatory power of such a partioning can be improved by dividing a given scenario into sub-scenarios, as is done in Section 4.6. This approach allows our algorithm to efficiently focus on the most critical sources of failure.

- objects are measured and weighed with an accuracy and precision that depend on the user and the measuring tools available;
- likewise, in the physical realization of a given design space point, objects would be laid out with an accuracy and precision that depend on the user and how the layout is communicated to them;
- objects coming from a set (e.g., dominoes) are assumed to have the same shape and weight;
- objects with a relatively simple convex shape (domino, ball) are idealized as regular solids;
- objects with a non-convex shape (e.g., cup) are approximated as unions of of convex shapes (for efficient simulation);
- some parameters of the model cannot be measured experimentally because they are not physical quantities (e.g., the per-object friction coefficient in Bullet Physics [27]), and are left to their default value;
- the model approximates physical phenomena (e.g., contact forces in Bullet Physics are implicitly handled by resolving non-penetration constraints [27]);
- the simulation time is discretized (and the objects' behavior may change depending on the time step);
- the simulation, like any finite-precision floating point computation, is prone to numerical errors;
- the model is deterministic, whereas the outcome of a physical run may change even if the same conditions are repeated;

Analyzing and quantifying uncertainty is a fundamental problem accross many scientific and engineering fields. Often it is sought to reduce the uncertainty surrounding an output variable (or set of variables), whether this variable is measured, minimized, maximized, or made to reach a target value. This can be achieved in two main ways: (i) reducing the uncertainty in the input variables or the model itself, or (ii) reducing the influence of the uncertain input variables on the output variable(s). In this work, we focus on the latter approach, i.e., increasing the *robustness* of chain reaction contraptions to the various sources of uncertainty mentioned above.

Uncertainty and robustness can be described in different ways [90]. Two of the most common representations are bounds (amounting to a worst-case analysis) and probability distributions. The latter is more suited to deal with the highly non-linear and near-chaotic behavior of chain reaction contraptions.[5] In such a framework, the uncertainty of a variable can be quantified by its variance, and the robustness of the output increases when the contribution of the inputs' variance to the variance of the output is reduced. Our first key assumption regarding robustness is the following: any change of a *single* input that results in a decrease of the variance of the output involves a reduction of the contribution of the variance of *several* inputs to the variance of the output. An example may be useful here. Let us consider an initial configuration of our SIMPLE scenario such that the ball hits the top left edge of the first domino in the simulation. In the corresponding physical experiment, it would not take a large perturbation for the ball to miss the domino entirely. The variance of the physical outcome depends significantly on the variance of the track's position and orientation, as well as the variance of the domino's position and height. Changing *any* layout parameter so that the ball collides slightly closer to the center of the domino in the simulation will therefore reduce the variance of the physical outcome by making it less sensitive to the variance of *all* these inputs.

When the output variable is binary, this reduction can also be expressed in terms of probability. Let us define O as the random variable that takes a value of 1 if the outcome of a physical run is successful, and 0 otherwise. O is a Bernoulli random variable with probability $p := \Pr(O = 1)$, which is the *probability of physical success*. It can be shown that the variance of O follows $\mathrm{Var}[O] = p(1 - p)$. This variance is maximal for $p = 0.5$ and minimal for $p = 0$ or 1. As long as $p \geq 0.5$, any increase in p leads to a decrease in the variance. Therefore, p is a suitable candidate

[5]The bound representation, on the other hand, is typically more suitable in deterministic contexts, and when the desirable region has a less complex structure than D^+ in Figure 4.7. Nevertheless, it can still give us a useful visual intuition of our objective. In robust optimization, the *stability radius* model associates the robustness of a given point to the radius of the largest ball centered at this point that is included in the region satisfying the requirements (i.e., D^+ in our case). The most robust point is therefore the one that lies inside D^+ farthest from the boundary ∂D^+, because it requires the largest perturbation to leave that region. In our case, the boundary is uncertain, but there is still the intuitive goal of moving "inwards" (in some fuzzy approximation of D^+) to increase robustness.

to measure robustness. We can then reformulate our problem as: finding the combination of layout parameter values that maximizes the probability of physical success. Formally, we wish to solve:

$$\arg\max_{\mathbf{x}\in D} \Pr(O = 1 \,;\mathbf{x}). \tag{4.1}$$

In this formulation, it is important to note that $\Pr(O = 1 \,;\mathbf{x})$ is a *probability mass function* (because O is discrete) *parametrized* by $\mathbf{x}$. Here, $\mathbf{x}$ represents the set of *nominal values* of the layout parameters, which serve as a reference to build the corresponding physical contraption.[6] This probability could be evaluated, for a given $\mathbf{x}$, by assembling and running the corresponding physical contraption several times and computing the expectation of O. Such a method, however, would be extremely time consuming, especially for longer chain reactions. We could also, in principle, use a Monte Carlo method to approximate $\Pr(O = 1 \,;\mathbf{x})$. In practice, however, this would require assigning probability distributions to many uncertain variables, of which some depend on the physical objects being used, some depend on the contraption builder, and others have no physical meaning. Moreover, estimating this probability over the entire design space would require a significant amount of sampling, especially since the simulated success region D^+ is likely to become small as the contraptions get more complex and the number of dimensions increases.

With a second key assumption, we can rely on fewer simulations to approximate the combination of layout parameter values that maximizes the robustness of the output. Let us define $\hat{O}$ as the random variable that takes a value of 1 if the outcome of a simulation is successful (i.e., global `success`), and 0 otherwise. Moreover, let $\mathbf{X}$ be a random sample of points in the design space D, and $\mathbf{y}$ be the vector of values of $\hat{O}$ for each point in $\mathbf{X}$ after simulation. We make the following key approximation:

$$\arg\max_{\mathbf{x}\in D} \Pr(O = 1 \,;\mathbf{x}) \approx \arg\max_{\mathbf{x}\in D} \Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y} \,;\mathbf{x}\right). \tag{4.2}$$

[6]The actual values of the layout parameters, on the other hand, are random variables, as are shape dimensions and other physical properties. Although the probability of physical success is implicitly conditioned on the value taken by these random variables, we omit them to keep the notation simple.

It is important to note the conceptual shift happening here. On the left-hand side, the value taken by O is a priori unknown because of the various sources of uncertainty mentioned above. Meanwhile, the value taken by $\hat{O}$ for a specific $\mathbf{x}$ is only unknown until the simulation is run. Instead of being simulated, however, it can be *inferred* from the available simulation data. While in many applications, this inference is used to save simulation costs, our method actually takes advantage of the resulting uncertainty. Our key assumption is that we can use this probability of *simulated* success[7] to approximate the point $\mathbf{x}$ that maximizes the probability of physical success.[8] Note that we do not assume that the probabilities of physical and simulated success are *equal* accross the design space; only that their maxima are close in D. In the remainder of this chapter, we will only be concerned with the probability of simulated success parametrized by the layout parameters. We shall henceforth refer to it as *parametric success probability*.

4.4 Overview

4.4.1 User experience

Our method workflow is divided in three steps: scenario definition, probability computation and optimization, and physical realization. Defining a scenario consists in specifying the scene, causal graph, and design space.

The user describes the scene by selecting the primitives, setting their geometric and physical parameters, organizing them as a scene graph (optional), and providing an initial layout (not necessarily a successful one). Setting the fixed parameters requires at least a few measurements (e.g., size and weight). The user then indicates a causal graph by choosing events relative to one or several primitives from a preexisting library, and connecting them by directed edges. Some events may also depend on physical parameters that can be tweaked, such as the minimal velocity required to consider an object to be moving. Lastly, the user specifies the design

[7] Which is a probability mass function *conditioned* on the data and *parametrized* by the nominal layout parameters $\mathbf{x}$.

[8] To borrow terminology from Kennedy and O'Hagan [66], we replace parameter uncertainty, model inadequacy and residual variability by *code uncertainty*.

space in terms of ranges of values for the six layout parameters (position and orientation) of each primitive. Parameters with no range are locked to their initial values. We note that specifying a scene hierarchy in the first step can simplify the ranges and help avoid the exploration of large irrelevant portions of the design space; just like accurate models however, a sophisticated hierarchy is not strictly necessary. From this input, a parametric success probability is built, allowing to optimize the contraption layout to find a solution robust to uncertainties. The solution is then exported as a printed outline to guide the user during assembly (see Appendix D).

4.4.2 Algorithm overview

The core of our method is the efficient computation of a success probability parametrized by the layout $\mathbf{x}$. We approach this problem as a classification task, where an estimator is trained on simulation data to predict the global `success` or `failure` of a run given $\mathbf{x}$. In Section 4.5, we propose algorithms to efficiently find successful points in the design space, train the classifier and improve its accuracy via *active learning*. Section 4.6 then demonstrates how this method can scale in high-dimensional design spaces using a divide-and-conquer method where the *global* probability of success is decomposed into conditional probabilities of success of *individual* events. Each of these components is restricted to the design parameters that really influence the corresponding event, thus reducing the dimensionality of their respective design subspace. Finally, in Section 4.7, we take the scenario instance with the highest success probability and refine it using a probability-based global energy that we minimize under physical validity constraints.

4.5 Computing the parametric success probability

We consider a scenario $\mathbb{S}$ where objects are laid out according to a vector of layout parameters $\mathbf{x} \in D$, with $D = D^{\varnothing} \cup D^{+} \cup D^{-}$. Our goal is to build an approximation of the parametric success probability (PSP) defined as $\Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right)$, using the data $(\mathbf{X}, \mathbf{y})$ provided by the simulator, and without resorting to densely sampling the entire design space.

We build the PSP estimator indirectly by training a classifier to predict whether

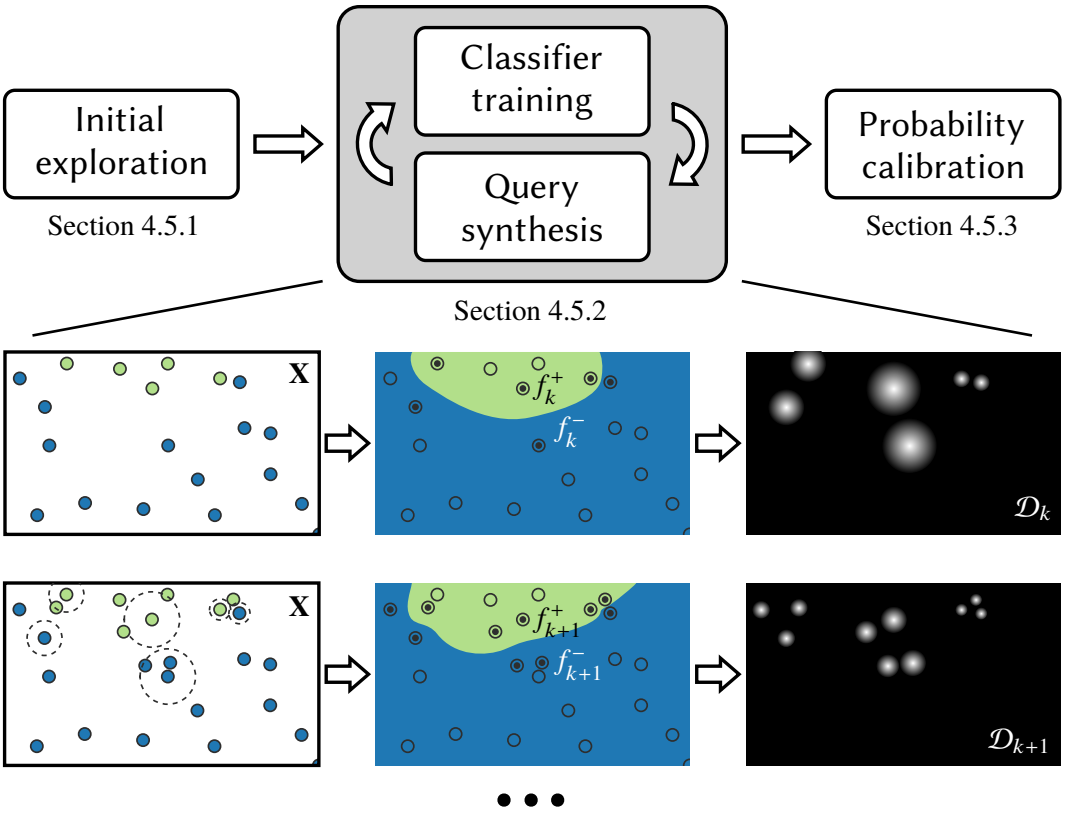


Figure 4.8: Building the PSP. Top: main steps of our global PSP approximation method. Bottom: detail of the active learning loop: at iteration k, we use the current dataset $\mathbf{X}$ (left) to train an SVM (middle; classification shown as background colors). The new support vectors (black dots) are used to build a new distribution $\mathcal{D}_k$ (right; probability density shown in white) that encourages additional sampling where the classifier is the most uncertain (i.e., near the boundary).

a point $\mathbf{x} \in D \setminus D^{\varnothing}$ belongs to the class "$\hat{O} = 1$". Our estimator is a Support Vector Machine (SVM) classifier whose decision function is calibrated after training to obtain a probability parametrized by $\mathbf{x}$. We chose SVMs not only for their robustness to overfitting in high-dimensional spaces, but also because they mesh very well with our active learning strategy, as described next (see Section 4.8 for a comparison with baseline methods).

In this section, we consider a single PSP computed on the entire design space. Our method, as shown in Figure 4.8-top, starts with an *initial exploration* of the design space (Section 4.5.1) by adaptively sampling D and running simulations to find a minimal number of successful instances. The main body of the algorithm (Section 4.5.2) then follows an *active learning* strategy in two alternating steps: first,

during *classifier training*, a non-linear kernel SVM is trained on the current dataset to approximate ∂D^+; second, during *query synthesis*, the decision function of the SVM helps identify uncertain regions of the design space which are then probed to augment the dataset. As a final step (Section 4.5.3), we apply a *probability calibration* to map the final SVM score to a class probability for "$\hat{O} = 1$".

During the entire process, new candidate samples are filtered to discard the physically impossible ones (e.g., those where rigid bodies intersect). Physical validity needs not be learned because it is enforced by constraints during layout optimization (see Section 4.7). Valid scenario instances are simulated under supervision of the causal graph, yielding a global `success` ($\hat{O} = 1$) or `failure` ($\hat{O} = 0$) label.

Algorithm 1 Exploration by adaptive sampling.

$\mathbf{X} \leftarrow \emptyset$
$\{n_i^+\} \leftarrow \emptyset$
$\mathcal{D}_0 \leftarrow$ SobolSequence()
$k \leftarrow 0$
enough $\leftarrow$ `false`
while $k \leq k^{\mathrm{e}}$ **and not** *enough* **do**
    $\mathbf{X}_k \leftarrow$ SamplePhysicallyValid($\mathcal{D}_k, N_k^s$)
    $\mathbf{X} \leftarrow \mathbf{X} \cup \mathbf{X}_k$
    // Simulate each sample point to get its number of successful events.
    $\{n_i^+\} \leftarrow \{n_i^+\} \cup \{$GetNumSuccessEvents($\mathbf{x}$) $\forall \mathbf{x} \in \mathbf{X}_k\}$
    if $|\{i : n_i^+ = n\}| \geq N^+$ **then**
        enough $\leftarrow$ `true` // Because $\mathbf{x}_i \in D^+ \Leftrightarrow n_i^+ = n$.
    else
        $I \leftarrow$ ArgNMax($\{n_i^+\}, N^+$)
        $\mathbf{w} \leftarrow \{n_{I_i}^+ \,/\, \sum_{j \in I} n_j^+ \;\forall i \in [1\,..\,N^+]\}$
        $\mathcal{D}_{k+1} \leftarrow \sum_{i=1}^{N^+} w_i \mathcal{N}(\mathbf{x}_{I_i}, \mathrm{diag}(\sigma|\mathbf{b} - \mathbf{a}|))$
        $k \leftarrow k + 1$
    end if
end while

4.5.1 Initial exploration by adaptive sampling

The goal of the exploration stage is to discover an initial number of successful instances N^+ (200 by default). Algorithm 1 details our adaptive sampling method. We iteratively grow a list of physically valid sample points $\mathbf{X} = \bigcup_k \mathbf{X}_k$, where $\mathbf{X}_k$ is the list of N_k^s points (10 by default) drawn from distribution $\mathcal{D}_k$ at step k, until either (i) the number of successful points $|\{\mathbf{x} \in \mathbf{X} : \mathbf{x} \in D^+\}|$ reaches N^+, or (ii) after k^{e}

iterations (500 by default, which was never reached in our experiments). The initial sampling $\mathbf{X}_0$ is drawn from the quasi-random Sobol sequence [129] (with $N_0^s = 500$ by default). We use the causal graph G to orient the sampling towards the most relevant regions of the design space: for each new sample point $\mathbf{x}_i \in \mathbf{X}$, we simulate the corresponding scenario instance and record the number of successful events n_i^+ (between 0 and n, where n is the total number of events); in other words, n_i^+ is the number of causal graph nodes whose state is `success` after simulation of a single instance $\mathbf{x}_i$. Then, we select the top N^+ values from $\{n_i^+\}$, and note I their indices. We use them to build a mixture of Gaussians

$$\mathcal{D}_k \sim \sum_{i=1}^{N^+} w_i \mathcal{N}(\mathbf{x}_{I_i}, \mathrm{diag}(\sigma)), \tag{4.3}$$

with a diagonal factor $\sigma = 0.01$ by default. There is one Gaussian per sample $\mathbf{x}_i$; their weight w_i reflects the relative success of $\mathbf{x}_i$ with

$$w_i = \frac{n_{I_i}^+}{\sum_{j \in I} n_j^+}.$$

This formulation focuses exploration around the current best *partially* successful scenario instances, which effectively helps it reach regions containing full successes even in high-dimensional design spaces. As in reinforcement learning, we can tune the balance between *exploration* and *exploitation*: for instance, a higher σ favors exploration, as points are sampled further from the current best. Moreover, as new successful data points are found, only taking the top N^+ points at each step means that our method progressively favors exploitation of full successes over exploration of partial successes. Lastly, we note that exploration can be made easier by providing a more detailed causal graph, as it yields a finer-grained distinction between partially successful instances.

4.5.2 Classifier training and query synthesis

The goal of this step is to obtain a classifier with sufficient accuracy (90% in our experiments). We iteratively train an SVM and query new design space points until

we reach either the target accuracy or the maximal number of iterations k^1 (5 by default), as illustrated in Figure 4.8-bottom and detailed in Algorithm 2. The list of samples is noted again $\mathbf{X} = \bigcup_k \mathbf{X}_k$, where $\mathbf{X}_0$ is the set of sample points obtained from the initialization step.

Algorithm 2 Classifier training and query synthesis.

$\mathbf{y} \leftarrow$ ComputeLabels($\mathbf{X}$) // Simulate each sample point.
$k \leftarrow 0$
// Initial classifier training
$f_k, \{\hat{\mathbf{x}}_i\}_{i=1}^{v}, U, acc \leftarrow$ TrainEstimator($\mathbf{X}, \mathbf{y}$)
while $k \leq k^1$ **and** $acc \leq 0.9$ **do**
    // Query synthesis
    $\mathbf{w} \leftarrow \{|f_k(\hat{\mathbf{x}}_i)| \;/\; \sum_{j=1}^{v} |f_k(\hat{\mathbf{x}}_j)| \;\forall i \in [1\,..\,v]\}$
    $\mathcal{D}_k \leftarrow \sum_{i=1}^{v} w_i \mathcal{N}(\hat{\mathbf{x}}_i, |f_k(\hat{\mathbf{x}}_i)|U)$
    $\mathbf{X}_k \leftarrow$ SamplePhysicallyValid($\mathcal{D}_k, 10N^s$)
    $I \leftarrow$ ArgNMin($\{|f_k(\mathbf{x}_i)| \;\forall \mathbf{x}_i \in \mathbf{X}_k\}, N^s$)
    $\mathbf{X}'_k \leftarrow \{\mathbf{x}_i \in \mathbf{X}_k : i \in I\}$
    $\mathbf{X} \leftarrow \mathbf{X} \cup \mathbf{X}'_k$
    $\mathbf{y} \leftarrow \mathbf{y} \cup$ ComputeLabels($\mathbf{X}'_k$) // Simulate each new sample point.
    // Classifier training
    $f_k, \{\hat{\mathbf{x}}_i\}_{i=1}^{v}, U, acc \leftarrow$ TrainEstimator($\mathbf{X}, \mathbf{y}$)
    $k \leftarrow k + 1$
end while

Classifier training. Following common machine learning practices, the dataset is first standardized (i.e., transformed to zero mean and unit variance). The SVM classifier has two hyperparameters: C, the regularization parameter, and γ, the inverse radius of influence of each support vector. We automatically select their optimal value from a logarithmic range using stratified 3-fold cross-validation. The accuracy at step k is given by the cross-validation score.

Support Vector Classifiers provide a confidence score (or "decision function") $f : \mathbb{R}^d \to \mathbb{R}$ quantifying the distance of a point $\mathbf{x}$ to the boundary between classes (represented by a set of v support vectors $\{\hat{\mathbf{x}}_i\}_{i=1}^{v} \subset \mathbf{X}$). In the case of a kernel SVM, the score takes the form

$$f(\mathbf{x}) = \sum_{i=1}^{v} y_i c_i K(\hat{\mathbf{x}}_i, \mathbf{x}) + b$$

where K is (in our case) the Radial Basis Function kernel, $\mathbf{y} \in \{-1, 1\}^v$ is the vector

assigning -1 for global `failure` and 1 for global `success` to each support vector, $\mathbf{c} \in \mathbb{R}^v$ is a vector of coefficients, and $b \in \mathbb{R}$ is a constant term.

Query synthesis. In active learning, a learner is able to improve its accuracy by querying an "oracle" for data points that were not part of its original training set [121]. The query, however, comes at a computational cost; the algorithm thus needs to choose its queries wisely in order to improve its performance. In our case, where the oracle is a simulator, any physically valid point in the design space can be queried to obtain a `success`/`failure` label; this problem is called *query synthesis*. A common strategy consists in reducing estimator uncertainty by querying regions of which the learner is the least certain about. For an SVM, this region is easy to find: it lies near the classification boundary, where the current decision function f_k is close to 0. Hence, after training the SVM at step k, we draw samples from the mixture of Gaussians

$$\mathcal{D}_k \sim \sum_{i=1}^{v} w_i \mathcal{N}(\hat{\mathbf{x}}_i, |f_k(\hat{\mathbf{x}}_i)|U), \tag{4.4}$$

where U is the inverse of the diagonal scaling matrix used for standardization. The Gaussians are weighted by $|f_k|$ with:

$$w_i = \frac{|f_k(\hat{\mathbf{x}}_i)|}{\sum_j |f_k(\hat{\mathbf{x}}_j)|},$$

thus giving sampling priority to the farthest support vectors (i.e., where the boundary is most uncertain). The decision function f_k allows to scale the Gaussians to sample the appropriate neighborhood around each support $\hat{\mathbf{x}}_i$. However, since samples are taken in all directions around each $\hat{\mathbf{x}}_i$, it is unlikely that many of such samples will actually lie near the boundary. Therefore, we first sample (without simulating) $10N^s$ points using $\mathcal{D}_k$, and only keep the N^s ones having the smallest $|f_k(\mathbf{x})|$ value.

