Assembly-aware Design of Printable Electromechanical Devices

Ruta Desai
Carnegie Mellon University
Pittsburgh, PA, USA
rutad@cmu.edu

James McCann
Carnegie Mellon University
Pittsburgh, PA, USA
jmccann@cs.cmu.edu

Stelian Coros
ETH
Zurich, Switzerland
scoros@inf.ethz.ch

Figure 1. Our computational design system allows casual users to make complex 3D printed devices with integrated off-the-shelf electromechanical components. Chirpy – a smart crib monitoring toy made with our system is shown here.

ABSTRACT

From smart toys and household appliances to personal robots, electromechanical devices play an increasingly important role in our daily lives. Rather than relying on gadgets that are mass-produced, our goal is to enable casual users to custom-design such devices based on their own needs and preferences. To this end, we present a computational design system that leverages the power of digital fabrication and the emergence of affordable electronics such as sensors and microcontrollers. The input to our system consists of a 3D representation of the desired device's shape, and a set of user-preferred off-the-shelf components. Based on this input, our method generates an optimized, 3D printable enclosure that can house the required components. To create these designs automatically, we formalize a new spatio-temporal model that captures the entire assembly process, including the placement of the components within the device, mounting structures and attachment strategies, the order in which components must be inserted, and collision-free assembly paths. Using this model as a technical core, we then leverage engineering design guidelines and efficient numerical techniques to optimize device designs. In a user study, which also highlights the challenges of designing such devices, we find our system to be effective in reducing the entry barriers faced by casual users in creating such devices. We further demonstrate the versatility of our approach by designing and fabricating three devices with diverse functionalities.

UIST'18, October 14–17, 2018, Berlin, Germany

DOI: http://dx.doi.org/10.475/123_4

CCS Concepts

•**Applied computing** → **Computer-aided design;** •**Human-centered computing** → *Graphical user interfaces;*

Author Keywords

Computational Design; Digital Fabrication; Optimization

INTRODUCTION

The emergence of easy-to-use computer-aided design (CAD) software, such as SketchUp and Autodesk's 123D suite, has enabled the general public to create content for digital fabrication [5, 2]. The goal of our work is to make equally accessible the creation of 3D printed devices capable of rich interactions with the world around them. Our work is inspired by *Voxel8*, a recently introduced 3D electronics printer [52]. Voxel8 employs a print-in-place strategy where external components are completely encased into 3D printed objects during the fabrication process. Such embedding of interactive elements during the printing process have also been explored for optical and pneumatic 3D printed interactive devices [54, 50]. While very promising for certain application domains, this approach does not allow the embedded components to be accessed for repair or upgrades after the device is created. This limitation highlights the importance of assembly-based approaches, more prevalent in traditional manufacturing wherein complex artifacts are designed to be put together, and taken apart as needed. Inspired by this, we also advocate an *assembly-aware* design approach, where off-the-shelf components are mounted within a 3D printed *enclosure* as a post process.

To design a physical device according to a desired functionality, one must first choose a suitable set of electromechanical components. The layout of the components within the device has to then be generated. For designing functional devices, the problem of choosing right components for a functionality, and layout design are both important but complementary.