4.5.3 Probability calibration

While we do not strictly need to compute a probability when there is only one classifier (as the SVM decision function can be maximized directly), the divide-and-conquer method of Section 4.6 requires probabilities to be output by each classifier so that they can be meaningfully combined or compared. The decision function f,

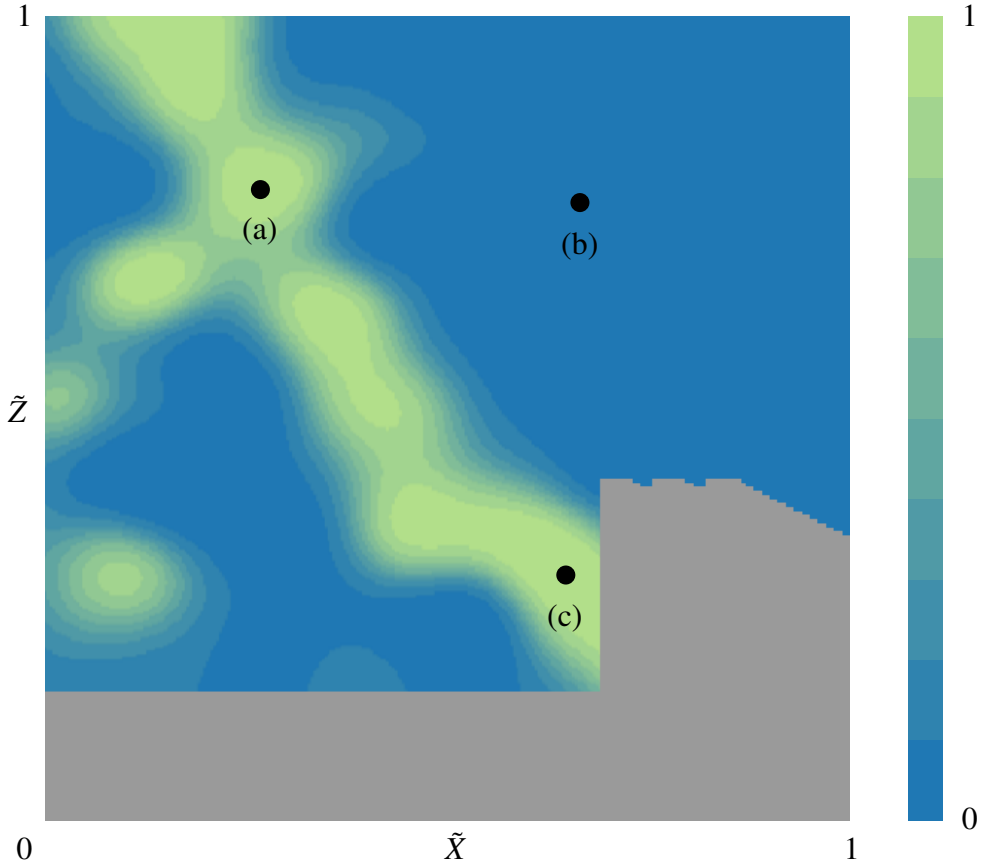


Figure 4.9: Visualization of the PSP. A slice of our learned PSP approximation in the domain $D \setminus D^{\varnothing}$ of the SIMPLE scenario (Figure 4.3). The parameters $\tilde{X}, \tilde{Z}$ and points (a-c) match those in Figure 4.7. The color range is discretized for clearer visualization of the isolevels. We observe that the area of highest probability matches the dense part of D^{+} in Figure 4.7 relatively well, while the complex pattern below this dense part has a low success probability overall.

however, approximates a signed distance to a regularized boundary and not a probability. Nevertheless, the parametric success probability $\Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right)$ can be approximated by applying a continuous transformation to the decision function, following a method known as Platt scaling [106] that fits a logistic regression model to the classifier's scores. Specifically, a maximum likelihood optimization is performed to calibrate the coefficients $\alpha, \beta \in \mathbb{R}$ in

$$\Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right) = \frac{1}{1 + \exp(\alpha f(\mathbf{x}) + \beta)}. \tag{4.5}$$

After this calibration, we can evaluate the PSP of a new scenario instance $\mathbf{x}$ by computing $\Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right)$ (see Figure 4.9).

4.6 Extension to complex causal chains

Chain reactions of reasonable visual complexity can easily depend on several dozens of layout parameters. To help the PSP computation scale to such a high number of dimensions, we propose a divide-and-conquer method where the global PSP for a scenario $\mathbb{S} = \{S, G, D\}$ is broken down into a set of success probabilities of simpler sub-scenarios $\mathbb{S}_i = \{S, G_i, D_i\}$, where G_i is a subgraph of G, and D_i is a subspace of D with dimension $d_i < d$. This inequality is key to the scalability of our method, as it reduces the combinatorial complexity of exploring D and approximating the PSP.

Before detailing our extended pipeline, let us demonstrate how to factorize $\Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right)$. By definition, a scenario is successful if and only if each event happens in the correct order; this is equivalent to each node of the causal graph reaching `success` after its parent(s) did the same. Formally, if we associate to each causal graph event e_i the random variable $\hat{E}_i$ taking a value of 1 if the event's final state after simulation is `success`, and 0 otherwise, we are trying to decompose the joint probability

$$\Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right) = \Pr\left(\{\hat{E}_i = 1\}_{i=1}^{n} \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right), \tag{4.6}$$

where n is the number of events. To do so, let us consider the directed graphical model $\mathcal{G}$ obtained by replacing each node e_i in G by the corresponding $\hat{E}_i$. By construction of the causal graph, the `success` of the parents is equivalent to the `success` of all ancestors; therefore the probabilistic event $\hat{E}_i = 1$, for a given $\mathbf{x}$, only depends on the parents of $\hat{E}_i$. In other words, $\mathcal{G}$ satisfies the local Markov property, expressed as conditional independence:

$$\forall \hat{E}_i \in V(\mathcal{G}) : \quad \hat{E}_i \perp\!\!\!\perp \left\{\mathrm{nd}(\hat{E}_i) \setminus \mathrm{pa}(\hat{E}_i)\right\} \mid \mathrm{pa}(\hat{E}_i)$$

where $V(\mathcal{G})$ is the set of vertices in $\mathcal{G}$, and $\mathrm{nd}(\hat{E}_i)$ and $\mathrm{pa}(\hat{E}_i)$ are respectively the set

of non-descendants and parents of $\hat{E}_i$ in $\mathcal{G}$. It can be shown that for directed acyclic graphs, this property is notably equivalent to the factorization of joint probabilities on the graph nodes into conditional probabilities given the node's parents [75]. In particular, Equation 4.6 yields the PSP factorization:

$$\Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right) = \prod_{i=1}^{n} \Pr\left(\hat{E}_i = 1 \mid \mathrm{pa}(\hat{E}_i) = 1, \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right). \tag{4.7}$$

We call the i-th factor of the above product the $\hat{E}_i$-CPSP (where C stands for "Conditional"). Of course, if we were to approximate each $\hat{E}_i$-CPSP as we approximate the global PSP, the complexity would be multiplied by n, rather than decreased. To effectively reduce it, we observe that given $\mathrm{pa}(\hat{E}_i) = 1$, having $\hat{E}_i = 1$ typically depends on few layout parameters; in other words, the variance of each $\hat{E}_i$-CPSP mostly only occurs in a relatively low-dimensional subspace $D_i \subset D$. Computing this *subspace mapping* (see second block in Figure 4.10) is described next.

First, as a pre-processing step, we identify which $\hat{E}_i$ are *quasi-deterministic*, i.e., nearly always equal to 1 when their parents are. Given $\mathbf{X}$ the sample obtained after initial exploration (Section 4.5.1), which contains a minimum number of globally successful sample points, we compute the expectation of each $\hat{E}_i$ and assign $D_i = \emptyset$ to each $\hat{E}_i$ having $\mathbb{E}\left[\hat{E}_i\right] \geq 0.95$. The corresponding $\hat{E}_i$-CPSPs are set to 1. We measure correlations between the remaining $\hat{E}_i$ and the design parameters using mutual information [28], which is sensitive to linear and nonlinear relationships. In our experiments, parameters were selected if their mutual information with $\hat{E}_i$ was greater than 0.2. Then, to approximate the non-constant factors, we apply the following method, illustrated as the stack of blocks of the extended pipeline in Figure 4.10. For each $\hat{E}_i$-CPSP, we consider the subgraph G_i containing only e_i and its ancestors. We run the training and query synthesis loop (Section 4.5.2) using G_i, with two slight modifications: (i) to satisfy the conditional probability in Equation 4.7, we only keep the sample points satisfying $\mathrm{pa}(\hat{E}_i) = 1$. We know that such points exist in the initial set for each $\hat{E}_i$, since some of the initial points are globally successful over G. (ii) During query synthesis, we restrict sampling

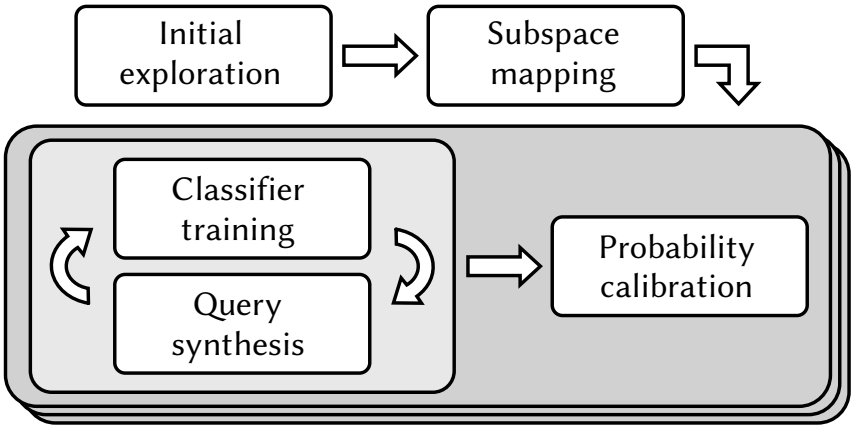


Figure 4.10: Extended PSP building pipeline. We propose an extended version of our PSP building to scale with a high-dimensional design space.

to the corresponding D_i by simply taking the indices of the parameters not in D_i, and setting their scale factor in U to 0: therefore, only the parameters in D_i have non-zero variance. Lastly, we calibrate each probability as in Section 4.5.3.

As a result of the above steps, the PSP approximation can be computed with a significant complexity reduction as

$$\Pr\left(\hat{O} = 1 \mid \mathbf{X}, \mathbf{y}\,; \mathbf{x}\right) \approx \prod_{i=1}^{n} r_i(\mathbf{x}), \tag{4.8}$$

where

$$r_i(\mathbf{x}) = \Pr\left(\hat{E}_i = 1 \mid \mathrm{pa}(\hat{E}_i) = 1, \phi_i(\mathbf{X}), \mathbf{y}\,; \phi_i(\mathbf{x})\right)$$

and $\phi_i : D \to D_i$ is the subspace mapping.

4.7 Layout optimization

Once the PSP has been computed, we take the sample point with the highest success probability as our most robust current solution. Although this design is indeed already quite robust, we further refine it by applying a nonlinear optimization. While the factorization could allow us, in theory, to optimize each $\hat{E}_i$-CPSP separately, in general they are not separable because their subspaces D_i overlap. Instead, we

aggregate all components into a global energy to solve

$$\min_{\mathbf{x}\in D} \mathcal{E}(\mathbf{x}) \quad \text{subject to} \quad \mathbf{C}^{\varnothing}(\mathbf{x}) \geq \mathbf{0} \tag{4.9}$$

with

$$\mathcal{E}(\mathbf{x}) = -\mathcal{S}_\alpha \circ \mathbf{r}(\mathbf{x}), \tag{4.10}$$

where $r_i(\mathbf{x})$ is the $\hat{E}_i$-CPSP approximation given in Equation 4.8, and the function $\mathcal{S}_\alpha : \mathbb{R}^n \to \mathbb{R}$ is the smooth minimum [74]:

$$\mathcal{S}_\alpha(\mathbf{z}) = \frac{\sum_i z_i \exp(-\alpha z_i)}{\sum_j \exp(-\alpha z_j)}$$

with $\alpha \in \mathbb{R}^+$ controlling the importance of the smallest component of $\mathbf{z}$. This choice (rather than taking the product, as in Equation 4.8) comes from the observation that a chain is only as strong as its weakest link. This entails that priority should be given to maximizing the minimal $\hat{E}_i$-CPSP value, rather than maximizing their product.

The constraint vector $\mathbf{C}^{\varnothing}$ ensures that the design stays physically valid. It aggregates (i) penetrations between distinct rigid bodies in the scene, and (ii) primitive-specific constraints, such as ensuring that the layout of a rope-pulley is compatible with the rope length. The former is easily obtained from the rigid body simulator, as penetrations are needed to compute the reaction force between colliding shapes [27]. While we could have learned invalid configurations when computing the PSP, thus integrating the constraint into the energy, we chose to explicitly enforce physical validity during optimization for two reasons: first, validity would not have been guaranteed since we only approximate the PSP, and second, *impossibility* and *failure* are two distinct concepts. Indeed, the probability of success does not necessarily decrease as a design $\mathbf{x} \in D^+$ is moved closer to $D^{\varnothing}$: for example, putting two successive dominoes in contact might robustly ensure that both topple.

As described earlier, the initial solution is the sample point with the highest PSP value. Assuming that this guess is close enough to the global minimum, we use Sequential Least-Squares Quadratic Programming [71] to find the optimal design.

4.8 Case study results

4.8.1 Implementation

Our framework was implemented in Python 3.5. For each primitive type, we implemented a parametric model for both visualization (using OpenSCAD) and simulation (using Bullet Physics). Most computations, including optimization, are done with NumPy and SciPy, while Scikit-learn is used for the SVM classifier and the other machine learning tools. Our graphical interface (described next) uses the Panda3D game engine. Code and additional details (including package versions) are available online.[9]

Interface. Our graphical interface allows users to define the scene and causal graph (see Figure 4.11). They can instantiate the primitives described in Section 4.3 and define the initial layout. We provide specific tools to help designing the most complex primitives: for instance, domino runs can be generated by simply drawing a path along which dominoes are automatically distributed. Simulation can be run in real time, providing visual feedback during design. Once the scene is complete, users define the causal graph by instantiating events, linking them to the objects in the scene and drawing directed edges between them, and finally specifying necessary parameters such as design space ranges. We note that the specific layout designed in the GUI does not need to be a fully successful one: all that matters is that there is a successful region somewhere in the design space. Additional configuration parameters (such as event specific parameters and design space ranges) are currently input through a text file, but could be integrated into the interface.

Export. The final optimized layout is automatically exported as PDF sheets to be printed, to provide guidance during assembly. The outline is obtained by projecting the convex hull of each object onto a vertical or horizontal plane, depending on the user's need. A pattern of grey lines is added to the background to help join the paper sheets after printing. Guidance patterns for the examples presented in this chapter are provided in Appendix D.

[9] `https://geometry.cs.ucl.ac.uk/projects/2019/causal-graphs/`

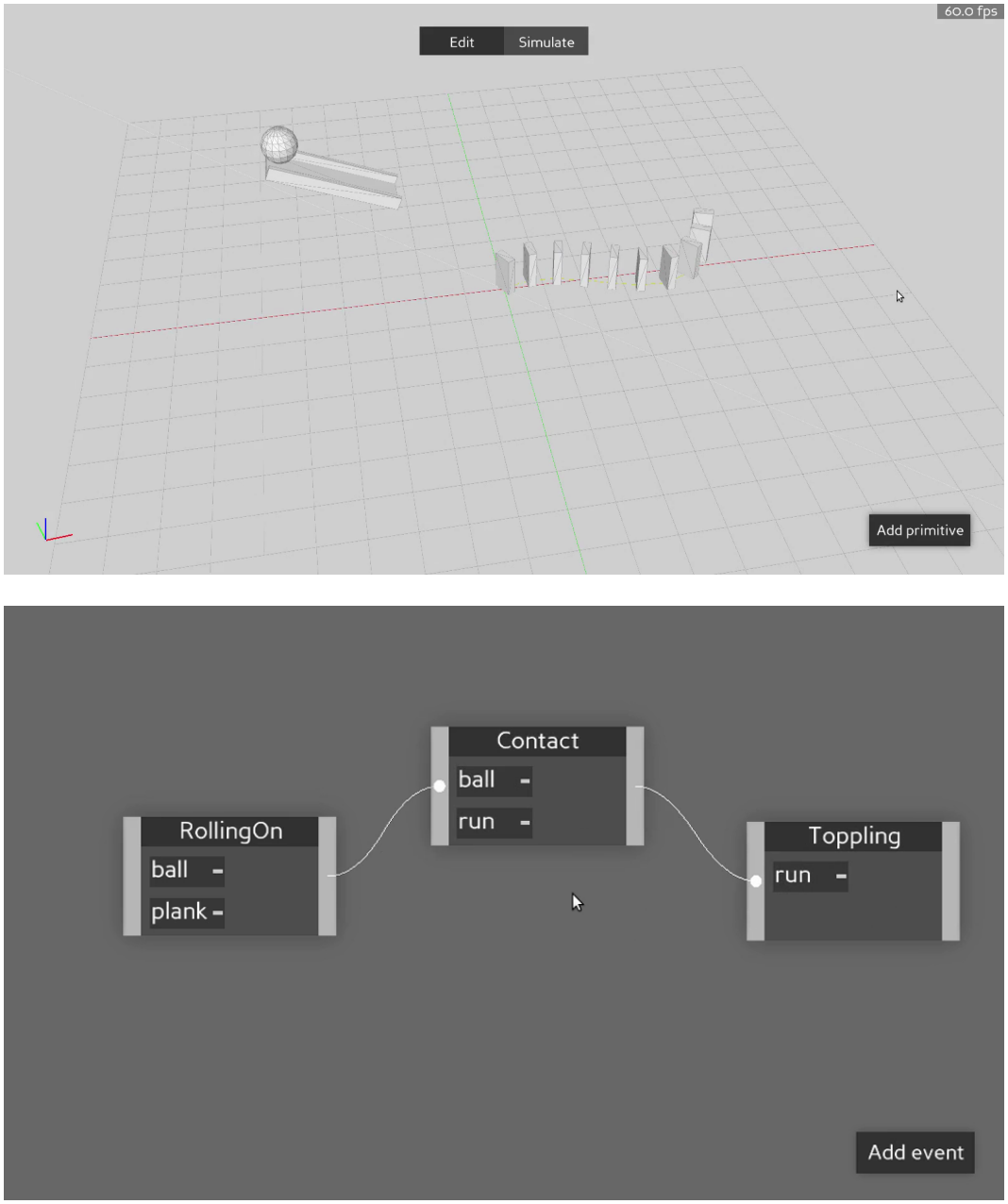


Figure 4.11: Graphical user interface. The top image shows our main 3D modeling interface. The user can add primitives, specify their dimensions, and move them in the scene. The contraption can be simulated at any time. The bottom image shows our causal graph design interface. New events can be added, linked to existing objects in the scene, and oriented edges can be drawn between them.

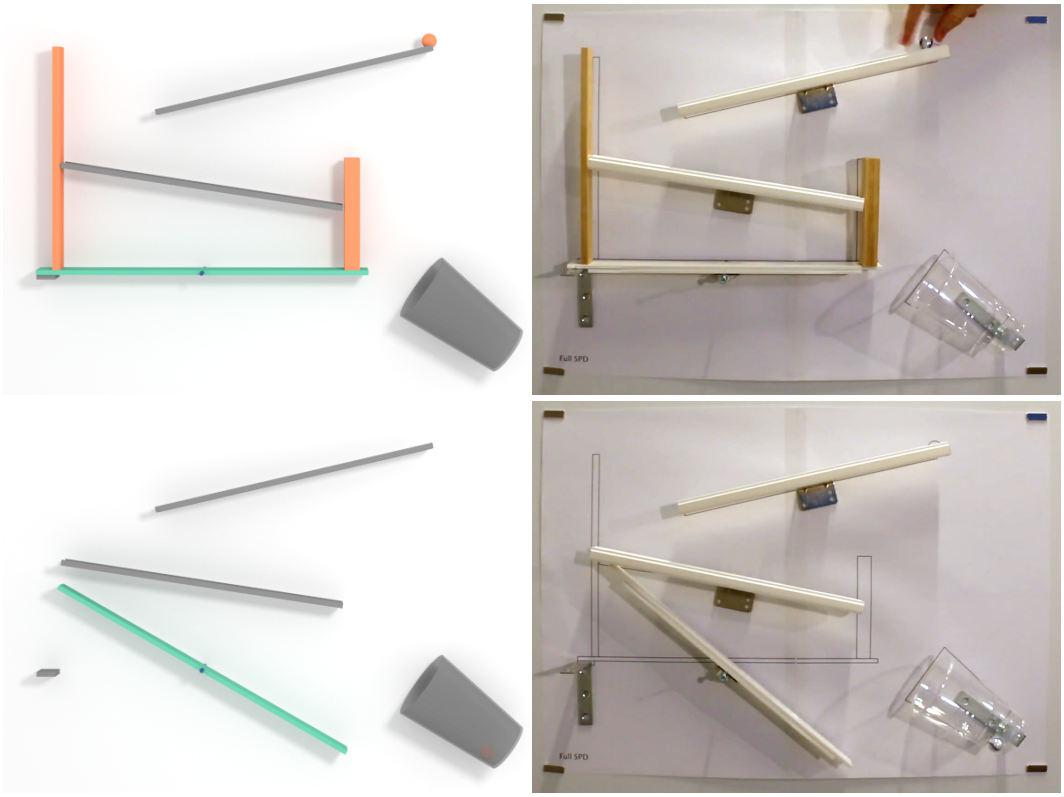
4.8.2 Qualitative evaluation

We designed, implemented, and physically realized a number of scenarios to validate our pipeline. We present here a selection of four examples, focusing on those most challenging due to complex movements and/or event synchronizations. Note that simple domino runs following long low-curvature paths are easy to design (as commonly seen in online videos) and hence were avoided in these experiments. The four presented sequences are called BALLRUN, CAUSALITYSWITCH, LONGCHAIN, and TEAPOTADVENTURE, in increasing order of complexity. The first two resulted in a successful real-life run after a single try; the others, due to their higher complexity, required a more careful adjustment of the parts to the printed layout and succeeded after 4–5 trials. We note that this number is much lower that the dozens of trials usually shown in behind-the-scenes videos found online. In this section, we describe each scenario at a high level, while further details are provided in Appendix C and video clips are available online (see link in Section 4.8.1).

In BALLRUN, the goal is to get the ball to roll down the tracks and fall into the cup. However, a wooden plank blocks the entrance of the cup. Synchronization is needed along the causal graph to realize the following sequence: the ball hits the first wooden block to get the lower support rotating, but the ball has to travel slowly enough to allow the other wooden block to fall, thus opening the pathway to the target cup (see Figure 4.12 but best seen in the video).

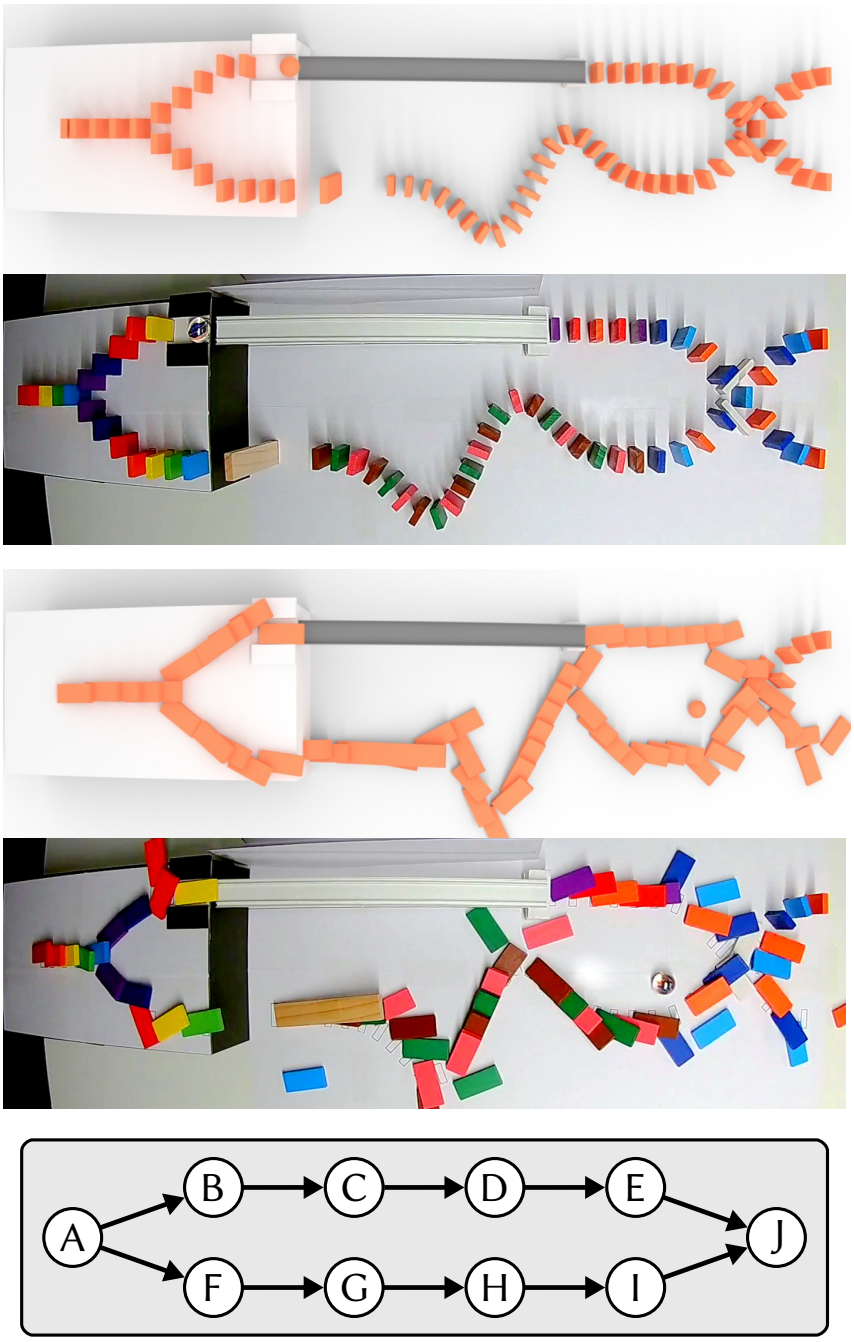
The CAUSALITYSWITCH contains two longer chains running in parallel until a domino "switch" (shown in Figure 4.14) allows only the fastest path to go through by blocking the way of the other. One path is a wave-like chain of dominoes, while the other involves a ball rolling on a track. This experiment demonstrates that we can choose to optimize for either side to be the fastest by modifying the causal graph accordingly. Figure 4.13 shows the causal graph along with the final and initial state with the "ball" side successfully reaching the switch first. Both versions endings are shown in Figure 4.15 and the online video (link in Section 4.8.1).

The LONGCHAIN is a long linear sequence of events. Under the weight of the box's contents, a lever pivots to topple the domino run that, in turn, nudges the ball



A : BALL ROLLS ON TOP TRACK
B : BALL HITS LEFT PLANK
C : BALL ROLLS ON BOTTOM TRACK
D : LEFT PLANK TOPPLES
E : LEVER PIVOTS
F : RIGHT PLANK FALLS
G : BALL ENTERS GOBLET
H : BALL STOPS

Figure 4.12: BALLRUN scenario. A ball rolls down a sequence of tracks into a goblet. To clear the path, it needs to trigger the fall of the right plank by hitting the left one, with the bottom lever pivoting *before* the ball reaches the goblet. Top: initial state; bottom: final state. See online video (link in Section 4.8.1).



A : BRANCHING DOM. RUN TOPPLES
B : LEFT BRANCH DOM. HITS BALL
C : BALL ROLLS ON TRACK
D : BALL HITS STRAIGHT DOM. RUN
E : STRAIGHT DOM. RUN TOPPLES
F : RIGHT BRANCH DOM. HITS PLANK
G : PLANK TOPPLES
H : PLANK HITS WAVE DOM. RUN
I : WAVE DOM. RUN TOPPLES
J : RIGHT END OF SWITCH TOPPLES + LEFT END OF SWITCH DOES NOT MOVE

Figure 4.13: CAUSALITYSWITCH scenario. A branching domino run topples and triggers two parallel branches. On one side ("ball run"), a ball rolls down a track and topples a short straight domino run. On the "domino run" side, a plank falls and topples a long, curved domino run. Whichever side is faster (in this figure: the left one) triggers the 'domino switch', closing the path of the other side. See online video (link in Section 4.8.1).

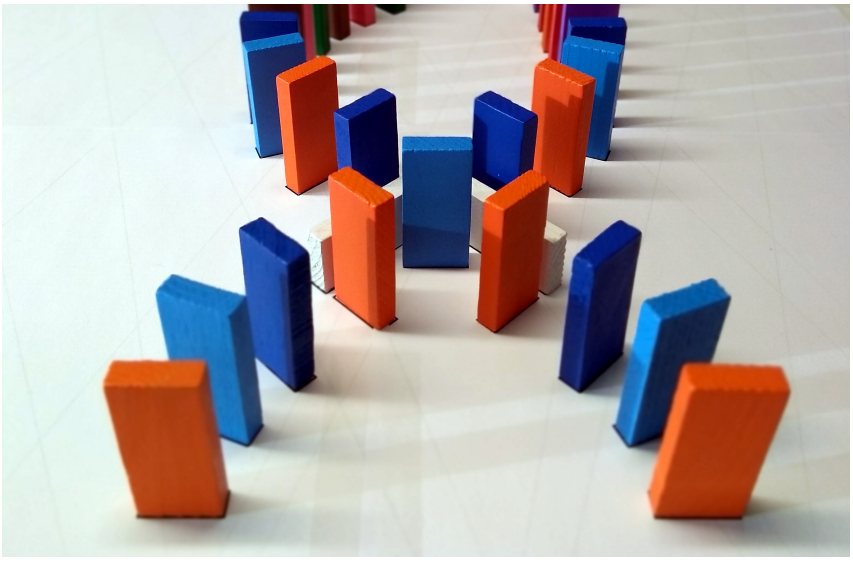


Figure 4.14: Domino "switch". The faster path closes the way of the slower path. E.g., when the left white domino falls, the left orange is pushed out of the path and can't fall anymore.

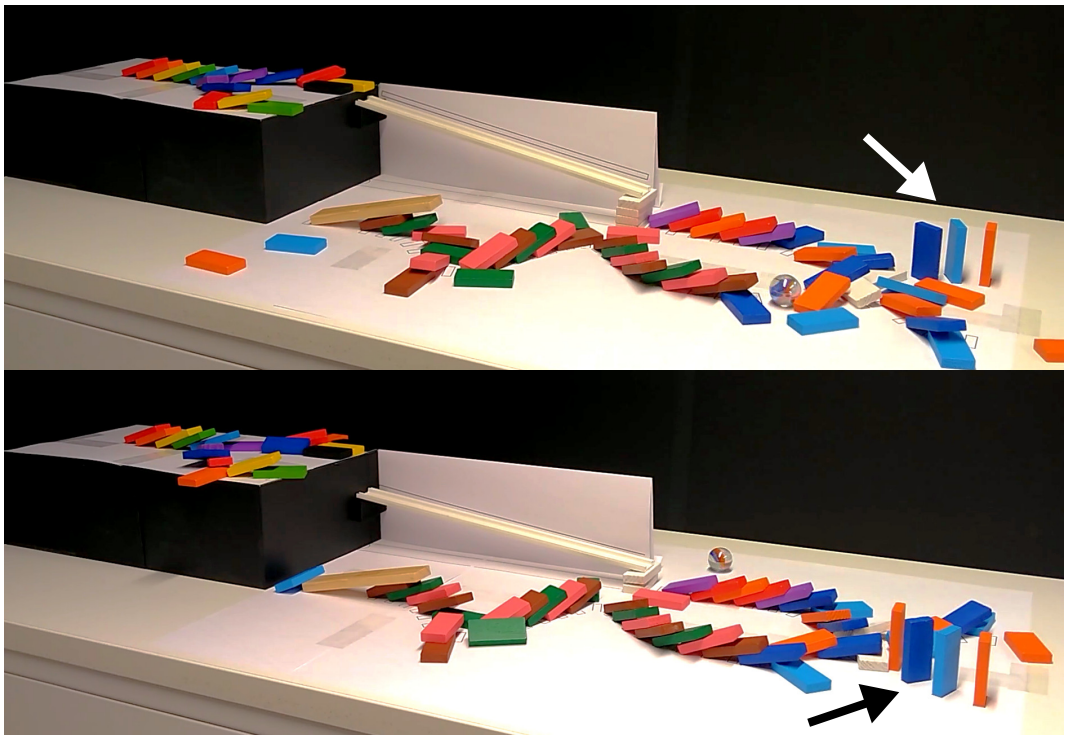
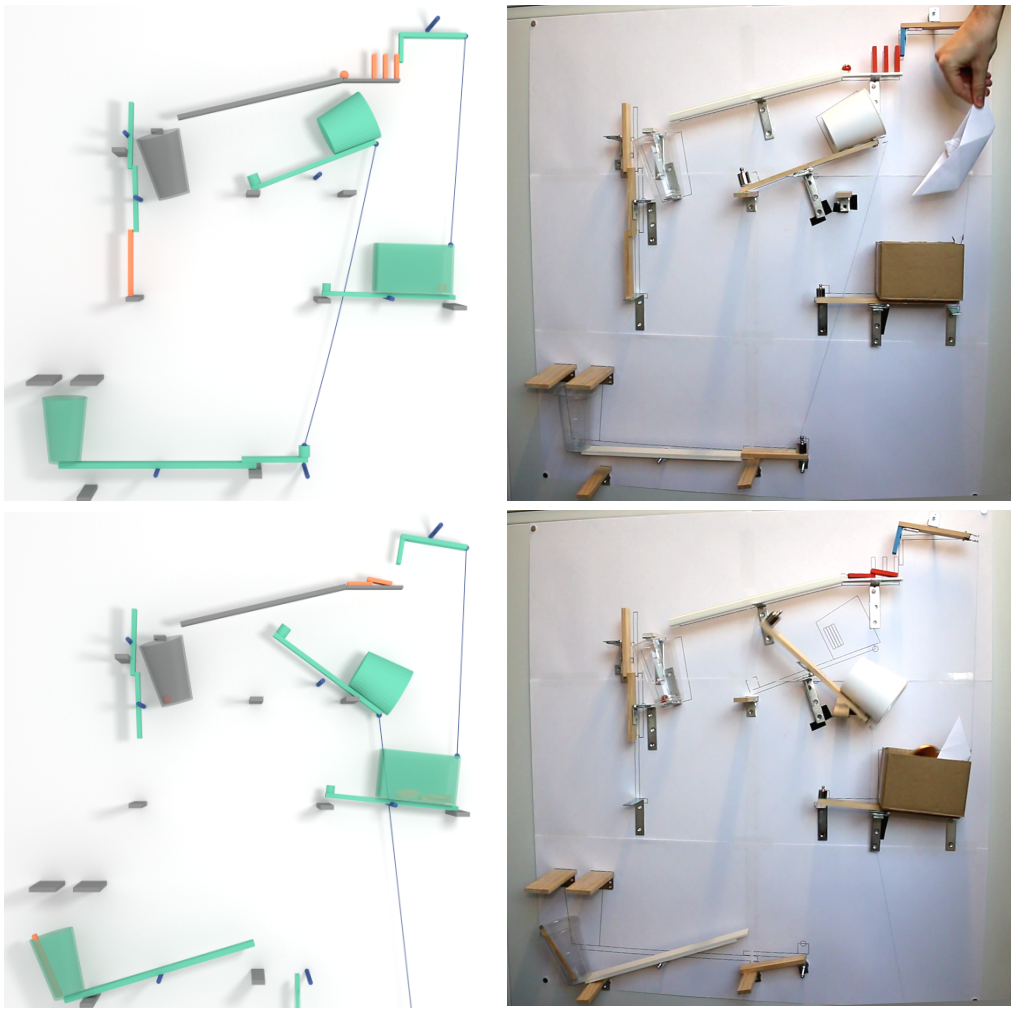
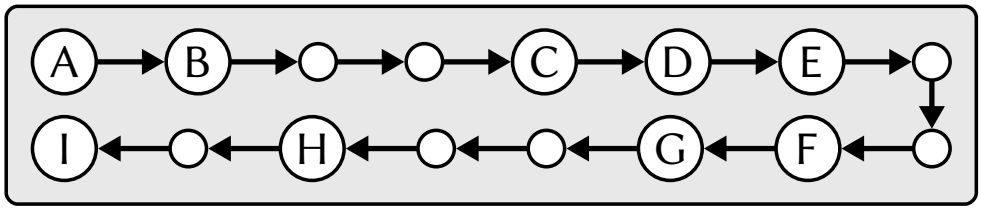


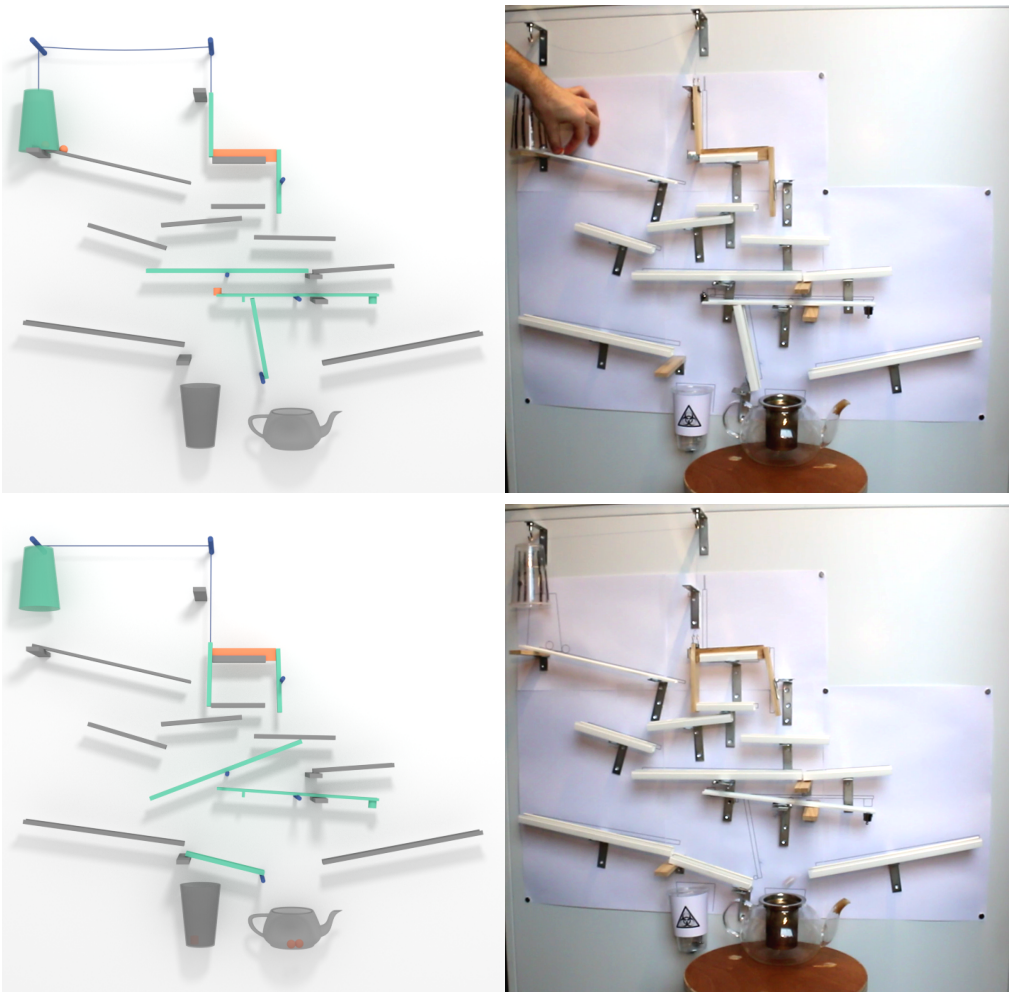
Figure 4.15: CAUSALITYSWITCH endings. In the top figure, the layout has been optimized so that the left side is faster (resulting in the left end still standing, as indicated by the arrow); and vice versa in the bottom figure. We can notably see that the track's position and angle differ.

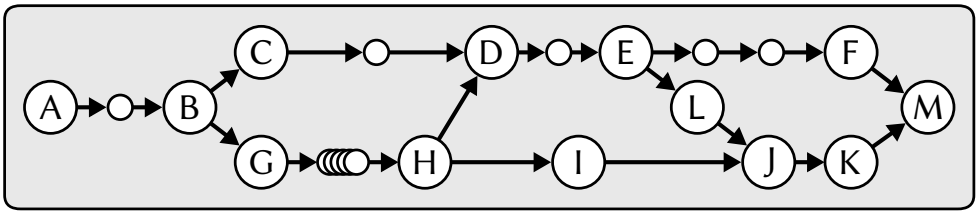




A : BOX LEVER PIVOTS
B : TOP-RIGHT LEVER PIVOTS
C : BALL ROLLS ON TRACK
D : BALL HITS BOUNCER
E : BALL HITS TOP-LEFT LEVER
F : PLANK TOPPLES
G : PLANK ENTERS BOTTOM GOBLET
H : BOTTOM WEIGHT FALLS
I : COINS ENTER BOX

Figure 4.16: LONGCHAIN scenario. A long chain of events that is triggered by the box's weight pivoting an initial lever. Dominoes topple and send a ball onto a track to go hit a lever, which makes a plank topple through a narrow entrance into a goblet. This triggers the fall of the bottom weight, and in turn, the central goblet pivots to let coins fall into the box. See online video (link in Section 4.8.1).





A : BALL 1 ROLLS ON START TRACK
B : BALL 1 HITS GATE LEVER
C : BALL 1 ROLLS ON RIGHT-TOP TRACK
D : BALL 1 ROLLS ON RIGHT LEVER TRACK
E : RIGHT LEVER TRACK PIVOTS
F : BALL 1 FALLS INSIDE TEAPOT
G : GATE FALLS
H : LEFT LEVER TRACK PIVOTS
I : BALL 2 ROLLS ON LEFT-BOTTOM TRACK
J : BALL 2 ROLLS ON BRIDGE
K : BALL 2 FALLS INSIDE TEAPOT
L : BRIDGE PIVOTS
M : BALL 1 MEETS BALL 2 (AND THEY LIVED HAPPILY EVER AFTER)

Figure 4.17: TEAPOTADVENTURE scenario. See Figure 4.2 for initial layout. This figure shows rendered versus assembled layouts for start and end frames of the optimized layout. Ball 1 triggers the fall of the middle gate, releasing ball 2. Ball 2 hits the gate and takes the left route, falling on a lever so that ball 1 can roll underneath. Ball 2 then makes the central weight fall, liberating the bottom bridge. Both balls then meet in the teapot. See online video (link in Section 4.8.1).

onto the track. The ball rolls down the track, bounces on a small platform, and hits a lever, resulting in the orange plank toppling. The plank tumbles and falls through a narrow entrance into a goblet, moving a pivot that makes the bottom weight fall, tugging on the central goblet from which coins fall into the box. Figure 4.16 shows the causal graph along with initial and final states of the optimized layout, while the full run is shown in the video.

The TEAPOTADVENTURE is the most complex example shown in this chapter (see Figures 4.2 and 4.17). As seen from the causal graph, success for this contraption requires a very challenging synchronization between delicate event chains. In short, there are two balls involved (one initially free, and one in a cage), that need to escape the contraption and reach the teapot by opening each other's path along the way. Note that such a sequence is very difficult to manually author without computational guidance as proposed in this chapter. On a lighter note, this kind of scenario illustrates the narrative power of chain reaction contraptions, such as can be seen, e.g., in the Japanese show *PythagoraSwitch*.

4.8.3 Quantitative evaluation

Local and global robustness. We compare the output of different methods with the following measures of robustness. Let $\mathbb{S}$ be a scenario with design space D, and $\mathbf{X} \subset D$ a set of points decomposed as $\mathbf{X} = \mathbf{X}^+ \cup \mathbf{X}^- \cup \mathbf{X}^\varnothing$ (respectively successes, failures and impossible instances). The *local robustness* $\rho_l : D \times [0,1] \to [0,1]$ is defined as

$$\rho_l(\mathbf{x}, \epsilon) = \begin{cases} \frac{|\mathcal{B}_\epsilon(\mathbf{x}) \cap \mathbf{X}^+|}{|\mathcal{B}_\epsilon(\mathbf{x}) \cap \{\mathbf{X}^+ \cup \mathbf{X}^-\}|} & \text{if } \mathbf{x} \in D \setminus D^\varnothing, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{B}_\epsilon(\mathbf{x})$ is the ball of radius ϵ centered at $\mathbf{x} \in D$. In this formulation, ϵ represents a uniform error on the layout parameters, while ρ_l is the success rate in the neighborhood $\mathcal{B}_\epsilon(\mathbf{x})$. The global robustness $\rho_g : D \to [0,1]$ is then defined as

$$\rho_g(\mathbf{x}) = \int_0^1 \rho_l(\mathbf{x}, \epsilon) d\epsilon.$$

Table 4.1: Data for each scenario. The simulation budget was computed by running our method $T = 10$ times and counting the average number of simulations performed.