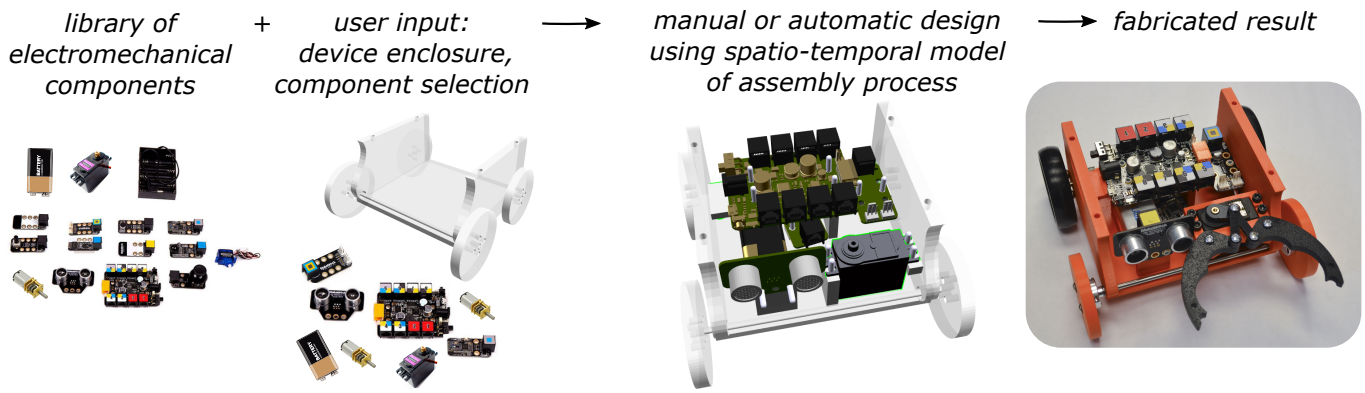

Figure 2. Overview: Given a user-defined device enclosure, and a set of components selected from a library, our system enables an assembly-aware design of the device (wires not shown for clarity).

Many solutions have started to emerge for the former task. In particular, Censi proposed an approach to select discrete robot components, including batteries and actuators, based on constraints operating on mixed discrete and continuous variables [10]. Likewise, Ramesh *et al.* also use constraint solvers to generate circuit level block diagrams, starting from user-specified requirements and a library of available components [41]. A system to automate component selection given robot function in natural language has also been proposed [31]. Therefore, we focus on the latter problem of device layout design in this paper. Layout design for physical devices is a highly challenging task. The placement of the components within the enclosure, the configuration of mounting structures and the assembly process itself (e.g. assembly order, collision-free assembly paths, attachment strategies, etc) are all highly coupled and have to be concurrently considered. The difficulty of this problem led to the emergence of the Design for Assembly (DFA) sub-field of engineering, where product design is studied from the point of view of 'ease of assembly' [8]. However, within the computer-aided design and manufacturing community (CAD-CAM), product design is an iterative but sequential process. A product's layout design (parts, and their connections) using DFA guidelines is first created by hand. For this manually created layout design, automatic assembly sequence generators [23] and industrial DFA softwares are used to evaluate the assembly process based on metrics such as time and cost [8], which allow the input design to be refined. However, creating an initial design or adapting an existing one remains a time-consuming, manually intensive task that is well beyond the capabilities of non-experts.

Overview and contributions

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

Apart from modeling the spatio-temporal assembly process, we also develop an efficient algorithm to concurrently optimize all aspects of this model. The algorithm we propose couples a Markov Chain Monte Carlo based optimization strategy [19] with a gradient-based method, while utilizing heuristics that encode insights from the CAD community. The resulting scheme handles both continuous and discrete model parameters, and features increased robustness to local minima. We demonstrate the effectiveness of our computational approach by designing and fabricating an assortment of electromechanical devices. Our examples are representative of the types of devices available in online community-driven repositories [21, 48]. Each device features unique form factors and functional capabilities, and employs standard off-the-shelf electromechanical components embedded into 3D printed enclosure. We also show that such devices are time consuming and difficult to design, especially by non-experts, through a user-study. While many participants failed to design certain devices in 45 minutes, our computational approach created valid designs for those devices within 4 minutes.

RELATED WORK

Design for fabrication: The research community has contributed heavily to the development of powerful computational tools that fuel the personal fabrication revolution. Examples include methods to generate 3D printable objects that are lightweight yet strong [28, 47], objects whose optimized mass distribution allows them to stand, spin or float stably [35, 6], and mechanical automata capable of creating compelling motions [13, 11]. These tools share the same high-level goal as ours: empowering casual users in creating complex physical artifacts without requiring domain specific knowledge.

However, these artifacts are limited by the abilities of 3D printers. To create objects with diverse functionalities, we aim to develop a computational framework to seamlessly integrates off-the-shelf components and 3D printed structures.