Scenario	Number of parameters d	Max. simulation time (s)	Simulation budget B
causalitySwitch	4	4	614
ballRun	8	4	1120
longChain	11	5	1915
teapotAdventure	16	8	2420

In practice, the evaluation dataset $\mathbf{X}$ is computed from a relatively dense sample of the design space (with 100K points for causalitySwitch, and 1M points for the three others). Additional details on the computation of ρ_l are provided in Appendix C.

Comparison with baseline methods. We define three baseline methods to compare against our technique:

(B1) Uniformly sample and simulate points in D until the first successful configuration is found.

(B2) Uniformly sample and simulate points in D, compute the local robustness $\mathbf{x} \mapsto \rho_l(\mathbf{x}, 0.1)$ for each, and take the best one.

(B3) Run a Bayesian Optimization with a Gaussian Process prior [122] (initialized with uniform sampling) using $\mathbf{x} \mapsto \rho_l(\mathbf{x}, 0.1)$ as an objective function.

Each baseline is given the same "simulation budget" B computed from our own method: we first run our method T times with a different random generator seed each time (with $T = 10$ in our experiments), and compute B as the average number of simulations carried out. Each baseline is then also run T times until B is reached (or until a success is found for baseline B1). Table 4.1 provides the budget computed for each scenario. The final local robustness curve is the average curve accross the different trials.

Results in Figure 4.18 and Table 4.2 show that our method outperforms the baselines for all four scenarios presented in Section 4.8.2. Interestingly, the Gaussian Process optimization (B3) appears to perform worse than the simpler uniform sampling (B2) in higher dimensions, which could be due to the presence of holes (i.e.,

Table 4.2: Experiments statistics. Both global robustness and processing time are averages of $T = 10$ random trials. The simulation step was 0.002s.

Scenario	Global robustness $\rho_g(\mathbf{x}^*)$				Processing time (s)			
	B1	B2	B3	Ours	B1	B2	B3	Ours
causalitySwitch	0.44	0.59	0.63	**0.65**	**263**	282	463	364
ballRun	0.34	0.35	0.40	**0.48**	59	**52**	786	153
longChain	0.09	0.13	0.13	**0.16**	1499	1597	1956	**636**
teapotAdventure	0.16	0.23	0.14	**0.27**	**536**	**536**	988	779

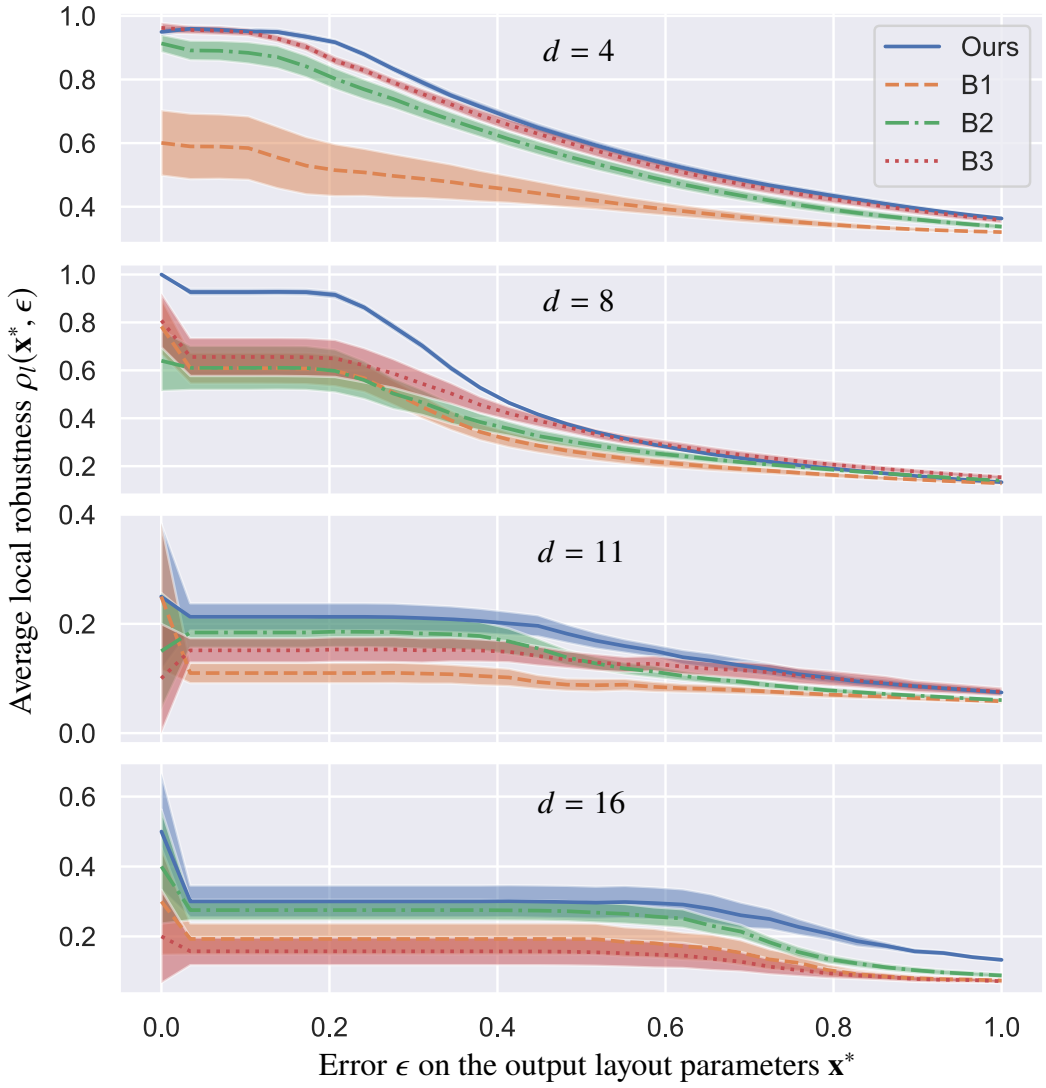


Figure 4.18: Local robustness plots. Average local robustness $\rho_l(\mathbf{x}^*, \epsilon)$ as a function of the error ϵ on the layout parameters $\mathbf{x}^*$ output by each method, for problems of different dimension d. Each curve is surrounded by a standard error of the mean interval (with $T = 10$ runs per method). Scenarios are respectively causalitySwitch, ballRun, longChain and teapotAdventure.

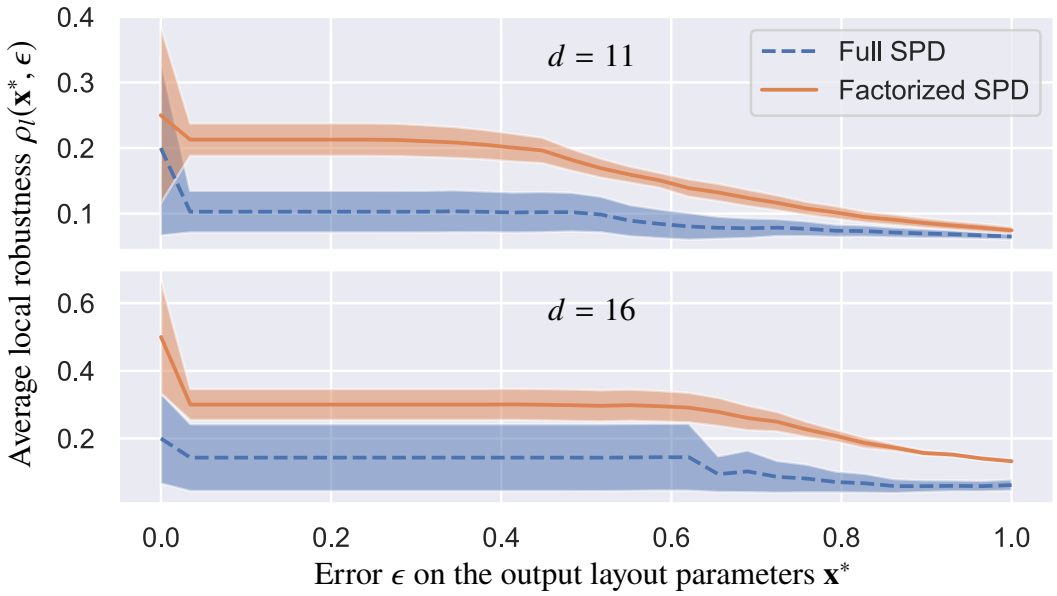


Figure 4.19: Full versus factorized PSP. We use the same metric as in Figure 4.18 to compare two versions of our method for the LONGCHAIN and TEAPOTADVENTURE scenarios. "Full" means that a single SVM was used over the entire design space, while "Factorized" follows the method in Section 4.6.

components of $D^{\varnothing}$) in the optimization landscape. Additionally, we demonstrate the positive impact of PSP factorization in Figure 4.19.

4.9 Summary of the case study

I have presented a framework to help design chain reaction contraptions by optimizing the components' layout for a target sequence of events specified as a causal graph. The method specifically focuses on robustifying the design against modeling and manual assembly uncertainty. At the core of this approach is the computation of a *parametric success probability* (PSP) that provides an approximate measure of robustness to uncertainties. The PSP is efficiently built with a combination of simulation-based search and machine learning techniques, before using it to obtain a robust design layout. Quantitative results showed a significant improvement over baseline methods across a wide range of dimensions, and the method was further validated by physically realizing a set of complex chain reaction contraptions optimized with this technique.

I argue that the results presented in this chapter support two of the main con-

tributions presented in Chapter1: first, this method allows specifications formulated directly at the level of the complex physical behavior representation of an assembly, in the form of a causal graph of events (C1). Second, the probability of deviation from physical behavior specifications is minimized by increasing the robustness of the chain reaction to uncertainties (C3).

In the next chapter, I conclude this thesis by summarizing the findings of the case studies and discussing avenues for future research.

Chapter 5

Conclusion

In this thesis, I have argued that efficient methods could support designers seeking complex behaviors by improving their level of control over these behaviors. I have proposed a user-centric conceptual framework involving the notions of *behavior*, *control* and *complexity* of a design, and have presented two case studies in which complex assemblies are designed with an artistic purpose favoring a complex outcome over a simple one. The first study focused on efficiently improving the *constructive* control of the user over the complex behavior of a drawing machine, while the second study investigated the problem of increasing the *physical* control of the user over the complex behavior of a chain reaction contraption. The key contributions of these studies are summarized next, before discussing avenues for future research.

5.1 Summary

Recalling from Chapter 1, the work presented in this thesis makes the following contributions:

(C1) Representation: support new types of complex physical behavior specifications.

(C2) Exploration: introduce a new method enabling users to easily explore design variations that respect their physical behavior specifications.

(C3) Optimization: propose a new method to minimize the chance of real-life deviation from physical behavior specifications.

Chapter 3 focused on drawing machines: relatively simple mechanical assemblies producing intricate visual patterns that are hard to control. The goal of this work was to enable users to easily explore and fine-tune machine-made drawings constrained by their visual preferences. Our drawing-centric interface lets the user search for patterns by only sketching parts, which are automatically symmetrized. Then, it provides an interactive representation where feature points (e.g., intersection points) can be selected, and geometric constraints can be applied to them (e.g., staying at the same position) (C1). Our novel dynamic reparameterization method, combined with a slider-based interface, enables the user to explore and fine-tune realizations that respect these constraints (C2). We evaluated this method in two main ways. On the qualitative side, we demonstrated a wide range of constrained transformations, both in figures (for simple examples) and in videos (for complex ones), allowing users to explore diverse variations of a drawing. Additionally, we fabricated three machines to show that these variations stayed physically feasible. On the quantitative side, we ran a small scale user study which, after analysis, supported the claim that users can modify drawings more efficiently with a given visual constraint than without, even though the role of each slider is not explicitly known.

Chapter 4 considered the task of assembling chain reaction contraptions in a way that is robust to uncertainties. The key events expected during a run of this machine are described with the help of a causal graph (C1), while the exact motion of each component is left unspecified. We proposed to use the degrees of freedom left by such a partial specification to find an assembly layout robust to uncertainties (C3). We showed that in the case of a binary success/failure outcome, this amounts to increasing the probability of success parametrized by the layout variables. Assuming that the most probably successful layout given all sources of uncertainty was similar to the most probably successful layout given a simulated sample of the design space, we converted the original probability estimation problem into a binary classification task requiring no estimation of the real uncertainties at play. Our core technical contributions consisted in efficient sampling and learning strategies to build the parametric success probability estimator, notably taking advantage of the causal

graph structure to scale to high-dimensional design spaces. We evaluated this approach in two ways. Qualitatively, we fabricated four chain reaction contraptions optimized with our learned success probability, which succeeded either immediately or after four or five trials for the most complex ones (compared to the dozens of takes usually reported in examples online). Quantitatively, we implemented a measure of robustness based on a dense sampling of the design space, and compared the layout given by our method against the result given by three baseline methods. We showed that our approach consistently outperformed the others, and demonstrated the efficiency of our strategy to scale to high numbers of layout variables.

Despite their different applications and goals, both problems present similarities within the context of this thesis. First, the physical behavior of each assembly can change in complex and unpredictable ways as design parameters are changed even by a small amount (in which sense they are "near chaotic"). Second, in both cases the purpose of these machines is primarily visual, which requires an intuitive graphical representation at the center of the design specifications. Lastly, the intentionally complex behavior of these machines calls for an approach based on partial specifications and an efficient exploration of the design space, rather than an exhaustive specification of the behavior of interest.

5.2 Future work

Developing efficient algorithms to ease the design and fabrication of assemblies displaying an intentionally complex behavior is a promising research direction, with many problems left to explore. To conclude this thesis, I will first discuss how the methods summarized above can be extended to a wider range of applications, before proposing three themes for future research.

The case studies presented in this thesis focused on machines with artistic applications. It is important to consider how the core contributions of these works could be applied to other concrete problems. For instance, as mentioned in Section 3.2, the complex trajectory patterns produced by drawing machines are also commonly used with electromagnetic beams for 3D scanning [77]. In this context, the constrained

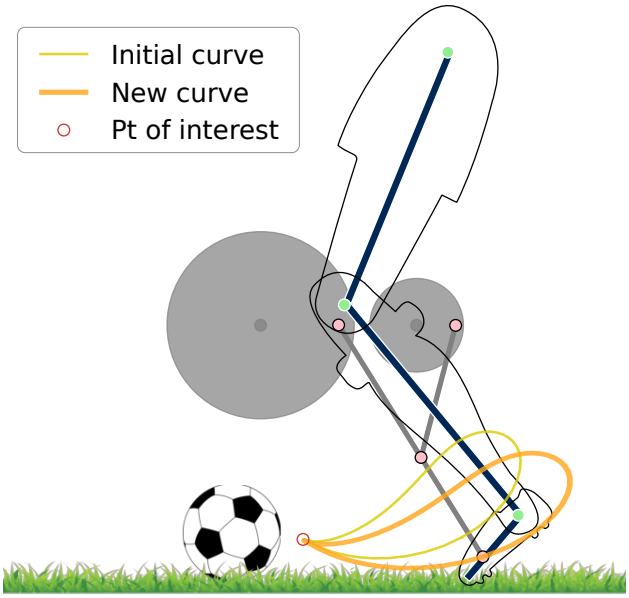


Figure 5.1: Extension to a mechanical character. A leg kicks a soccer ball.

exploration method could help explore different patterns respecting user-specified constraints. Moreover, this method is not only useful for intricate curves: it remains valid for general mechanisms with simpler continuous cyclic trajectories, as illustrated in Figure 5.1. I selected one of the elementary mechanisms from Coros et al.'s paper [26] and added it to our system. Connecting a "leg" to the end-effector (animated with inverse kinematics), one can imagine a situation where a user wishes to have the foot kicking a ball at the maximum of curvature (therefore deciding to fix its position) while exploring the various trajectories taken by the foot to reach this position. Beyond toys, designing cyclic motions is also relevant in industrial settings such as assembly lines, where the available constraints on a point of interest could be extended to speed and force, with no change needed in the rest of the pipeline except the choice of a physics engine allowing to compute these values.

Likewise, our method to improve the robustness of chain reaction contraptions could be applied to more practical mechanisms involving sequences of events, such as automatic dispensers[1], pop-up tents and ejection seats, to name a few. Indeed,

[1]Surprisingly, the first automatic vending machine dates as far back as Greek antiquity. Hero of Alexandria, in his book *Mechanics and Optics*, described a device able to automatically dispense holy water after a coin was inserted [125]. The coin would fall onto a platform that tipped, opening a valve letting the liquid flow. Then, the coin would fall off the tipped platform, closing the valve again—thus completing the chain reaction.

while we chose to limit our design space to layout parameters (as components were assumed to be preexisting), nothing prevents the inclusion of other geometric and physical parameters (although additional physical validity checks may have to be performed). The causal graph, however, would need to support additional features such as disjunctions (i.e., "if/else" paths), "or" preconditions (i.e., checking for an event condition if *any* of its parent events happened) and duration-based events (i.e., with a condition expected to be true for a given amount of time). This does not fundamentally change the method in Section 4.5; as for the factorization in Section 4.6, it only requires the graph to be directed and acyclic.

Beyond these extended applications, I propose three axes for future research.

5.2.1 Behavior specification and exploration

Chapter 3 addresses a problem for which, arguably, open-ended exploration might be more suitable than direct queries because it is difficult for the user to imagine which patterns are possible in the first place. These approaches, however, complete rather than oppose each other. An interesting problem, in terms of interaction, would be to investigate how these two modes of design might be alternated or combined in a way that is most intuitive and efficient for users. Nevertheless, while many works in computational design have explored problems in which the main interaction is a direct user query (see Section 2.4), constrained exploration has received far less attention, especially when designing physical behaviors. Some direct extensions to our method would be to allow visual preferences in the form of soft (rather than hard) constraints, and to allow the formulation of constraints on more complex elements of the behavior representation: e.g., trajectory arcs and groups of arcs. As mentioned above, beyond geometry, the potential applications would greatly expand with the addition of physical constraints, such as speed and force.

Moreover, while our work relies on sliders to explore local behavior variations under constraints in a continuous design space, many methods have been proposed to organize and explore collections of shapes [152]. From an interaction standpoint, one weakness of the slider-based approach is that the effect of each slider might be difficult to understand or remember, especially above three sliders. This problem

could be alleviated in various ways: for instance, previsualizing the effects with arrows (similar to the work by Mitra et al. on illustrating how mechanical assemblies work [97]), or using additional dimensionality reduction techniques (e.g., multidimensional scaling) to navigate possible variations in a 2D map. Such data-driven methods, however, tend to be applied in advance to vast preexisting collections, rather than to a smaller set of behavior variations computed on the fly. Interactive behavior exploration poses two additional challenges worth investigating: (i) how to manage small design parameter changes leading to large behavior changes; and (ii) how to ensure that the method is fast enough to stay interactive.

5.2.2 Computational models of physical causality

A key goal in computational design is to develop models able to automatically identify and exploit functional relationships between the different parts of an assembly. Physical causality is central to such an understanding of functionality, as it links spatial (e.g., the dimensions and layout of parts) and temporal variables (e.g., the occurrence and duration of an event) logically or probabilistically (depending on how causality is represented). Models of physical causality have many important applications. For instance, while in Chapter 4 we assume that the design space contains a continuous region of successful instances, such models could be used to predict whether a success region exists, i.e., whether a user-defined behavior is achievable at all. Feedback could then be provided to indicate failure points. A better understanding of causal relationships could also help our system use simulations more efficiently, by stopping runs early if failure is expected. In the same vein, when building the CPSP factors, further analysis could allow restarting simulations mid-run, i.e., from an event-specific initial state.

While our work takes advantage of situations where the physical behavior is under-specified, an even more ambitous goal would be to automatically complete assemblies when parts of the behavior are not specified at all. To give an example inspired by our own work, a user could want to generate a variety of chain reaction contraptions by providing only the beginning and end of the chain. To reduce the large number of valid solutions, constraints could also be added manually (e.g., in

the form of a partial causal graph, or motion sketches) or automatically (by inferring the functionality of the artifact). While existing methods in computational design for mechanical assemblies already allow the completion of kinematic chains with a library of components [26, 130], generalizing these solutions to assemblies of disconnected bodies is particularly challenging, as the possible component combinations are much higher and the resulting behaviors more diverse. This in turn requires a significant amount of prediction and decision making on the part of the algorithm. The applications, on the other hand, are also quite promising, from helping novice users repair broken artifacts [73] or augment existing ones [84], to supporting engineers and designers in creating new inventions.

Lastly, the ability to model physical causality may help differentiate impossible behaviors by quantifying *how impossible* they are (i.e., quantifying the changes required to make these behaviors possible). Such a metric could be useful to explore regions of the design space in which the simulated behavior of an artifact is undefined. For instance, some design parameters may be discrete, or the violation of physical validity constraints may result in an unstable simulation that does not reflect real-world physics. Transitioning through such undefined behaviors, however, could be valuable both for exploration (e.g., when using the 2D map metaphor mentioned in Section 5.2.1) and optimization (i.e., relaxing the problem to reach the optimum faster). A promising line of research is to combine such physical causality models with machine learning models able to hallucinate an approximation of these intermediate behaviors.

5.2.3 Uncertainty and robustness

While the method presented in Chapter 4 efficiently increases the probability of simulated success of a chain reaction contraption, the connection with the physical robustness of the design relies on assumptions that are worth investigating. The first hypothesis to test is whether increasing the robustness of the layout to the uncertainty of a single controllable variable reliably increases its robustness to the uncertainty of uncontrollable variables. The second hypothesis is the similarity between layouts maximizing the probabilities of simulated and physical success.

Going further, how correlated are these parametric probabilities across the design space? An answer to this question would be particularly useful to give the user an estimate of the probability of physical success of the final layout, and by extension, an estimate of the number of attempts to expect before the contraption works. Moreover, quantifying the sensitivity of the physical probability to the variables involved would help to guide and advise the user regarding which object measurements or placements require more care than others. Additionally, as mentioned in Section 2.5, our method could be extended to include a calibration step increasing the accuracy of the simulator. At the other end of the pipeline, alternative ways of communicating the final layout to the user could be experimented. For instance, Peng et al. [104] very recently proposed a system using interactive projection mapping to guide users building domino runs.

Given the research directions mentioned above, it is clear that computational design has many exciting problems left to explore. This thesis, by proposing efficient methods to help users create assemblies displaying an intentionally complex physical behavior, makes a small step towards more intelligent and intuitive design tools.

Appendix A

Kinematics of drawing machines

A.1 Introduction

This appendix describes the parameterization and kinematic behavior of the drawing machines supported by our system. For each machine, we present the following elements.

- **Design parameters.** These parameters describe the layout and physical dimensions of the components of a machine. They can be continuous or discrete.
- **Constraints on design parameters.** These constraints translate high-level feasibility requirements and symmetry reductions (see typology below) into a set of (in)equalities.
- **Kinematic equations.** The pattern drawn by a machine is expressed as a function of time and design parameters.

A.1.1 Time interval

Although the kinematic equations were obtained using a geometric approach, their time-periodicity can be determined by applying modular arithmetic on a subset of the design parameters. Let us first consider a pair of meshing gears, G_1 and G_2, which both influence the drawing pen. We note their (equivalent) radii r_1 and r_2, which we assume to be integers. The law of gearing gives:

$$\frac{r_1}{r_2} = \frac{\omega_2}{\omega_1} = \frac{\mathcal{T}_1}{\mathcal{T}_2}, \tag{A.1}$$

where ω_i and $\mathcal{T}_i$ are respectively the angular speed and period of gear G_i, related by $\omega_i = 2\pi/\mathcal{T}_i$.

The system has performed a full cycle when both gears have simultaneously come back to their initial configuration, after a period:

$$\mathcal{T}_{12} = n_1\mathcal{T}_1 = n_2\mathcal{T}_2, \tag{A.2}$$

where n_1 and n_2 are the integer numbers of rotations experienced by each gear. By definition, the minimal value that satisfies this relation is the Least Common Multiple. Therefore:

$$\mathcal{T}_{12} = \mathrm{lcm}(\mathcal{T}_1, \mathcal{T}_2). \tag{A.3}$$

If we add a new gear G_3, which meshes with one of the existing gears and also influences the drawing pen, all we need to do is compute the new period:

$$\mathcal{T}_{123} = \mathrm{lcm}(\mathcal{T}_{12}, \mathcal{T}_3), \tag{A.4}$$

and so on for any additional gear.

A.1.2 Typology of constraints

For each machine, constraints on shape parameters reflect a number of functional and practical features.

Finite pattern. The first requirement is that the drawing can be produced in a reasonable amount of time, and *a fortiori*, a finite one. For each pair of gears, combining Eq. A.1 and A.2 gives the necessary condition

$$\frac{r_1}{r_2} = \frac{\mathcal{T}_1}{\mathcal{T}_2} = \frac{n_1}{n_2} \in \mathbb{Q}. \tag{A.5}$$

In other words, if $r_1/r_2 \notin \mathbb{Q}$, the gears will take an infinite amount of turns to come back to their initial configuration.

We incorporate this rationality constraint in our parameterizations by taking gear radii in $\mathbb{N}_{>0}$. This constraint can be extended to non-circular gears, as demon-

Table A.1: Main notations used in this appendix.

Notation	Meaning
r (resp. $\overline{r}$)	Free (resp. fixed) parameter
A	Geometric point
$\overrightarrow{\mathrm{AB}}$ (resp. $\vec{v}$)	Bound (resp. free) vector
$\|.\|$	Euclidean norm
$\mathfrak{R}$	Reference frame
R_θ	Rotation of angle θ centered at the origin

strated in Section A.2.2.

Compatible curvatures. Although, in classic mechanisms, non-circular gears often come in pairs with conjugate profiles, interesting trajectories can be obtained when rolling such a gear along a shape of different type. However, all not shapes can be rolled along each other without interference. Let us consider the case of an ellipse $\mathcal{E}_1$ rolling inside another ellipse $\mathcal{E}_2$. We can formulate the curvature compatibility constraint as

$$\kappa_{min}^{\mathcal{E}_1} \geq \kappa_{max}^{\mathcal{E}_2}.$$

This constraint ensures that at the point where the moving ellipse is the 'straightest', or least curved, it can still fit wherever the fixed enclosing ellipse is the most curved. While necessary, this constraint is not sufficient in more general cases, where *ad hoc* conditions should be added to avoid collisions.

Symmetry and congruence reduction. For a single machine, the space of available patterns can be very diverse but also highly redundant. This is typically caused by symmetries in the mechanical layout, or parameter combinations giving patterns that only differ by an affine transformation. Whenever this is possible, we put constraints on the parameters to remove such redundancies.

Valid layout. This type of condition only considers the abstract geometric model. It prevents components belonging to the same layer from overlapping. These are only basic constraints though, as in practice, collision avoidance depends on the specific physical implementation of the machine.

Drawing bounds. The size of the canvas is bounded, and the pen should never leave this drawing area.

Singularity avoidance. This problem has already been described in previous works [3]. Singular configurations typically happen when two links become perfectly aligned, which should be avoided (be it in the simulation or in real life).

Practical dimensions. Lastly, although some parameters could grow indefinitely without being limited by one of the previous constraints, we set some arbitrary bounds to ensure that the machine keeps a reasonable size.

The constraints are not always easy to evaluate. For instance, in the general case, the "drawing bounds" condition would require to produce the pattern in order to estimate whether or not the bounds are satisfied. This poses a chicken-and-egg problem, since we want to compute a pattern only if the constraints are satisfied. As a consequence, every time this situation arose, we determined a sufficient condition that only depends on the shape parameters.

A.1.3 Notations

See Table A.1 for the main notations. To improve the readability of equations, we sometimes make the following simplifications:

- all vectors are expressed in a fixed reference frame $\mathfrak{R}_0$ unless stated otherwise,
- all variable vectors and scalars of kinematic equations are time functions taken defined over $[0, \mathcal{T}]$, where $\mathcal{T}$ is the period defined in Section A.1.1.

A.2 Drawing machines

A.2.1 Basic Spirograph

The Spirograph is a 1965 drawing toy developed by Denys Fisher. Nowadays, the Spirograph is a complete set composed of various toothed plastic shapes. In our work, however, we focused on the "classic" version: a gear rolling without slipping inside or outside a circular ring.

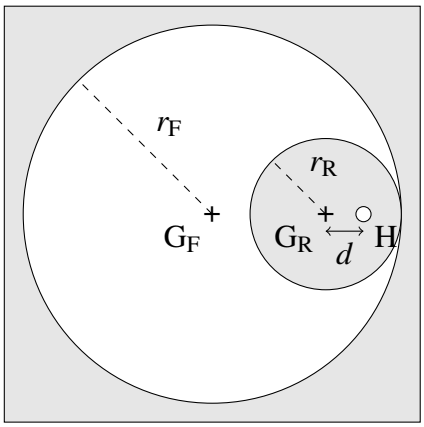


Figure A.1: Dimensions of a basic Spirograph. (In the case where the moving gear rolls inside the fixed one.)

Table A.2: Symbols for the Spirograph model.

Component	Geometric center
Fixed gear	G_F
Rolling gear	G_R
Pen hole	H

Parameter	Meaning
r_F	Radius of the fixed gear's primitive
r_R	Radius of the rolling gear's primitive
d	Distance between G_R and H

Design parameters. Three parameters completely describe the figures drawn by a classic Spirograph (factoring out rigid transformations). See Table A.2 for symbols and Figure A.1 for a geometric representation.

Constraints. The following constraints are grouped under the categories outlined in Section A.1.2.

- Finite pattern:

$$r_F,\ r_R \in \mathbb{N}_{>0}, \tag{A.6}$$

$$r_F < \mathcal{B}_r, \tag{A.7}$$

where $\mathcal{B}_r$ is an arbitrary integer constant.

- Valid layout:

$$0 < r_{\mathrm{R}} < r_{\mathrm{F}}, \tag{A.8}$$

$$|d| \leqslant r_{\mathrm{R}}. \tag{A.9}$$

- Symmetry and congruence reduction:

$$\gcd(r_{\mathrm{F}}, r_{\mathrm{R}}) = 1, \tag{A.10}$$

$$d > 0. \tag{A.11}$$

Kinematic equations. The basic Spirograph produces curves called *trochoids* (*hypotrochoids* if the gear rolls inside the ring, and *epitrochoids* if it rolls outside the ring). The parametric equation of a trochoid in Cartesian coordinates is:

$$\vec{\gamma}(t) \coloneqq \overline{r_{\mathrm{R}}}(\overline{q} \mp 1) \begin{bmatrix} \cos(t) \\ \sin(t) \end{bmatrix} + \overline{d} \begin{bmatrix} \pm\cos\left((\overline{q} \mp 1)\, t\right) \\ -\sin\left((\overline{q} \mp 1)\, t\right) \end{bmatrix} \quad t \in [0, \mathcal{T}], \tag{A.12}$$

where $\overline{q} = \overline{r_{\mathrm{F}}}/\overline{r_{\mathrm{R}}}$, and the upper and lower operators respectively denote the hypo- and epitrochoid cases.

A.2.2 Elliptic Spirograph

This slight variation of the base Spirograph does not correspond to a particular commercial toy. We developed it as a simple extension that adds one continuous parameter to the previous model. For this reason, we voluntarily constrained the pen hole to lie on the major axis of the elliptic primitive. However, the geometric analysis gets already more involved than in the basic case.

Design parameters. The elliptic Spirograph is described by four parameters defining the layout and dimensions of two components. See Table A.3 for symbols and Figure A.2 for a geometric representation.

Ellipses are usually described by their semi-minor and semi-major axes a and b. However, the finite-curve constraint is more complex to write in terms of these

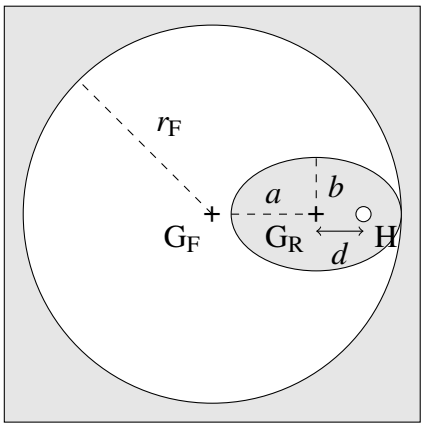


Figure A.2: Dimension of the elliptic Spirograph model.

parameters. For non-circular gears, Equation A.5 can be generalized to

$$\frac{P_1}{P_2} = \frac{\mathcal{T}_1}{\mathcal{T}_2} = \frac{n_1}{n_2} \in \mathbb{Q} \tag{A.13}$$

where P_i is the perimeter of gear i. For an elliptic profile, it is given by

$$P = 4aE(e), \tag{A.14}$$

where E is the well-known complete elliptic integral of the second kind:

$$E(k) = \int_0^{\frac{\pi}{2}} \sqrt{1 - k^2 \sin^2 \theta} d\theta, \tag{A.15}$$

and $e = \sqrt{1 - \frac{b^2}{a^2}}$ is the eccentricity of the ellipse.

Thus in practice, we consider the more convenient pair (r_R^{eq}, e), where r_R^{eq}, the 'equivalent radius', is the radius of a circle with the same perimeter. Therefore r_R^{eq} can follow the same condition as the other radii. The link between the two pairs of parameters is given by:

$$2\pi r_R^{eq} = 4aE(e), \tag{A.16}$$

$$b = a\sqrt{1 - e^2}. \tag{A.17}$$

Constraints. The following constraints are grouped by the categories outlined in Section A.1.2.

- Finite pattern:

$$r_{\mathrm{F}},\ r_{\mathrm{R}}^{eq} \in \mathbb{N}_{>0}, \tag{A.18}$$

$$r_{\mathrm{F}} < \mathcal{B}_r, \tag{A.19}$$

  where $\mathcal{B}_r$ is an arbitrary integer constant.

- Compatible curvatures:

$$a < \sqrt{br_{\mathrm{F}}}. \tag{A.20}$$

- Valid layout:

$$0 < r_{\mathrm{R}}^{eq} < r_{\mathrm{F}}, \tag{A.21}$$

$$0 \leqslant e < 1, \tag{A.22}$$

$$|d| \leqslant a. \tag{A.23}$$

- Symmetry and congruence reduction:

$$\gcd(r_{\mathrm{F}}, r_{\mathrm{R}}^{eq}) = 1, \tag{A.24}$$

$$d > 0. \tag{A.25}$$

Kinematic equations. As a generalization of trochoids, when a point M is attached to a moving curve Γ_m that rolls without slipping (RWS) along a fixed curve Γ_f, both being in the same plane $\mathcal{P}$, M describes a curve Γ_r in $\mathcal{P}$, called a *roulette curve*. In *The General Theory of Roulettes* [148], Walker gives a general method to obtain the parametric equation of roulette curves. We follow the ideas of his demonstration but adapt the equations to our own formulation.

Table A.3: Symbols for the elliptic Spirograph model. Parameter pairs marked by * and † can equivalently describe the elliptic primitive.

Component	Geometric center
Fixed gear	G_F
Rolling gear	G_R
Pen hole	H

Parameter	Meaning
r_F	Radius of the fixed gear's primitive
a	Semi-major axis of the rolling gear's primitive*
b	Semi-minor axis of the rolling gear's primitive*
r_R^{eq}	Radius of the circle with the same perimeter† as the rolling gear's primitive
e	Eccentricity of the rolling gear's primitive†
d	Distance between G_R and H

We are looking for an expression of the roulette curve:

$$\vec{\gamma}(t) \coloneqq \overrightarrow{G_F H}(t) \quad t \in [0, \mathcal{T}]. \tag{A.26}$$

We first introduce two reference frames:

- $\mathfrak{R}_0$, a frame attached the fixed gear with G_F as its origin,
- and $\mathfrak{R}_R$, a moving frame attached to the rolling gear with G_R as its origin.

We consider the primitive curves of each gear Γ_F and Γ_R, initially touching each other at a contact point C_0 with their tangents aligned. Each curve is described in its own reference frame by a differentiable parametric function, namely $\vec{\gamma_F}$ and $\vec{\gamma_R}$. At every t, the contact point C travels on both curves, so that:

$$\overrightarrow{G_F C}(t) = \vec{\gamma_F}(t) = [\vec{\gamma_R}(t)]_{\mathfrak{R}_0}, \tag{A.27}$$

where $[.]_{\mathfrak{R}_0}$ denotes the reference frame. More specifically, for any point M attached to $\mathfrak{R}_R$:

$$\left[\overrightarrow{G_F M}\right]_{\mathfrak{R}_0} = \overrightarrow{G_F G_R}(t) + R_{\phi_{FR}(t)}\left(\left[\overrightarrow{G_R M}\right]_{\mathfrak{R}_R} - \overrightarrow{G_F G_R}(t)\right), \tag{A.28}$$

where $\phi_{FR}(t)$ is the oriented angle between $\mathfrak{R}_0$ and $\mathfrak{R}_R$ at t. Since the tangents to each curve at C_0 must always be aligned, the rotation should compensate for the angular difference between the tangents in their own reference frame. Therefore:

$$\phi_{FR}(t) = -\left(\phi_F(t) - \phi_R(t)\right) \quad \forall t, \tag{A.29}$$

where ϕ_F and ϕ_R are the tangential angles of $\overrightarrow{\gamma_F}$ and $\overrightarrow{\gamma_R}$ respectively.

Applying Equation A.28 to $\overrightarrow{\gamma_R}$ and using the coincidence relation Equation A.27 gives:

$$\overrightarrow{\gamma_F}(t) = \overrightarrow{G_F G_R}(t) + R_{\phi_{FR}(t)}\left(\overrightarrow{\gamma_R}(t) - \overrightarrow{G_F G_R}(t)\right) \quad \forall t. \tag{A.30}$$

Then, we can evaluate Equation A.28 at point H, substract the above equation to get rid of $\overrightarrow{G_F G_R}$, and finally use the curve definition Equation A.26 to write:

$$\overrightarrow{\gamma}(t) \coloneqq \overrightarrow{\gamma_F}(t) + R_{\phi_{FR}(t)}\left(\left[\overrightarrow{G_R H}\right]_{\mathfrak{R}_R} - \overrightarrow{\gamma_R}(t)\right) \quad \forall t. \tag{A.31}$$

Equation A.31 allows us to compute the roulette once the specific gear profile parameterizations γ_F and γ_R are known. It does not, however, tell us how the RWS condition is enforced. Following this constraint, the contact point must travel on both curves at the same speed, meaning that:

$$\left\|\overrightarrow{\dot{\gamma}_F}(t)\right\| = \left\|\overrightarrow{\dot{\gamma}_R}(t)\right\| \quad \forall t. \tag{A.32}$$

Integrating this equation translates into an equality of arc lengths for all t. Therefore we introduce a common arc length function:

$$\Lambda : \begin{cases} [0, \mathcal{T}] \to [0, \mathcal{L}] \\ t \mapsto \Lambda(t) = \Lambda_F(t) = \Lambda_R(t) \end{cases} \tag{A.33}$$

where $\mathcal{L}$ is the total distance traveled during the drawing. Using the fundamental theorem of calculus, it is easy to show that the arc length function is continuous and strictly monotone, and therefore bijective. We use the reciprocal function to

implement the RWS condition via a change of variables:

$$\overrightarrow{\gamma_*}'(s) \coloneqq \overrightarrow{\gamma_*} \circ \Lambda^{-1}(s) \quad \forall s \in [0, \mathcal{L}], \tag{A.34}$$

where s is called the *arc length parameter*, and is the same for both curves.

Finally, the roulette curve in Equation A.31 becomes:

$$\overrightarrow{\gamma}'(s) \coloneqq \overrightarrow{\gamma_{\mathrm{F}}}'(s) + R_{\phi_{\mathrm{FR}}(s)} \left(\left[\overrightarrow{\mathrm{GH}} \right]_{\Re_{\mathrm{R}}} - \overrightarrow{\gamma_{\mathrm{R}}}'(s) \right) \quad s \in [0, \mathcal{L}]. \tag{A.35}$$

The above developments are purely theoretical, and could be applied to gears of various primitive shapes, as long as they have compatible curvatures and do not collide during the drawing. Let us now demonstrate the case of the elliptic Spirograph. The basic parameterizations are:

$$\overrightarrow{\gamma_{\mathrm{F}}}(t) \coloneqq \overline{r_{\mathrm{F}}} \begin{bmatrix} \cos(t) \\ \sin(t) \end{bmatrix} \quad t \in [0, \mathcal{T}], \tag{A.36}$$

and

$$\overrightarrow{\gamma_{\mathrm{R}}}(t) \coloneqq \begin{bmatrix} \overline{a} \cos(t) \\ \overline{b} \sin(t) \end{bmatrix} \quad t \in [0, \mathcal{T}]. \tag{A.37}$$

Transforming Equations A.36 and A.37 into arc length parameterizations necessitates to invert their respective arc length functions. While being easy for $\overrightarrow{\gamma_{\mathrm{F}}}(t)$:

$$\begin{aligned} \Lambda_{\mathrm{F}}(t) &\coloneqq \overline{r_{\mathrm{F}}}\, t & \forall t \in [0, \mathcal{T}], \\ \Lambda_{\mathrm{F}}^{-1}(s) &\coloneqq s / \overline{r_{\mathrm{F}}} & \forall s \in [0, \mathcal{L}], \end{aligned}$$

it is more complicated for $\overrightarrow{\gamma_{\mathrm{R}}}(t)$:

$$\Lambda_{\mathrm{R}}(t) \coloneqq \overline{a} \left(E\left(t + \frac{\pi}{2}, \overline{e}\right) - E(\overline{e}) \right) \quad \forall t \in [0, \mathcal{T}], \tag{A.38}$$

where this time $E(\phi, k)$ denotes the *incomplete* elliptic integral of the second kind, function of an amplitude ϕ and an elliptic modulus k. Inverting E with respect

to amplitude is theoretically possible; however, as noted in an article by Ghrist et al. [47], while inverses of elliptic integrals of the first kind are well-known in the literature (and called *Jacobi elliptic functions*), very little has been written on their second kind counterparts.

In practice we could compute the inverse of E numerically; however, in this case we opt for a more efficient method, which can be applied to other complicated scenarios. The idea is to keep the original parameterization of the more complex shape, and only invert the arc length function of the simpler one. Here for instance, we keep the ellipse parameterization $\overrightarrow{\gamma_{\mathrm{R}}}$, and use a new circle parameterization

$$\overrightarrow{\gamma_{\mathrm{F}}}' \coloneqq \overrightarrow{\gamma_{\mathrm{F}}} \circ \Lambda_{\mathrm{F}}^{-1} \circ \Lambda_{\mathrm{R}}, \tag{A.39}$$

which only requires to evaluate E directly. This "trick" allows us to enforce the RWS condition efficiently.

A.2.3 Cycloid Drawing Machine

The Cycloid Drawing Machine is a relatively recent project by Joe Freedman [40], with a virtual version by Jim Bumgardner [16] available online. Despite its name, this machine can produce figures that are far more complex than cycloids, which can be obtained from a simple Spirograph. It allows several configurations, among which we only implemented the simplest one (single gear, fixed fulcrum).

Design parameters. Our configuration of the Cycloid Drawing Machine is described by six parameters, controlling the layout and dimensions of six components. See Table A.4 for symbols and Figure A.3 for a geometric representation.

Constraints. The following constraints are grouped by the categories outlined in Section A.1.2.

- Finite pattern:

$$r_{\mathrm{T}},\ r_{\mathrm{E}} \in \mathbb{N}_{>0}, \tag{A.40}$$

$$r_{\mathrm{T}},\ r_{\mathrm{E}} < \mathcal{B}_r, \tag{A.41}$$

where $\mathcal{B}_r$ is an arbitrary integer constant.