Combining off-the-shelf components with 3D printed parts allows us to harness the best of both worlds – traditionally manufactured parts offer cost-effectiveness, durability, and advanced functionality while 3D printing allows customization. Motivated by this, other researchers have also recently proposed tools for fabricating 3D printed artifacts with sensors and motors such as walking creatures [30, 16], and multicopters [17]. Our system intends to provide similar ease of design for making generic smart devices with a wide range of functional repertoires.

3D printed smart objects: Researchers in the HCI community have also been interested in enabling users to build 3D printed objects with embedded electronics [51, 44, 24, 3]. Savage *et al.* [44] and Jones *et al.* [24] present a fabrication pipeline using tangible means, while Ashbrook *et al.* [3] use an augmented fabrication system to enable fabrication of functional and assemblable objects with embedded components. Tools for tangibly designing laser-cut custom enclosures for prototyping interactive objects [45], and retrofitting existing devices with sensors and actuators [40] have also been proposed. Some of these systems also automatically create mounting structures for the components. However, the onus of deciding component layout such that the design is assemblable lies on the end user in their systems. Our system offloads many of the decisions requiring engineering knowledge from the user, while involving them in the creative aspects of the design process. In a complementary approach, Weichel *et al.* [53] propose to design an enclosure that 'fits' all the desired components. Instead, we assume a fixed, pre-defined enclosure for applications where the users might care about functionality or aesthetics, such as an owl-shaped smart toy (Fig. 1).

Layout design and Stochastic optimization: Computational layout design has been addressed for a wide range of applications such as webpage and document designs [38], VLSI design [26], city and architectural layouts [32], furniture layout for interior design [58, 33], and for the design of simulated worlds [18, 57]. Many of these approaches apply Markov Chain Monte Carlo (MCMC) based techniques to optimize the layouts, owing to the highly multimodal nature of the layout cost function. Motivated by their success, we also apply MCMC based optimization for our problem. We are also inspired by other stochastic optimization approaches that utilize gradient information for efficient sampling such as Hamiltonian Monte Carlo (HMC) [36], Latin Complement Sampling (LCS) [7], and Sequential Monte Carlo mixed with gradient descent [27]. While HMC and LCS are more suitable for continuous domains, our cost function has both discrete and continuous parameters. To handle the additional assembly and fabrication constraints that our problem poses, we propose an interleaved optimization strategy.

Assembly planning: Assembly planning is a well-studied problem in automated manufacturing and robotics [43, 55, 56, 23]. Traditionally, an assembly planner computes all geometrically feasible sequences of assembly operations, given a known layout. However, a design's layout and assembly plan generation are highly coupled. Therefore, our solution concurrently optimizes with the help of a new model that captures spatio-temporal aspects of the assembly process.

DESIGN PROCESS

Our system targets users interested in creating an electromechanical device by mounting off-the-shelf components in a 3D-printed enclosure. To create such a device, users must make informed decisions about four connected aspects of the design:

1. **Component selection.** Which electromechanical components to include.
2. **Layout design.** Where to place the selected components within the device's enclosure.
3. **Mounting structures.** Where to add material to the enclosure so that components can be fastened at their selected locations.
4. **Assembly plan.** How to assemble the components onto their mounting structures.