- Valid layout:

$$0 < r_{\mathrm{T}} < d_{\mathrm{TF}}, \tag{A.42}$$

$$0 < d_{\mathrm{FH}} < d_{\mathrm{FE}} - r_{\mathrm{E}}, \tag{A.43}$$

$$0 \leqslant r_{\mathrm{S}} < r_{\mathrm{E}}, \tag{A.44}$$

with:

$$\begin{aligned} d_{\mathrm{FE}} &\coloneqq \left\| \overrightarrow{\mathrm{FG_E}} \right\| \\ &= \left(d_{\mathrm{FH}}^2 + (r_{\mathrm{T}} + r_{\mathrm{E}})^2 - 2 d_{\mathrm{FH}} (r_{\mathrm{T}} + r_{\mathrm{E}}) \cos \theta_{\mathrm{TE}} \right)^{\frac{1}{2}} \end{aligned}$$

- Symmetry and congruence reduction:

$$\gcd(r_{\mathrm{T}}, r_{\mathrm{E}}) = 1, \tag{A.45}$$

$$0 \leqslant \theta_{\mathrm{TE}} \leqslant \pi. \tag{A.46}$$

- Drawing bounds:

$$\max_t \left\| \overrightarrow{\mathrm{G_T H}} \right\| < r_{\mathrm{T}}, \tag{A.47}$$

with

$$\arg\max \left\| \overrightarrow{\mathrm{G_T H}} \right\| = \begin{bmatrix} d_{\mathrm{TF}} \\ 0 \end{bmatrix} + d_{\mathrm{FH}} \begin{bmatrix} \cos(\theta_{\mathrm{FE}} - \alpha) \\ \sin(\theta_{\mathrm{FE}} - \alpha) \end{bmatrix}, \tag{A.48}$$

where α is the amplitude of the oscillation:

$$\alpha \coloneqq \arctan\left(\frac{r_{\mathrm{S}}}{d_{\mathrm{FE}}} \right), \tag{A.49}$$

and θ_{FE} is the polar angle of $\overrightarrow{\mathrm{FG_E}}$:

$$\theta_{\mathrm{FE}} \coloneqq \pi - \arctan\left(\frac{(r_{\mathrm{T}} + r_{\mathrm{E}}) \sin \theta_{\mathrm{TE}}}{d_{\mathrm{TF}} - (r_{\mathrm{T}} + r_{\mathrm{E}}) \cos \theta_{\mathrm{TE}}} \right) \tag{A.50}$$

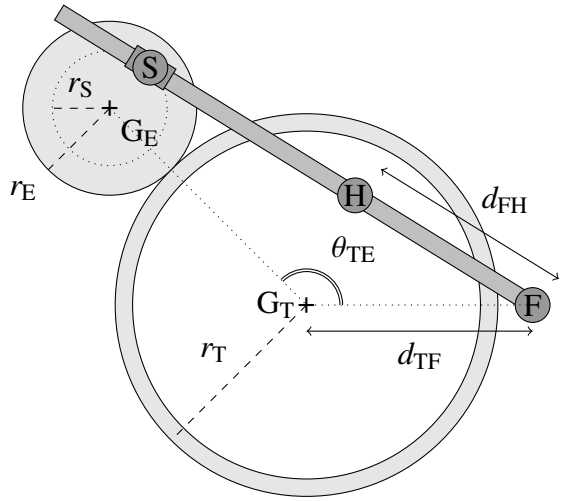


Figure A.3: Dimensions of the Cycloid Drawing Machine model.

Table A.4: Symbols for the Cycloid Drawing Machine model.

Component	**Geometric center**
Turntable gear	G_T
External gear	G_E
Fulcrum (or pivot)	F
Slider	S
Pen-holder	H

Parameter	**Meaning**
r_T	Radius of the turntable gear's primitive
r_E	Radius of the external gear's primitive
d_{TF}	Distance between G_T and F
θ_{TE}	Polar angle of $\overrightarrow{G_T G_E}$
r_S	Distance between G_E and S
d_{FH}	Distance between F and H

- Practical dimensions:

$$d_{TF} \leqslant \mathcal{B}_d, \tag{A.51}$$

  where $\mathcal{B}_d$ is an arbitrary constant.

Kinematic equations. The Cycloid Drawing Machine and the Hoot-Nanny both use a turntable to increase the complexity of the drawing they produce. The equation of the corresponding curve can be expressed in a general way.

We first consider two frames, sharing the same origin $\mathrm{G_T}$:

- a fixed frame of reference $\mathfrak{R}_0$,
- and a rotating frame of reference $\mathfrak{R}_\mathrm{T}$, attached to the turntable.

The polar angle from $\mathfrak{R}_0$ to $\mathfrak{R}_\mathrm{T}$ is equal to θ_T, the rotation angle of the turntable gear. Since both frames have the same origin, the frame change equation is simpler than for the elliptic Spirograph (*cf.* Equation A.28. For any point M attached to the rotating frame, it comes:

$$[\overrightarrow{\mathrm{G_TM}}]_{\mathfrak{R}_\mathrm{T}} = R_{-\theta_\mathrm{T}}[\overrightarrow{\mathrm{G_TM}}]_{\mathfrak{R}_0}. \tag{A.52}$$

We can then define the drawing obtained from turntable-based machines as the curve:

$$\vec{\gamma}(t) \coloneqq [\overrightarrow{\mathrm{G_TH}}(t)]_{\mathfrak{R}_T} \quad t \in [0, \mathcal{T}]. \tag{A.53}$$

Let us now express $\vec{\gamma}(t)$ as a function of the shape parameters. We have

$$\begin{aligned}\overrightarrow{\mathrm{G_TH}} &= \overrightarrow{\mathrm{G_TF}} + \overrightarrow{\mathrm{FH}} \\ &= \begin{bmatrix} \overline{d_\mathrm{TF}} \\ 0 \end{bmatrix} + \overline{d_\mathrm{FH}} \frac{\overrightarrow{\mathrm{FS}}}{\left\|\overrightarrow{\mathrm{FS}}\right\|},\end{aligned}$$

with:

$$\begin{aligned}\overrightarrow{\mathrm{FS}} &= \overrightarrow{\mathrm{FG_T}} + \overrightarrow{\mathrm{G_TG_E}} + \overrightarrow{\mathrm{G_ES}} \\ &= \begin{bmatrix} -\overline{d_\mathrm{TF}} \\ 0 \end{bmatrix} + (\overline{r_\mathrm{T}} + \overline{r_\mathrm{E}}) \begin{bmatrix} \cos \overline{\theta_\mathrm{TE}} \\ \sin \overline{\theta_\mathrm{TE}} \end{bmatrix} + \overline{r_\mathrm{S}} \begin{bmatrix} \cos \theta_\mathrm{ES} \\ \sin \theta_\mathrm{ES} \end{bmatrix}\end{aligned}$$

where θ_ES is the polar angle of $\overrightarrow{\mathrm{G_ES}}$.

All that remains is to express the gear angles as functions of time. Integrating the law of gearing (Equation A.1) and taking the constant to be 0 yields:

$$\theta_\mathrm{ES}(t) = -\frac{\overline{r_\mathrm{T}}}{\overline{r_\mathrm{G}}} \theta_\mathrm{T}(t) \quad \forall t. \tag{A.54}$$

Table A.5: Symbols for the Hoot-Nanny model.

Component	Geometric center
Turntable gear	G_T
External gear i	G_i
Pivot joint i	P_i
Arm i	A_i
Pen-holder	H

Parameter	Meaning
r_T	Radius of the turntable gear's primitive
r_{G_i}	Radius of external gear i 's primitive
θ_{12}	Angle $\widehat{G_1 G_T G_2}$
r_{P_i}	Distance between P_i and G_i
l_i	Length of arm i

Finally, if the turntable gear is considered as the driving gear, θ_T can be simply taken as:

$$\theta_T(t) := t \quad \forall t \in [0, \mathcal{T}]. \tag{A.55}$$

While a dedicated driving gear can be added for convenience, it does not influence the aspect of the drawing and therefore is not considered in this model.

A.2.4 Hoot-Nanny

The Hoot-Nanny, also known as Magic Designer, is an older toy from the middle of the 20th century, produced and commercialized at the time by Northern Signal Company, Milwaukee. The "trick" of fixing the sheet of paper onto a turntable to increase the complexity of the drawings may have inspired the design of the Cycloid Drawing Machine described above. An online version by Abel Vincze [147] called "HTML Spirograph" allows to draw an impressive range of beautiful patterns by varying the color and transparency of the curve.

Design parameters. Our Hoot-Nanny is described by eight parameters, controlling the layout and dimensions of eight components. See Table A.5 for symbols and Figure A.4 for a geometric representation.

Constraints. The following constraints are grouped by the categories outlined in Section A.1.2.

- Finite pattern:

$$r_{\mathrm{T}},\ r_{\mathrm{G}_i} \in \mathbb{N}_{>0}, \tag{A.56}$$

$$r_{\mathrm{T}},\ r_{\mathrm{G}_i} < \mathcal{B}_r, \tag{A.57}$$

  where $\mathcal{B}_r$ is an arbitrary integer constant.

- Valid layout:

$$r_{\mathrm{P}_i} < r_{\mathrm{G}_i}, \tag{A.58}$$

$$r_{\mathrm{G}_1} + r_{\mathrm{G}_2} < \left\|\overrightarrow{\mathrm{G}_1\mathrm{G}_2}\right\|, \tag{A.59}$$

$$r_{\mathrm{G}_i} + r_{\mathrm{P}_i} < l_i < 2r_{\mathrm{T}} + r_{\mathrm{G}_i} - r_{\mathrm{P}_i}, \tag{A.60}$$

  with:

$$\left\|\overrightarrow{\mathrm{G}_1\mathrm{G}_2}\right\| = \left(d_{\mathrm{TG}_1}^2 + d_{\mathrm{TG}_2}^2 - 2d_{\mathrm{TG}_1}d_{\mathrm{TG}_2}\cos\theta_{12}\right)^{\frac{1}{2}} \tag{A.61}$$

  where $d_{\mathrm{TG}_i} \coloneqq r_{\mathrm{T}} + r_{\mathrm{G}_i}$.

- Symmetry and congruence reduction:

$$\gcd(r_{\mathrm{T}}, r_{\mathrm{G}_i}) = 1, \tag{A.62}$$

$$0 \leqslant \theta_{12} \leqslant \pi. \tag{A.63}$$

- Drawing bounds:

$$\max_t \left\|\overrightarrow{\mathrm{G}_\mathrm{T}\mathrm{H}}\right\| < r_{\mathrm{T}}, \tag{A.64}$$

  for which a sufficient condition can be found by observing that a feasible trajectory of the pen in the fixed reference frame (which is not the same as the drawing curve) is always inscribed in a quadrilateral whose vertices are reached when the triplets $(\mathrm{G}_1, \mathrm{P}_1, \mathrm{H})$ and $(\mathrm{G}_2 1, \mathrm{P}_2, \mathrm{H})$ become respectively

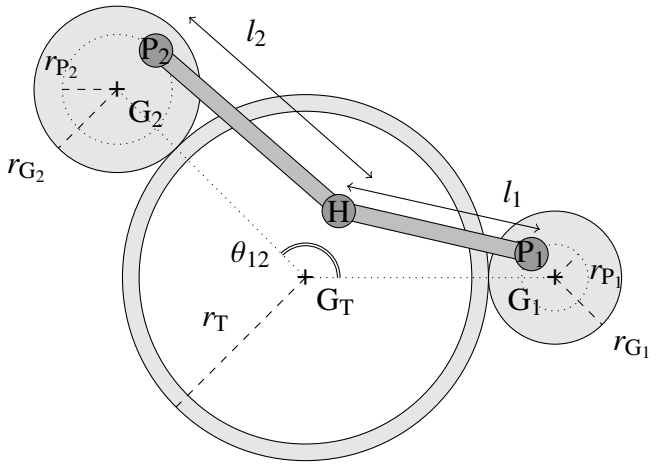


Figure A.4: Dimensions of the Hoot-Nanny model.

aligned. For these points the distances between the gear centers and the pen are extremal, with values $d^{ext}_{\mathrm{HG}_i} = r_{\mathrm{G}_i} \pm l_i$, which makes indeed four combinations. Therefore, all that we need to determine is the position of the four vertices, and find the distance of the one that is farthest from the turntable center. The position of a vertex is obtained by applying the law of cosines to the triangle $\mathrm{HG_2G_1}$:

$$\overrightarrow{\mathrm{G_TH}} = \overrightarrow{\mathrm{G_TG_2}} + d^{ext}_{\mathrm{HG}_2} R_\alpha \frac{\overrightarrow{\mathrm{G_2G_1}}}{\left\|\overrightarrow{\mathrm{G_1G_2}}\right\|}, \tag{A.65}$$

where α is the angle $\widehat{\mathrm{HG_2G_1}}$, given by:

$$\alpha = \arccos\left(\frac{-(d^{ext}_{\mathrm{HG}_1})^2 + \left\|\overrightarrow{\mathrm{G_1G_2}}\right\|^2 + (d^{ext}_{\mathrm{HG}_2})^2}{2 d^{ext}_{\mathrm{HG}_2} \left\|\overrightarrow{\mathrm{G_1G_2}}\right\|}\right). \tag{A.66}$$

We can then reformulate the sufficient condition as:

$$\max_{\mathrm{H}\in\mathcal{V}} \left\|\overrightarrow{\mathrm{G_TH}}\right\| < r_{\mathrm{T}} \tag{A.67}$$

where $\mathcal{V}$ denotes the set of vertices.

- Singularity avoidance:

$$\min_t \left\|\overrightarrow{\mathrm{P_1H}} \times \overrightarrow{\mathrm{P_1P_2}}\right\| > 0, \tag{A.68}$$

for which a sufficient condition is:

$$\left\|\overrightarrow{\mathrm{G_1G_2}}\right\| + r_{\mathrm{P}_1} + r_{\mathrm{P}_2} < l_1 + l_2, \tag{A.69}$$

reflecting the fact that the arms need to be long enough to prevent alignment when the pivots are farthest from each other.

Kinematic equations. As with the Cycloid Drawing Machine, we introduce the rotation angle of each external gear: θ_1 and θ_2. Again, they can be simply related by integrating the law of gearing (Equation A.1) and taking the constant to be 0:

$$\theta_i(t) = -\frac{\overline{r_{\mathrm{T}}}}{\overline{r_{\mathrm{G}_i}}}\theta_{\mathrm{T}}(t) \quad \forall t. \tag{A.70}$$

Let us now express the curve function (Equation A.53) via the gears' angles. First, the law of cosines applied to the triangle $\mathrm{HP_2P_1}$ gives

$$\overrightarrow{\mathrm{G_TH}} = \overrightarrow{\mathrm{G_TP_2}} + \overline{l_2} R_\alpha \frac{\overrightarrow{\mathrm{P_2P_1}}}{\left\|\overrightarrow{\mathrm{P_2P_1}}\right\|}, \tag{A.71}$$

where α is the angle $\widehat{\mathrm{HP_2P_1}}$, given by:

$$\alpha = \arccos\left(\frac{-\overline{l_1}^2 + \left\|\overrightarrow{\mathrm{P_2P_1}}\right\|^2 + \overline{l_2}^2}{2\overline{l_2}\left\|\overrightarrow{\mathrm{P_2P_1}}\right\|}\right). \tag{A.72}$$

The remaining vectors are given by:

$$\overrightarrow{\mathrm{G_TP_2}} = (\overline{r_{\mathrm{T}}} + \overline{r_{\mathrm{G}_2}}) \begin{bmatrix} \cos\overline{\theta_{12}} \\ \sin\overline{\theta_{12}} \end{bmatrix} + \overline{r_{\mathrm{P}_2}} \begin{bmatrix} \cos\theta_2 \\ \sin\theta_2 \end{bmatrix} \tag{A.73}$$

and

$$\overrightarrow{\mathrm{P_2P_1}} = (\overline{r_{\mathrm{T}}} + \overline{r_{\mathrm{G}_1}}) + \overline{r_{\mathrm{P}_1}} \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix} - \overrightarrow{\mathrm{G_TP_2}}. \tag{A.74}$$

Appendix B

Additional user study results for Chapter 3

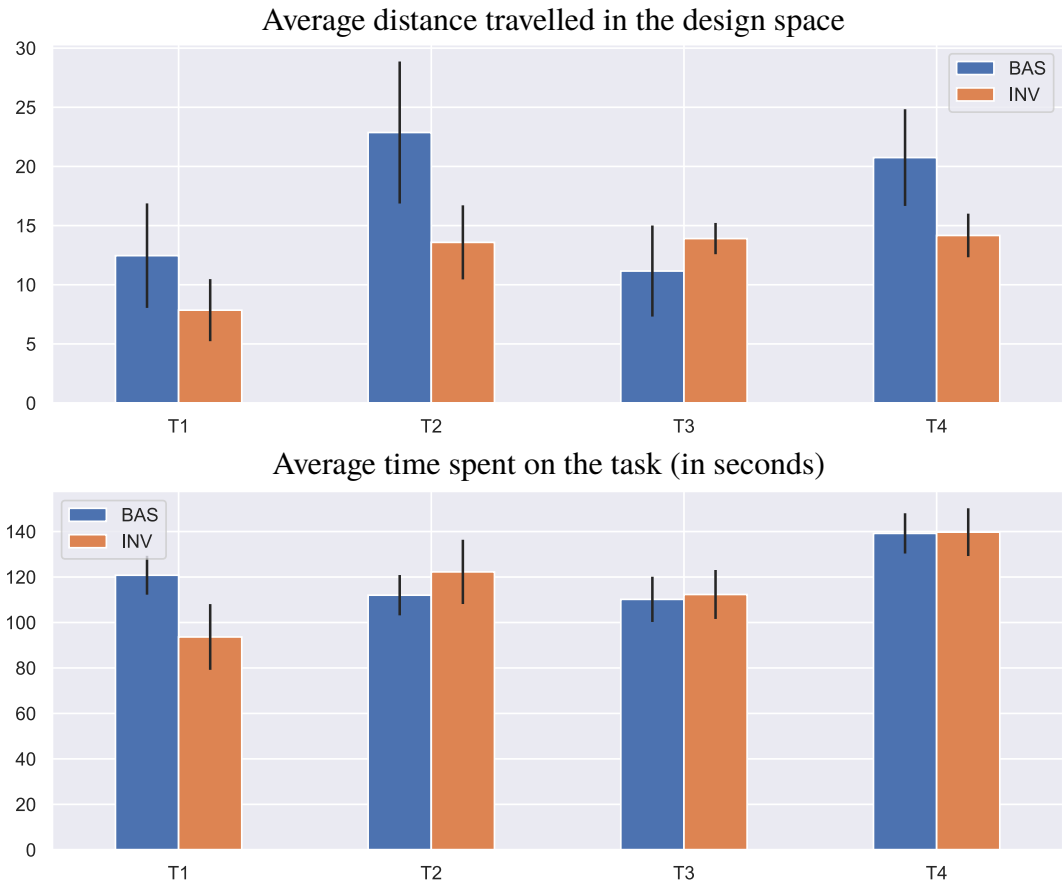


Figure B.1: Additional user study results. Lower values mean better results. Black bars show the standard error of the mean. "BAS" and "INV" respectively denote the base and invariant-space parameterizations.

Appendix C

Implementation details for Chapter 4

C.1 Primitives

This section describes in more detail the primitives implemented in our system. Please refer to the code for a complete description.

C.1.1 Simple primitives

Simple primitives describe single solid objects. Each such object is instantiated with design parameters describing its dimensions (e.g., width, length, height, etc.). Additionally, arbitrary physical parameters can be provided (e.g., mass, friction, restitution, etc.). Since simple primitives are constructed by combining rigid body shapes from Bullet, please refer to the documentation [27] to find available parameters. Importantly, a simple primitive without mass will be static in the simulation (and conversely, defining a mass makes it dynamic).

Most simple primitives are 3D: `Ball`, `Box`, `OpenBox`, `Cylinder`, `Capsule`, `Goblet`, `Track`. There is a single 2D primitive, `Plane`, and a '0D' primitive, `Empty`, whose role is to serve as parent of other primitives in the scene graph (similar to what exists in Blender).

C.1.2 Constraint primitives

Constraint primitives take one or two simple primitives and add a Bullet constraint. `Pivot`, as the name suggests, adds a pivot constraint (one degree of freedom), while `Fastener` glues objects together (0 degrees of freedom).

C.1.3 Complex primitives

Complex primitives are created by combining any number of the above primitives. For instance, `Lever` and `Pulley` respectively combine `Box` and `Cylinder` with a `Pivot`. Meanwhile, `DominoRun` provides an easier interface to define `Box`es aligned along a path. `TensionRope` combines several Bullet constraints with input primitives to emulate the behavior of a rope in tension (although the rope itself has no physical presence in the world, i.e., it is not involved in collisions). Lastly, `RopePulley` is an even more complex version combining input primitives with constraints and a callback function approximating the effect of a rope-pulley system. As with `TensionRope`, the rope is purely visual.

C.1.4 Primitives used in each scenario

The following simply provides the primitives involved in each of the scenarios presented in the evaluation. Please see the configuration files for a complete description.

- CAUSALITYSWITCH: `Ball`, `Box`, `DominoRun`, `Plane`, `Track`
- BALLRUN: `Ball`, `Box`, `Goblet`, `Lever`, `Track`
- LONGCHAIN: `Ball`, `Box`, `Cylinder`, `DominoRun`, `Fastener`, `Goblet`, `Lever`, `OpenBox`, `TensionRope`, `Track`
- TEAPOTADVENTURE: `Ball`, `Box`, `Cylinder`, `Fastener`, `Goblet`, `Lever`, `Pivot`, `RopePulley`, `Track`

C.2 Events

As described in Chapter 4, the occurrence of events in the simulation is checked with specific conditions based on rigid bodies' spatial transforms and their derivatives. `Toppling` simply requires one of the Euler angles to be greater than a threshold, and `NotMoving` checks that the position has not changed since the beginning. Meanwhile, `Falling`, `Pivoting`, `Rising` and `Stopping` all compare a component of the body's linear or angular velocity with a given threshold. `Contact` and its opposite, `NoContact`, use the simulator's internal collision check. `RollingOn` combines

`Pivoting` with `Contact`. Lastly, `Inclusion` uses the simulator's internal ray casting ability to check that one body is inside another.

C.3 Local robustness

In Chapter 4, we mention an evaluation dataset $\mathbf{X} \subset D$ created for each scenario $\mathbb{S}$ with design space D, decomposed as $\mathbf{X} = \mathbf{X}^+ \cup \mathbf{X}^- \cup \mathbf{X}^\varnothing$ (respectively successes, failures and impossible instances). The local robustness $\rho_l : D \times [0,1] \to [0,1]$ is then defined as

$$\rho_l(\mathbf{x}, \epsilon) = \begin{cases} \frac{|\mathcal{B}_\epsilon(\mathbf{x}) \cap \mathbf{X}^+|}{|\mathcal{B}_\epsilon(\mathbf{x}) \cap \{\mathbf{X}^+ \cup \mathbf{X}^-\}|} & \text{if } \mathbf{x} \in D \setminus D^\varnothing, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{B}_\epsilon(\mathbf{x})$ is the ball of radius ϵ centered at $\mathbf{x} \in D$.

In practice, the local robustness is computed differently depending on whether it is used as an objective function $\mathbf{x} \mapsto \rho_l(\mathbf{x}, 0.1)$ for baseline methods (B2) and (B3), or as a function of the error ϵ in Figures 15 and 16. In the former case, for each function call, d^2 physically valid points are uniformly sampled around $\mathbf{x}$ and simulated on the fly (d being the number of layout parameters). Therefore, the simulation budget B defined for the baselines directly translates to a maximum number of function evaluations $\lfloor B/d^2 \rfloor$. In the latter case, as explained in Chapter 4, the evaluation dataset $\mathbf{X}$ is obtained by physically checking and simulating points for each scenario; specifically, 100K points for causalitySwitch, and 1M points for ballRun, longChain and teapotAdventure. These points are drawn from the quasi-random Sobol sequence [129]. However, as there is little chance to find the solution $\mathbf{x}^*$ of a method in the evaluation dataset $\mathbf{X}$, the value of $\rho_l(\mathbf{x}^*, 0)$ is very likely to be 0, since the ball $\mathcal{B}_0(\mathbf{x}^*)$ is very likely to be empty. To counter this, we give a 'thickness' to the ball: i.e., we use a slightly modified local robustness $\epsilon \mapsto \rho_l(\mathbf{x}^*, \epsilon + \eta)$. Then, for each solution $\mathbf{x}^*$, we sample and simulate 100 points in the local neighborhood $\mathcal{B}_\eta(\mathbf{x}^*)$ and temporarily add them to $\mathbf{X}$. In our experiments, we used a thickness $\eta = 0.1 l_\epsilon$, with l_ϵ the length of a plot step ($l_\epsilon = 1/30$).