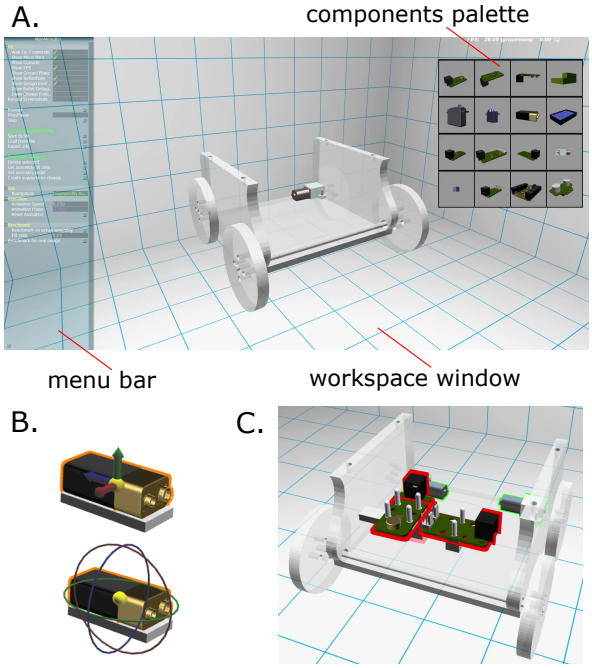

Figure 3. (A) Our user interface is shown here with its three main elements – components palette, main workspace window, and editing menu bar. (B) Translation and rotation widgets can be used to manually configure a component. (C) In order to provide guidance to the user, our system highlights components that lead to infeasible designs in red.

Unfortunately, these four aspects are all intertwined – selection of components constrains layout, which in turn dictates mounting, which may end up blocking or complicating assembly. Further, coming up with an assembly plan requires users to reason not only about component positions, but about how – and in what order – those components should be moved to reach their final positions.

Our system (Fig. 2) simplifies the design process by providing tools to visualize conflicts during manual design, as well as providing the option to automatically and simultaneously determine a layout, mounting design, and assembly plan from a set of components and an enclosure.

USER INTERFACE

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

In *manual design mode*, the user uses translation and rotation widgets to place each component (Fig. 3(B)); and sets the assembly plan using options in the menu bar. Our system provides assistance by highlighting components that lead to design infeasibility (Fig. 3(C)). Such infeasibility may occur when components are colliding with each-other, or when a component or its fasteners' assembly is blocked by other components in the device. In contrast, *automatic design mode* determines a valid layout and assembly plan using optimization. As it optimizes, our system displays the current best configuration, and allows users to pause and modify the design if desired.

Once the device is designed, users can export the enclosure (including synthesized mounting structures) for 3D printing, and the assembly plan can be animated for guidance during fabrication. Figures. 1, 8 show some custom enclosures generated in this manner. The accompanying video illustrates various capabilities of our system, along with automatic design.

MODELING ELECTROMECHANICAL DEVICES

Formally, our system represents an electromechanical device (Fig. 4B) and its assembly plan as an ordered tuple $\mathbb{D}$ of parts. Each part d_i has a configuration $\phi_i = (x_i, R_i)$ consisting of 3D position x_i and a 3D rotation R_i, shape S_i at ϕ_i, and an assembly path $P_i(t)$ that defines configuration for the part at every time t during its assembly:

$$\begin{aligned} \mathbb{D} &\equiv (d_1, \ldots, d_n), \\ d_i &\equiv (S_i, \phi_i, P_i(t)). \end{aligned} \tag{1}$$

Notice that this representation stores both the layout of the device – each d_i ends up assembled at configuration ϕ_i – and the assembly plan – start with all components at $P_i(0)$, then in order from 1 to n move each part i along path P_i to its final configuration $P_i(1) = \phi_i$. We therefore call this a spatio-temporal model of device assembly. Note that assembly plans represented in this model always assemble parts into their final configuration one at a time as suggested by the Design For Assembly (DFA) guidelines [8, 39]. DFA guidelines also suggest that multiple rotations and complex paths during assembly can be confusing. Therefore, we only allow piece-wise linear assembly paths P_i, and translational motions along P_i (R_i is kept fixed).

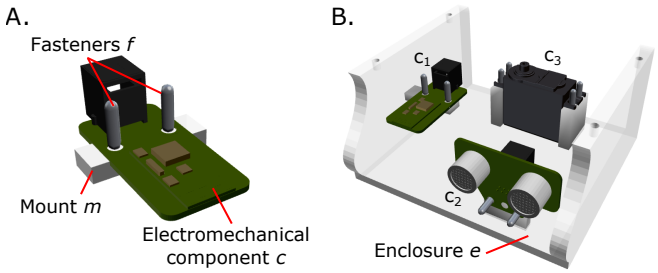

Figure 4. (A) An electromechanical component c is shown with its fasteners f and mount m. The configurations of f and m are defined with respect to c's local coordinate system. (B) A device $\mathbb{D}$ consisting of electromechanical components c_1, c_2, c_3 bounded within an enclosure e. c_i can be supported by a single or multiple mounts and their unique set of fasteners attach to c_i in a specific manner. c_i's mounts get extended to rigidly attach to e's walls.

Validity

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

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

$$\forall i \neq j : \; S_i \cap S_j = \emptyset. \tag{2}$$

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

$$\forall i < j : \; S_i \cap \widetilde{S_j} = \emptyset. \tag{3}$$

where the "swept path" $\widetilde{S_j}$ of a component is the union of its shape S_j over all positions along its assembly path P_j. (Fig. 5B):

$$\widetilde{S_j} = \bigcup_{P_j} S_j. \tag{4}$$

Parts

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

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

$$e \equiv (S_e, \phi_e, 0). \tag{5}$$

In our system, enclosures are always convex polytopes (S_e) with one or more solid faces and one or more lids (which are assumed to be assembled last). While our framework is not

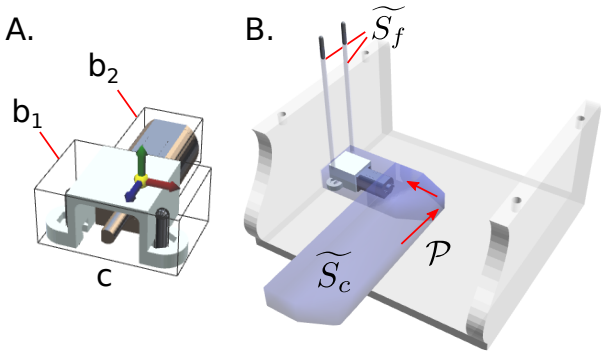

Figure 5. (A) An electromechanical component c is shown with its shape attribute – a set of bounding boxes ($S_c = \bigcup b_i$). Fasteners are encoded with capsule shapes (shown by black mesh). These shapes are defined in the component's local coordinate frame shown at its center with a triad. (B) The swept shape $\widetilde{S_c}$ of c along a piecewise linear path P and that of its fasteners ($\widetilde{S_f}$) along their default paths are shown.

limited to work only with convex objects, we make this assumption owing to the off-the-shelf collision detection that we use within our system. When checking that the layout is valid (eq. 2), collision checks are performed against the exterior of the polytope; while when checking for valid assembly paths (eq. 3) collision checks are performed against only the solid faces of the polytope.

Components are electromechanical components (e.g., microcontrollers, sensors, motors, batteries). Every component c has a shape given by a union of axis-aligned boxes (Fig. 5A), a configuration ϕ_c, and an assembly path P_c:

$$c \equiv (S_c, \phi_c, P_c)\,,$$
$$S_c \equiv \bigcup b_i. \tag{6}$$

Assembly paths P_c are piece-wise linear (as per DFA). In our implementation, P_c are defined by at most two linear segments – P_c^1 and P_c^2 (see Appendix for mathematical definition). The second path segment P_c^2 is only used for components that need to be slid into mounts or through holes in the enclosure, like motors (e.g., Fig. 5B). The direction and length of P_c^2 is set based on a fixed value stored in the component library, so it does not contribute any degrees of freedom to the optimization. For components that do not need to be slid into mounting structures, P_c is single segmented ($P_c^2 = 0$). P_c^1 is parameterized by two spherical angles, α, β, and a radius, r:

$$P_c^1 \equiv r\,[\cos\alpha \cdot \cos\beta, \cos\alpha \cdot \sin\beta, \sin\alpha]^T\,. \tag{7}$$

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

Components have associated mounts and fasteners (Fig. 4A), which are also represented as parts in the device:

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

$$m \equiv (S_m^c, \phi_m^c, 0)\,, \tag{8}$$

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

Fasteners (e.g., rivets, screws) are small parts used to affix components to their mounts. Each fastener f has shape S_f given by a bounding capsule, while its configuration ϕ_f depend on the sockets on c, and can thus be determined using fixed orientation and position offsets from the orientation and position of its associated component c (Fig. 4A).

$$f \equiv \left(S_f, \phi_f^c, P_f^c\right) \tag{9}$$

where superscript ^{c} shows the dependence on c. As with components, the assembly path P_f is set to be sufficiently long so that the fastener starts outside the enclosure. Unlike components, however, the path's direction is fixed based on sockets on c (Fig. 5B).