Appendix D

Exported layouts of chain reaction contraptions

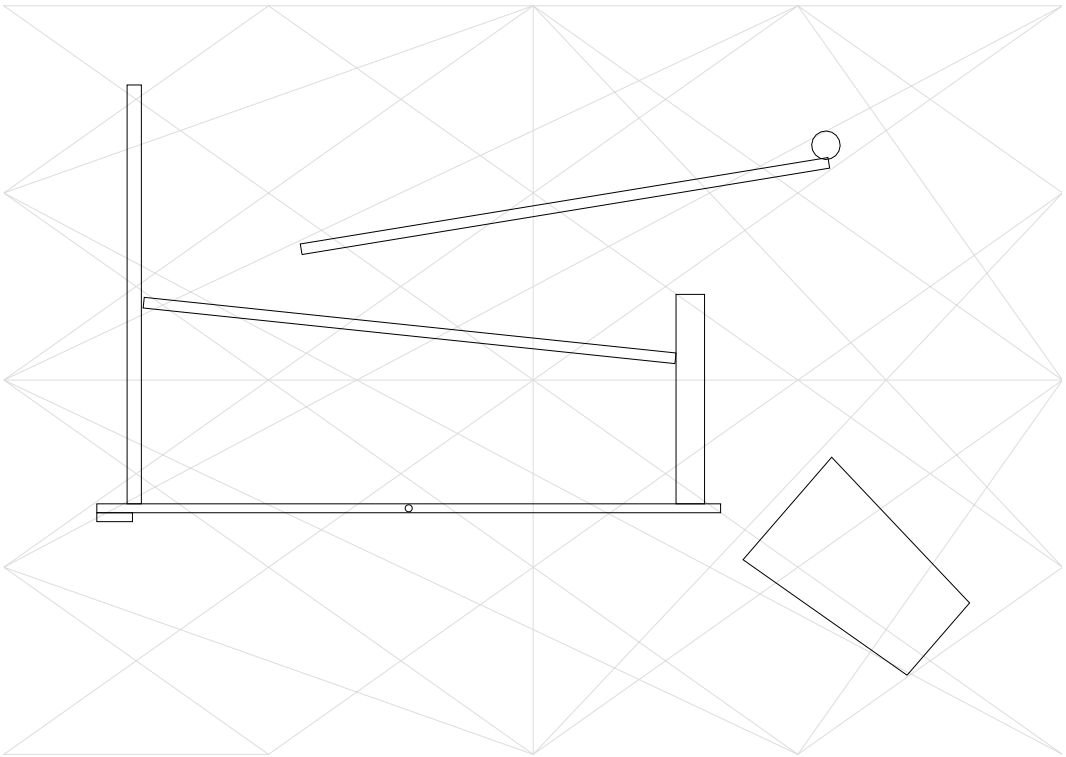


Figure D.1: Exported layout for BALLRUN.

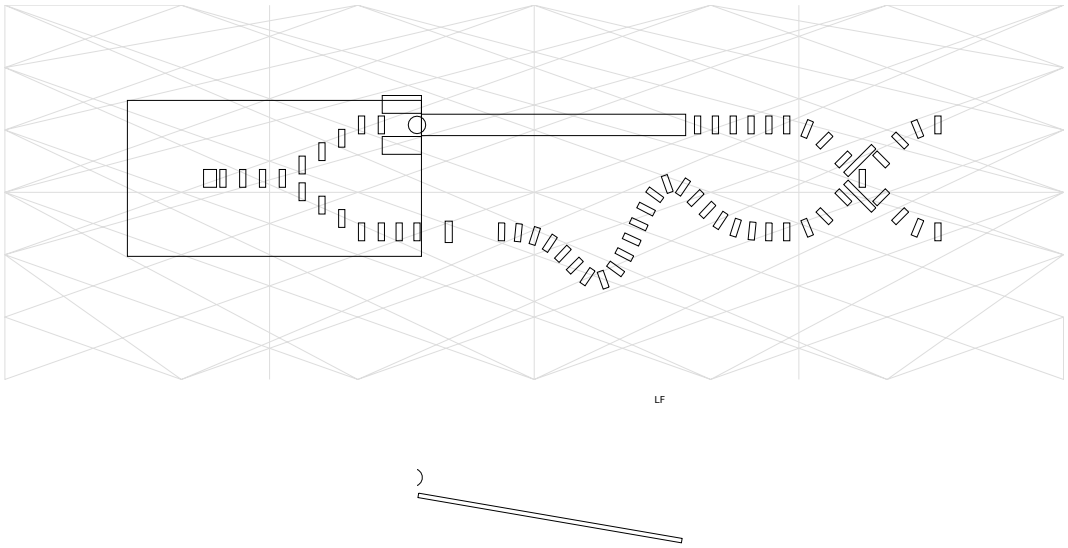


Figure D.2: Exported layout for CAUSALITYSWITCH (faster ball run). Top and side views.

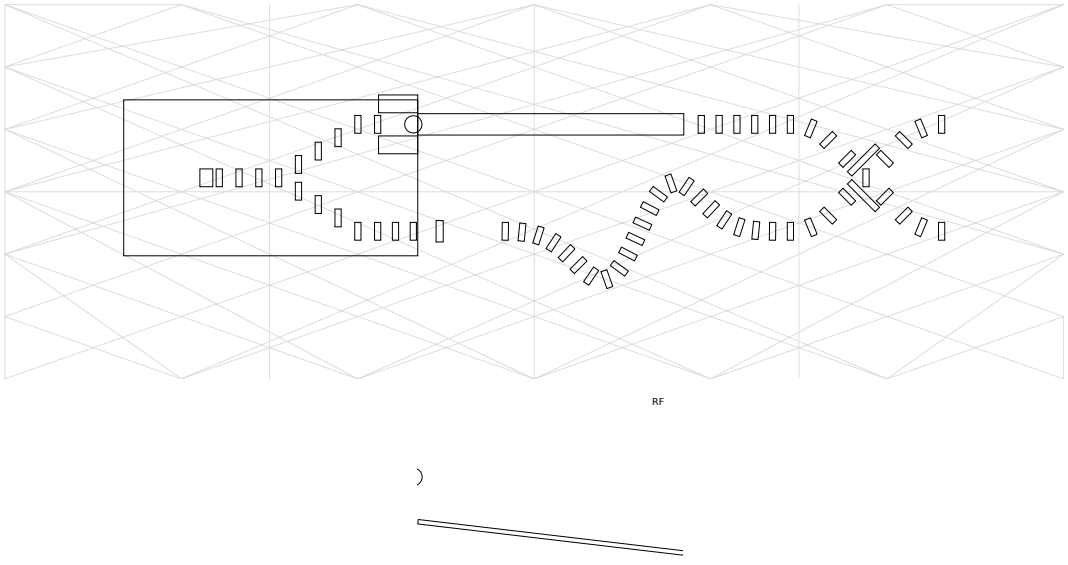


Figure D.3: Exported layout for CAUSALITYSWITCH (faster domino run). Top and side views.

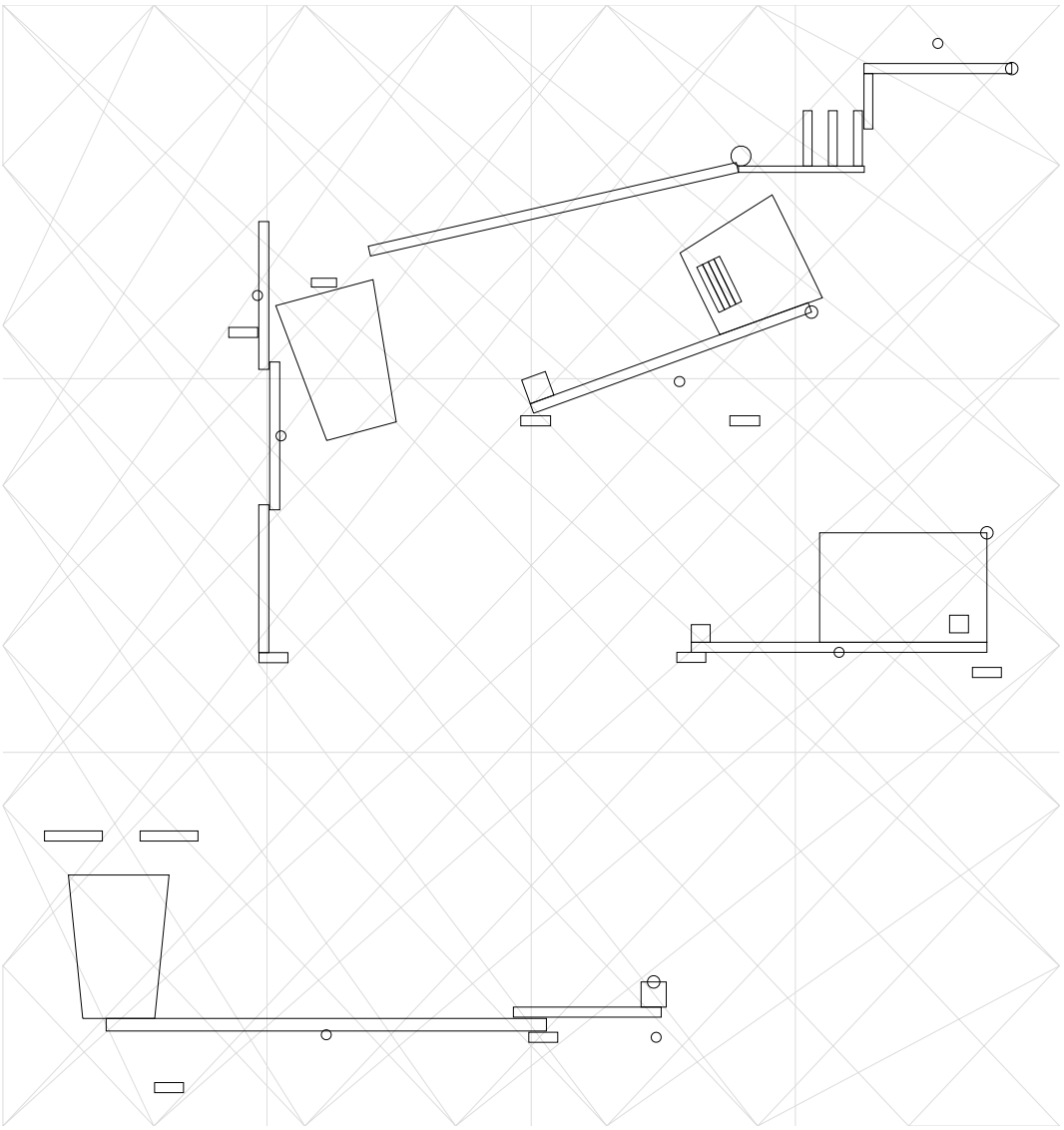


Figure D.4: Exported layout for longChain.

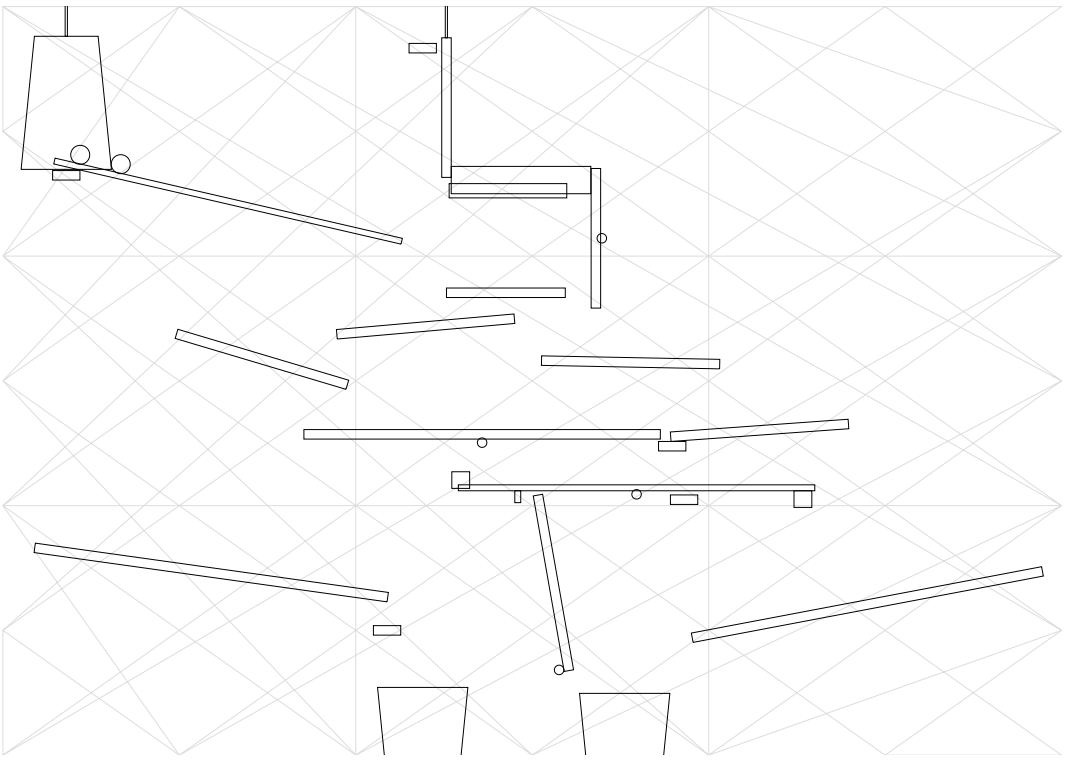


Figure D.5: Exported layout for TEAPOTADVENTURE.

Image credits

Figure 3.3 (p. 41):

(a) “Spirograph set” by Multicherry. [CC BY-SA 3.0], via Wikimedia (https://w.wiki/WDd).

(b) “Harmonograph” by Matemateca IME-USP / Rodrigo Tetsuo Argenton. [CC BY-SA 4.0], via Wikimedia (https://w.wiki/WDs).

(c) “Hoot-Nanny No. 1” by Northern Signal Company. [Public domain], via DrawingMachines.org (https://drawingmachines.org/post.php?id=164).

(d) “Jean Tinguely, cyclograveur” by Neural. [CC BY-NC-ND 2.0], via Flickr (https://www.flickr.com/photos/48831443@N00/2390041200).

Figure 4.1 (p. 75):

- Top: “Professor Butts and the Self-Operating Napkin” by Rube Goldberg. [Public domain], via Wikimedia (https://w.wiki/3gE).
- Bottom: “PSU Rube Goldberg 5” by Penn State. [CC BY-NC 2.0], via Flickr (https://www.flickr.com/photos/pennstatelive/6859199341/).

Bibliography

[1] George Adams. *Geometrical and graphical essays, containing a general description of the mathematical instruments used in geometry, civil and military surveying, levelling, and perspective; with many new practical problems, illustrated by thirty-four copper plates (third edition)*, pages 148–151. Printed by W. Glendinning for, and sold by W. and S. Jones, London, United Kingdom, 1803.

[2] Peter Ashwin, Jennifer Creaser, and Krasimira Tsaneva-Atanasova. Fast and slow domino regimes in transient network dynamics. *Physical Review E*, 96:052309, November 2017.

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

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

[5] Biing W. Bair. Computerized tooth profile generation of elliptical gears manufactured by shaper cutters. *Journal of Materials Processing Technology*, 122(2-3):139–147, 2002.

[6] Ronen Barzel, John F. Hughes, and Daniel N. Wood. Plausible motion simulation for computer graphics animation. *Proceedings of the Eurographics workshop on Computer animation and simulation*, 96:183–197, 1996.

[7] Jon L. Bentley and Thomas A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, September 1979.

[8] Amit H. Bermano, Thomas Funkhouser, and Szymon Rusinkiewicz. State of the art in methods and representations for fabrication-aware design. *Computer Graphics Forum*, 36(2):509–535, May 2017.

[9] James M. Bern, Kai-Hung Chang, and Stelian Coros. Interactive design of animated plushies. *ACM Transactions on Graphics*, 36(4), July 2017.

[10] Ian Berry, editor. *Chain Reaction: Rube Goldberg and Contemporary Art*. Tang Teaching Museum, July 2002.

[11] Gaurav Bharaj, Stelian Coros, Bernhard Thomaszewski, James Tompkin, Bernd Bickel, and Hanspeter Pfister. Computational design of walking automata. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, pages 93–100, New York, NY, USA, 2015. Association for Computing Machinery.

[12] Gaurav Bharaj, Danny Kaufman, Etienne Vouga, and Hanspeter Pfister. Metamorphs: bistable planar structures, 2018.

[13] Gaurav Bharaj, David I. W. Levin, James Tompkin, Yun Fei, Hanspeter Pfister, Wojciech Matusik, and Changxi Zheng. Computational design of metallophone contact sounds. *ACM Transactions on Graphics*, 34(6):1–13, 2015.

[14] Bernd Bickel, Cignoni Paolo, Malomo Luigi, and Pietroni Nico. State of the art on stylized fabrication. *Computer Graphics Forum*, 37(6):325–342, 2018.

[15] D. Braha and O. Maimon. The measurement of a design structural and functional complexity. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 28(4):527–535, July 1998.

[16] Jim Bumgardner. CDMS: Built with Processing and Processing.js. Retrieved from `http://wheelof.com/sketch`. Last accessed on 2020-07-02.

[17] Kaira M. Cabañas and Hans-Christian Von Herrmann. *Jean Tinguely: Retrospective*. Walther König, July 2016.

[18] Marco Ceccarelli. *Distinguished figures in mechanism and machine science: Their contributions and legacies*, page 230. Springer Netherlands, Netherlands, 2007.

[19] Duygu Ceylan, Wilmot Li, Niloy J. Mitra, Maneesh Agrawala, and Mark Pauly. Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics*, 32(6):1–11, 2013.

[20] Desai Chen, David I. W. Levin, Wojciech Matusik, and Danny M. Kaufman. Dynamics-aware numerical coarsening for fabrication design. *ACM Transactions on Graphics*, 36(4):84:1–84:15, July 2017.

[21] Xiang "Anthony" Chen, Stelian Coros, Jennifer Mankoff, and Scott E. Hudson. Encore: 3D printed augmentation of everyday objects with printed-over, affixed and interlocked attachments. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, UIST '15, pages 73–82, New York, NY, USA, 2015. Association for Computing Machinery.

[22] Xiang "Anthony" Chen, Jeeeun Kim, Jennifer Mankoff, Tovi Grossman, Stelian Coros, and Scott E. Hudson. Reprise: a design tool for specifying, generating, and customizing 3D printable adaptations on everyday objects. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 29–39, New York, NY, USA, 2016. Association for Computing Machinery.

[23] Xiaozhong Chen, Sheldon Andrews, Derek Nowrouzezahrai, and Paul G. Kry. Ballistic shadow art. In *Proceedings of the 43rd Graphics Interface Conference*, GI '17, pages 190–198, Waterloo, CAN, 2017. Canadian Human-Computer Communications Society.

[24] Philip Church, Justin Matheson, Xiaoying Cao, and Gilles Roy. Evaluation of a steerable 3D laser scanner using a double Risley prism pair. In John (Jack) N. Sanders-Reed and Jarvis (Trey) J. Arthur III, editors, *Degraded Environments: Sensing, Processing, and Display 2017*, volume 10197, pages 203–211. International Society for Optics and Photonics, SPIE, 2017.

[25] António Coelho, Pedro Branco, and João Martinho Moura. A brief overview on the evolution of drawing machines. In *Intelligent Technologies for Interactive Entertainment*, pages 14–24. Springer International Publishing, 2019.

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

[27] Erwan Coumans. Bullet Physics SDK. Retrieved from `https://github.com/bulletphysics/bullet3`. Last accessed on 2020-07-02.

[28] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (Wiley series in telecommunications and signal processing)*. Wiley-Interscience, New York, NY, USA, 2006.

[29] Robert Craig. The mechanical drawing of cycloids, the geometric chuck. In *Bridges London: Mathematics, Music, Art, Architecture, Culture*, pages 203–210. Tarquin Publications, 2006.

[30] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[31] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000.

[32] Joshua D. Deaton and Ramana V. Grandhi. A survey of structural and mul-

tidisciplinary continuum topology optimization: post 2000. *Structural and Multidisciplinary Optimization*, 49(1):1–38, January 2014.

[33] Ludovic Delchambre. Weighted principal component analysis: a weighted covariance eigendecomposition approach. *Monthly Notices of the Royal Astronomical Society*, 446(4):35453555, December 2014.

[34] Bailin Deng, Sofien Bouaziz, Mario Deuss, Alexandre Kaspar, Yuliy Schwartzburg, and Mark Pauly. Interactive design exploration for constrained meshes. *Computer-Aided Design*, 61(C):1323, April 2015.

[35] Bailin Deng, Sofien Bouaziz, Mario Deuss, Juyong Zhang, Yuliy Schwartzburg, and Mark Pauly. Exploring local modifications for constrained meshes. *Computer Graphics Forum*, 32(2pt1):11–20, 2013.

[36] Leo Dorst and Robert P. W. Duin. Spirograph theory: a framework for calculations on digitized straight lines. *IEEE transactions on pattern analysis and machine intelligence*, PAMI-6(5):632–639, 1984.

[37] Ian L. Dryden and Kanti V. Mardia. *Statistical shape analysis*, volume 4. Wiley Chichester, 1998.

[38] Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Information Systems Department, Technical University of Vienna, 1994.

[39] Katayoon Etemad, Sheelagh Carpendale, and Faramarz Samavati. Spirograph inspired visualization of ecological networks. *Proceedings the Workshop on Computational Aesthetics (Expressive 2014)*, pages 81–91, 2014.

[40] Joe Freedman. Cycloid Drawing Machine | leafpdx. Retrieved from `https://leafpdx.bigcartel.com/product/cycloid-drawing-machine`. Last accessed on 2020-07-02.

[41] Tsukasa Fukusato, Morihiro Nakamura, and Takeo Igarashi. Interactive design and optimization of free-formed returning boomerang. In *SIGGRAPH*

Asia 2018 Technical Briefs, SA '18, New York, NY, USA, 2018. Association for Computing Machinery.

[42] Yohsuke Furuta, Jun Mitani, Takeo Igarashi, and Yukio Fukui. Kinetic art design system comprising rigid body simulation. *Computer-Aided Design and Applications*, 7(4):533–546, 2010.

[43] Pablo Garcia. DrawingMachines.org. Retrieved from `https://drawingmachines.org/`. Last accessed on 2020-07-02.

[44] Akash Garg, Alec Jacobson, and Eitan Grinspun. Computational design of reconfigurables. *ACM Transactions on Graphics*, 35(4):1–14, 2016.

[45] Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. Skaterbots: optimization-based design and motion synthesis for robotic creatures with legs and wheels. *ACM Transactions on Graphics*, 37(4), July 2018.

[46] John S. Gero and Udo Kannengiesser. The function-behaviour-structure ontology of design. In *An Anthology of Theories and Models of Design*, pages 263–283. Springer, 2014.

[47] Michelle Ghrist and Eric Lane. Be careful what you assign: constant speed parametrizations. *Mathematics Magazine*, 86(1):3–14, 2013.

[48] Daniel Groeger, Elena Chong Loo, and Jürgen Steimle. HotFlex: post-print customization of 3D prints using embedded state change. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 420–432, New York, NY, USA, 2016. Association for Computing Machinery.

[49] Gurobi Optimization, Inc. Gurobi optimizer reference manual. Retrieved from `https://www.gurobi.com`, 2016.

[50] 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*, 37(13-14):1521–1536, 2018.

[51] Leon M. Hall. Trochoids, roses, and thorns - beyond the spirograph. *College Mathematics Journal*, 23(1):20–35, 1992.

[52] Godfrey Harold Hardy and Edward Maitland Wright. *An introduction to the theory of numbers (fifth edition)*, pages 23–24. Clarendon Press, Oxford, United Kingdom, 1979.

[53] Armand Hatchuel, Benoit Weil, and Pascal Le Masson. Towards an ontology of design: lessons from c-k design theory and forcing. *Research in Engineering Design*, 24(2):147–163, 2013.

[54] J.C. Helton, J.D. Johnson, C.J. Sallaberry, and C.B. Storlie. Survey of sampling-based methods for uncertainty and sensitivity analysis. *Reliability Engineering & System Safety*, 91(10):1175–1209, 2006. The Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004).

[55] Jennifer L. Herman, Lynn Hall, Deborah Kuzawa, Leah Wahlin, and Mary Faure. *Writing as knowing: creative knowing through multiple messaging modes in an engineering technical communications course*, pages 99–120. Springer International Publishing, Cham, 2017.

[56] R. Hu, M. Savva, and O. van Kaick. Functionality representations and applications for shape analysis. *Computer Graphics Forum*, 37(2):603–624, 2018.

[57] Matthias B. Hullin, Ivo Ihrke, Wolfgang Heidrich, Tim Weyrich, Gerwin Damberg, and Martin Fuchs. Computational Fabrication and Display of Material Appearance. In M. Sbert and L. Szirmay-Kalos, editors, *Eurographics 2013 - State of the Art Reports*. The Eurographics Association, 2013.

[58] Alexandra Ion, Johannes Frohnhofen, Ludwig Wall, Robert Kovacs, Mirela Alistar, Jack Lindsay, Pedro Lopes, Hsiang-Ting Chen, and Patrick Baudisch.

Metamaterial mechanisms. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 529–539, New York, NY, USA, 2016. ACM.

[59] Alexandra Ion, David Lindlbauer, Philipp Herholz, Marc Alexa, and Patrick Baudisch. Understanding metamaterial mechanisms. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, New York, NY, USA, 2019. Association for Computing Machinery.

[60] Reza N Jazar. *Theory of applied robotics: kinematics, dynamics, and control (2nd edition)*, pages 325–380. Springer US, 2010.

[61] Yunwoo Jeong, Han-Jong Kim, and Tek-Jin Nam. Mechanism perfboard: an augmented reality environment for linkage mechanism design and fabrication. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, New York, NY, USA, 2018. Association for Computing Machinery.

[62] Devendra Kalra and Alan H. Barr. Modeling with time and events in computer animation. *Computer Graphics Forum*, 11(3):45–58, May 1992.

[63] Craig S. Kaplan. Hypocycloid juggling patterns. In Eve Torrence, Bruce Torrence, Carlo Séquin, Douglas McKenna, Kristóf Fenyvesi, and Reza Sarhangi, editors, *Proceedings of Bridges 2016: Mathematics, Music, Art, Architecture, Education, Culture*, pages 71–78, Phoenix, Arizona, 2016. Tessellations Publishing.

[64] Hiroshi Katayama, Kenta Sawa, Reabook Hwang, Norio Ishiwatari, and Nobuyuki Hayashi. Analysis and classification of Karakuri technologies for reinforcement of their visibility, improvement and transferability: An attempt for enhancing lean management. In *Proceedings of PICMET '14 Conference: Portland International Center for Management of Engineering and Technology; Infrastructure and Service Integration*, pages 1895–1906, July 2014.

[65] A. B. Kempe. On a general method of describing plane curves of the nth degree by linkwork. *Proceedings of the London Mathematical Society*, s1-7(1):213–216, 1875.

[66] Marc C. Kennedy and Anthony O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

[67] Jeeeun Kim, Anhong Guo, Tom Yeh, Scott E. Hudson, and Jennifer Mankoff. Understanding uncertainty in measurement and accommodating its impact in 3D modeling and printing. In *Proceedings of the 2017 Conference on Designing Interactive Systems*, DIS '17, pages 1067–1078, New York, NY, USA, 2017. Association for Computing Machinery.

[68] Yilip Kim and Namje Park. Development and application of STEAM teaching model based on the Rube Goldberg's invention. In Sang-Soo Yeo, Yi Pan, Yang Sun Lee, and Hang Bae Chang, editors, *Computer Science and its Applications*, pages 693–698, Dordrecht, 2012. Springer Netherlands.

[69] Alexander Kobel. Automated generation of Kempe linkages for algebraic curves in a dynamic geometry system. Bachelor's thesis, Saarland University, Faculty of Natural Sciences and Technology, Department of Computer Science, Saarbrucken, Germany, September 2008.

[70] Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J. Mitra. Creating works-like prototypes of mechanical objects. *ACM Transactions on Graphics*, 33(6):1–9, 2014.

[71] Dieter Kraft. A software package for sequential quadratic programming. Technical report, Institut fuer Dynamik der Flugsysteme, Oberpfaffenhofen, 1988.

[72] Michelle Y. Kuo and Anna Engberg-Pedersen. *Olafur Eliasson: Experience*. Phaidon Press, London, United Kingdom, 2018.

[73] Nikolas Lamb, Sean Banerjee, and Natasha Kholgade Banerjee. Automated reconstruction of smoothly joining 3D printed restorations to fix broken objects. In *Proceedings of the ACM Symposium on Computational Fabrication*, SCF '19, New York, NY, USA, 2019. Association for Computing Machinery.

[74] Mandy Lange, Dietlind Zühlke, Olaf Holz, and Thomas Villmann. Applications of lp-norms and their smooth approximations for gradient based learning vector quantization. In *ESANN*, pages 271–276, Bruges, April 2014.

[75] Steffen L. Lauritzen. Causal inference from graphical models. *Complex stochastic systems*, pages 63–107, 2001.

[76] Quentin Letts. Lights! Camera! Retake! *The Telegraph*, April 2003.

[77] Anhu Li, Wansong Sun, Wanli Yi, and Qiyou Zuo. Investigation of beam steering performances in rotation Risley-prism scanner. *Optics Express*, 24(12):12840–12850, June 2016.

[78] Honghua Li, Ruizhen Hu, Ibraheem Alhashim, and Hao Zhang. Foldabilizing furniture. *ACM Transactions on Graphics*, 34(4):90:1–90:12, July 2015.

[79] Jian Li, Sheldon Andrews, Krisztian G. Birkas, and Paul G. Kry. Task-based design of cable-driven articulated mechanisms. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*, SCF '17, New York, NY, USA, 2017. Association for Computing Machinery.

[80] M. Lin, T. Shao, Y. Zheng, N. J. Mitra, and K. Zhou. Recovering functional mechanical assemblies from raw scans. *IEEE Transactions on Visualization and Computer Graphics*, 24(3):1354–1367, March 2018.

[81] Ye Lin and Romain Vuillemot. Spirograph designs for ambient display of tweets. In *IEEE VIS 2013*, Atlanta, GA, United States, 2013. IEEE, IEEE.

[82] Yang Liu and J. Michael McCarthy. Design of mechanisms to draw trigonometric plane curves. *Journal of Computing and Information Science in Engineering*, 9(2), 2017.

[83] Li-Ke Ma, Yizhonc Zhang, Yang Liu, Kun Zhou, and Xin Tong. Computational design and fabrication of soft pneumatic objects with desired deformations. *ACM Transactions on Graphics*, 36(6):239:1–239:12, November 2017.

[84] Chandan Mahapatra, Jonas Kjeldmand Jensen, Michael McQuaid, and Daniel Ashbrook. Barriers to end-user designers of augmented fabrication. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, New York, NY, USA, 2019. Association for Computing Machinery.

[85] Gerald F. Marshall. Risley prism scan patterns. In Leo Beiser, Stephen F. Sagan, and Gerald F. Marshall, editors, *Optical Scanning: Design and Application*, volume 3787, pages 74–86. International Society for Optics and Photonics, SPIE, 1999.

[86] Tobias Martin, Nobuyuki Umetani, and Bernd Bickel. OmniAD: data-driven omni-directional aerodynamics. *ACM Transactions on Graphics*, 34(4), July 2015.

[87] Jonàs Martínez, Jérémie Dumas, and Sylvain Lefebvre. Procedural voronoi foams for additive manufacturing. *ACM Transactions on Graphics*, 35:1–12, 2016.

[88] Peter Marzio. *Rube Goldberg: his life and work*. Harper & Row, New York, 1973.

[89] Mark Masry and H. Lipson. A sketch-based interface for iterative design and analysis of 3D objects. In Joaquim Armando Pires Jorge and Takeo Igarashi, editors, *Eurographics Workshop on Sketch-Based Interfaces and Modeling*. The Eurographics Association, 2005.

[90] Hermann G. Matthies. Quantifying uncertainty: modern computational representation of probability and applications. In Adnan Ibrahimbegovic and

Ivica Kozar, editors, *Extreme Man-Made and Natural Hazards in Dynamics of Structures*, pages 105–135, Dordrecht, 2007. Springer Netherlands.

[91] Asla Medeiros e Sá, Karina Rodriguez Echavarria, Nico Pietroni, and Paolo Cignoni. State of the art on functional fabrication. In *Eurographics Workshop on Graphics for Digital Fabrication*. The Eurographics Association, 2016.

[92] Vittorio Megaro, Espen Knoop, Andrew Spielberg, David I. W. Levin, Wojciech Matusik, Markus Gross, Bernhard Thomaszewski, and Moritz Bächer. Designing cable-driven actuation networks for kinematic chains and trees. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, New York, NY, USA, 2017. Association for Computing Machinery.

[93] Vittorio Megaro, Bernhard Thomaszewski, Damien Gauge, Eitan Grinspun, Stelian Coros, and Markus Gross. ChaCra: an interactive design system for rapid character crafting. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '14, pages 123–130, Goslar, DEU, 2015. Eurographics Association.

[94] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3D-printable robotic creatures. *ACM Transactions on Graphics*, 34(6):1–9, 2015.

[95] Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus Gross, and Bernhard Thomaszewski. A computational design tool for compliant mechanisms. *ACM Transactions on Graphics*, 36(4), July 2017.

[96] Niloy Mitra, Michael Wand, Hao (Richard) Zhang, Daniel Cohen-Or, Vladimir Kim, and Qi-Xing Huang. Structure-aware shape processing. In *SIGGRAPH Asia 2013 Courses*, SA '13, New York, NY, USA, 2013. Association for Computing Machinery.

[97] Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh

Agrawala. Illustrating how mechanical assemblies work. *ACM Transactions on Graphics*, 29(4):58:1–11, 2010.

[98] Morihiro Nakamura, Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. An interactive design system of free-formed bamboo-copters. *Computer Graphics Forum*, 35(7):323–332, October 2016.

[99] Gen Nishida, Adrien Bousseau, and Daniel G. Aliaga. Multi-pose interactive linkage design. *Computer Graphics Forum*, 38(2):277–289, 2019.

[100] Paul O'Dowd. Robot poetics: toward a materially sensitive drawing robot. In *Proceedings of the Conference on Electronic Visualisation and the Arts*, EVA '18, pages 104–112. BCS Learning & Development Ltd., 2018.

[101] Hyunjoo Oh, Jeeeun Kim, Cory Morales, Mark Gross, Michael Eisenberg, and Sherry Hsi. FoldMecha: exploratory design and engineering of mechanical papercraft. In *Proceedings of the Eleventh International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '17, pages 131–139, New York, NY, USA, 2017. Association for Computing Machinery.

[102] Elaine O'Hanrahan. The contribution of Desmond Paul Henry (1921–2004) to twentieth-century computer art. *Leonardo*, 51(2):156–162, 2018.

[103] Zofia Pawlikowska-Brożek. On mathematical life in Poland. In Catherine Goldstein, Jeremy Gray, and Jim Ritter, editors, *Mathematical Europe: history, myth, identity*, chapter 13, pages 291–302. Maison des sciences de l'homme, Paris, 1996.

[104] Yichen Peng, Yuki Mishima, Yamato Igarashi, Ryoma Miyauchi, Masahiro Okawa, Haoran Xie, and Kazunori Miyata. Sketch2Domino: interactive chain reaction design and guidance. In *2020 Nicograph International (NicoInt)*, pages 32–38, 2020.

[105] Jesús Pérez, Miguel A. Otaduy, and Bernhard Thomaszewski. Computa-

tional design and automated fabrication of Kirchhoff-Plateau surfaces. *ACM Transactions on Graphics*, 36(4), July 2017.

[106] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[107] Steve Price. 10 Chain Reaction Tips | Build a Rube Goldberg Machine. Retrieved from `https://www.youtube.com/watch?v=p8Wwq_B5S7I`, March 2017. Last accessed on 2020-07-02.

[108] M. O. Riedl and R. M. Young. From linear story generation to branching story graphs. *IEEE Computer Graphics and Applications*, 26(3):23–31, May 2006.

[109] Dustyn Roberts. *Making things move: DIY mechanisms for inventors, hobbyists, and artists*. McGraw-Hill Professional, 2010.

[110] Timothy J. Robinson, Connie M. Borror, and Raymond H. Myers. Robust parameter design: a review. *Quality and Reliability Engineering International*, 20(1):81–101, 2004.

[111] Robin Roussel, Marie-Paule Cani, Jean-Claude Léon, and Niloy J. Mitra. SPIROU: constrained exploration for mechanical motion design. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*, SCF '17, pages 7:1–7:11, New York, NY, USA, 2017. ACM.

[112] Robin Roussel, Marie-Paule Cani, Jean-Claude Léon, and Niloy J. Mitra. Exploratory design of mechanical devices with motion constraints. *Computers & Graphics*, 74:244–256, 2018.

[113] Robin Roussel, Marie-Paule Cani, Jean-Claude Léon, and Niloy J. Mitra. Designing chain reaction contraptions from causal graphs. *ACM Transactions on Graphics*, 38(4):43:1–43:13, 2019.

[114] Norman Routledge. Computing farey series. *The Mathematical Gazette*, 92(523):55–62, 2008.

[115] Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. SketchChair: an all-in-one chair design system for end users. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '11, pages 73–80, New York, NY, USA, 2010. Association for Computing Machinery.

[116] Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. Design and fabrication by example. *ACM Transactions on Graphics*, 33(4), July 2014.

[117] 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.

[118] Adriana Schulz, Harrison Wang, Eitan Crinspun, Justin Solomon, and Wojciech Matusik. Interactive exploration of design trade-offs. *ACM Transactions on Graphics*, 37(4):131:1–131:14, July 2018.

[119] Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. Interactive design space exploration and optimization for CAD models. *ACM Transactions on Graphics*, 36(4):157:1–157:14, July 2017.

[120] Daniel L. Schwartz and Mary Hegarty. Coordinating multiple representations for reasoning about mechanical devices. In *Proceedings of the AAAI Spring Symposium on Cognitive and Computational Models of Spatial Representation*, Menlo Park, CA, 1996. AAAI Press.

[121] Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[122] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: a review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, January 2016.

[123] John Sharp. Linkages to Op-art. In Reza Sarhangi and John Sharp, editors, *Bridges London: Mathematics, Music, Art, Architecture, Culture*, pages 497–502, London, United Kingdom, 2006. Tarquin Publications.

[124] John Sharp. Rigge envelopes as art inspiration. In Reza Sarhangi and Carlo H. Séquin, editors, *Proceedings of Bridges 2011: Mathematics, Music, Art, Architecture, Culture*, pages 171–178, Phoenix, Arizona, 2011. Tessellations Publishing.

[125] Andrew Neil Sherwood, Milorad Nikolic, John William Humphrey, and John Peter Oleson. *Greek and Roman technology: a sourcebook of translated Greek and Roman texts*. Routledge, London, United Kingdom, 2019.

[126] Maria Shugrina, Ariel Shamir, and Wojciech Matusik. Fab Forms: customizable objects for fabrication with validity and geometry caching. *ACM Transactions on Graphics*, 34(4):100:1–100:12, July 2015.

[127] Kate Sierzputowski. Crank Out Infinite Geometric Designs With The Wooden Cycloid Drawing Machine. *Colossal*, March 2016.

[128] Kate Sierzputowski. Metal and Wood Drawing Machines by James Nolan Gandy Form Mesmerizing Multi-Color Ellipses. *Colossal*, July 2019.

[129] I.M. Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[130] Peng Song, Xiaofei Wang, Xiao Tang, Chi-Wing Fu, Hongfei Xu, Ligang Liu, and Niloy J. Mitra. Computational design of wind-up toys. *ACM Transactions on Graphics*, 36(6):238:1–238:13, November 2017.

[131] Matthew Sparkes. The art of motion control (Final Word). *Computing Control Engineering Journal*, 17(5):48–48, 2006.

[132] Timothy Sun and Changxi Zheng. Computational design of twisty joints and puzzles. *ACM Transactions on Graphics*, 34(4):1–11, 2015.

[133] Jack Tait. Near chaos in graphic drawings from analogue machines. In Jonathan P. Bowen, Graham Diprose, Nick Lambert, and Jon Weinel, editors, *Proceedings of the Conference on Electronic Visualisation and the Arts*, Workshops in Computing, pages 103–111. BCS Learning & Development Ltd., 2019.

[134] T. Takahashi, J. Zehnder, H. G. Okuno, S. Sugano, S. Coros, and B. Thomaszewski. Computational design of statically balanced planar spring mechanisms. *IEEE Robotics and Automation Letters*, 4(4):4438–4444, October 2019.

[135] Takuto Takahashi and Hiroshi G. Okuno. Design and implementation of programmable drawing automata based on cam mechanisms for representing spatial trajectory. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 450–455, October 2018.

[136] Jerry O. Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. Exploratory modeling with collaborative design spaces. *ACM Transactions on Graphics*, 28(5):1–10, December 2009.

[137] Alexander Teibrich, Stefanie Mueller, François Guimbretière, Robert Kovacs, Stefan Neubert, and Patrick Baudisch. Patching physical objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, UIST '15, pages 83–91, New York, NY, USA, 2015. Association for Computing Machinery.

[138] Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. Computational design of linkage-based characters. *ACM Transactions on Graphics*, 33(4):64:1–64:9, July 2014.

[139] Toyota Global. econohito season2 | The Toyota "Karakuri" meister leading our continuous improvements | Toyota. Retrieved from `https://www.youtube.com/watch?v=Om2txc-Rq5s`, May 2018. Last accessed on 2020-07-02.

[140] Steven Turner. Demonstrating harmony: Some of the many devices used to produce Lissajous curves before the oscilloscope. *Rittenhouse*, 11:33–51, 1996.

[141] Dylan Tweney. How OK Go's Amazing Rube Goldberg Machine Was Built. *Wired*, March 2010.

[142] Nobuyuki Umetani and Bernd Bickel. Learning three-dimensional flow for interactive aerodynamic design. *ACM Transactions on Graphics*, 37(4):89:1–89:10, July 2018.

[143] Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Transactions on Graphics*, 31(4):1–11, 2012.

[144] Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. Pteromys: interactive design and optimization of free-formed free-flight model airplanes. *ACM Transactions on Graphics*, 33(4):1–10, 2014.

[145] J. M. J. van Leeuwen. The domino effect. *ArXiv Physics e-prints*, January 2004.

[146] J. M. J. van Leeuwen. Domino magnification. *ArXiv e-prints*, January 2013.

[147] Abel Vincze. HTML Spirograph. Retrieved from `http://htmlspirograph.com`. Last accessed on 2020-07-02.

[148] Gordon Walker. The general theory of roulettes. *National Mathematics Magazine*, 12(1):21–26, 1937.

[149] Wikipedia contributors. Rube Goldberg machine — Wikipedia, The Free Encyclopedia. Retrieved from `https://en.wikipedia.org/w/index.`

php?title=Rube_Goldberg_machine&oldid=962004698, 2020. Last accessed on 2020-07-02.

[150] Hao Xu, Tianwen Fu, Peng Song, Chi-Wing Fu, Mingjun Zhou, and Niloy J. Mitra. Computational design and optimization of non-circular gears. *Computer Graphics Forum*, 2020.

[151] Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. Bend-It: design and fabrication of kinetic wire characters. *ACM Transactions on Graphics*, 37(6), December 2018.

[152] Kai Xu, Vladimir G. Kim, Qixing Huang, Niloy Mitra, and Evangelos Kalogerakis. Data-driven shape analysis and processing. In *SIGGRAPH ASIA 2016 Courses*, SA '16, New York, NY, USA, 2016. Association for Computing Machinery.

[153] Yong-Liang Yang, Yi-Jun Yang, Helmut Pottmann, and Niloy J. Mitra. Shape space exploration of constrained meshes. *ACM Transactions on Graphics*, 30(6):1–12, December 2011.

[154] Ye Yuan, Changxi Zheng, and Stelian Coros. Computational design of transformables. *Computer Graphics Forum*, 37(8):103–113, 2018.

[155] Ran Zhang, Thomas Auzinger, Duygu Ceylan, Wilmot Li, and Bernd Bickel. Functionality-aware retargeting of mechanisms to 3D shapes. *ACM Transactions on Graphics*, 36(4), July 2017.

[156] Lifeng Zhu, Benyi Xie, Yongjie Jessica Zhang, and Lap-Fai Yu. Cartonist: automatic synthesis and interactive exploration of nonstandard carton design. *Computer-Aided Design*, 114:215–223, 2019.

[157] Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided mechanical toy modeling. *ACM Transactions on Graphics*, 31(6):127:1–127:10, November 2012.