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

Degrees of Freedom

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

$$L(\mathbb{D}) = \left\{ p_i^l \mid \forall c_i \in \mathbb{D} \right\} \tag{10}$$

Components also have two assembly DOF (the spherical angles α and β used to define the assembly path P). Further, all parts in our model have an assembly order given by their index in the device tuple (eq. 1). Because mounts and the enclosure don't move during assembly, our system always places them first in the tuple; similarly, fasteners always appear immediately after their associated components in the assembly order (see Fig. 6). Thus, the order of the components determines the overall assembly order of the device. Similar to $L(\mathbb{D})$, we succinctly define the device assembly plan $L(\mathbb{D})$ as:

$$A(\mathbb{D}) = \{ i, p_i^a \mid \forall c_i \in \mathbb{D} \}\,. \tag{11}$$

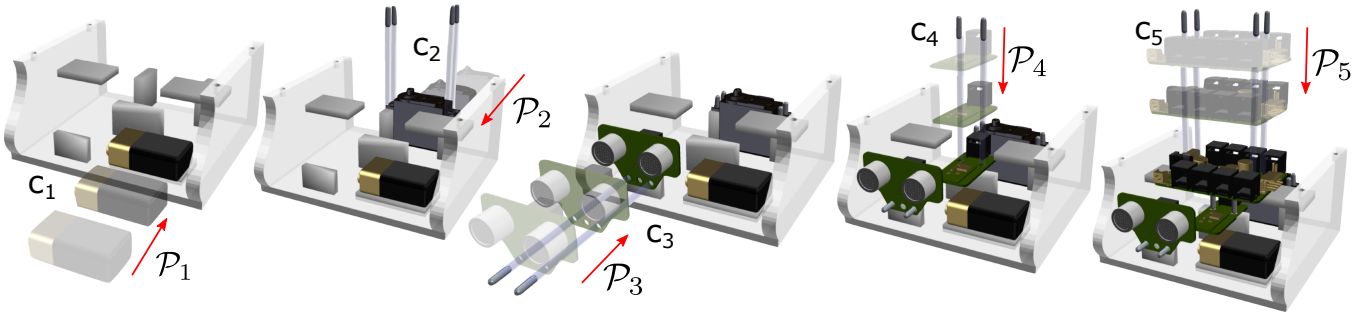

Figure 6. The assembly of a device with 5 components is shown. Each component c_i is assembled one at a time by translating along its assembly path P_i. The assembly process is parameterized by component's assembly order (i in $\mathbb{D}$), and the parameters of P_i. After assembly, c_i is affixed by assembling its fasteners. The fasteners need to be assembled along a specific path. There is a strong interplay between the layout of the components, and their assembly order and paths. This can be seen during the assembly of c_4. Owing to its configuration, c_4 can only be assembled along P_4 before c_5.

where $p_i^a = (\alpha_i, \beta_i)$ are the assembly path parameters, and i is the index of component c_i in the device tuple. (eq. 1)

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

ASSEMBLY-AWARE DEVICE OPTIMIZATION

Our optimization aims to find a device design $\mathbb{D}$ with valid layout (eq. 2), and a valid assembly plan (eq. 3), by simultaneously searching over both layout and assembly degrees of freedom (eq. 10, 11).

This simultaneous optimization stands in contrast to previous work which either optimizes layout for applications of furniture, and virtual world layout design [33, 58, 18] or optimizes assembly given layout for various manufacturing and engineering design applications [23, 55].

Cost function

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

$$\underset{L(\mathbb{D}),A(\mathbb{D})}{\text{minimize}} \quad J(\mathbb{D}),$$
$$J(\mathbb{D}) \equiv J_c + J_b. \tag{12}$$

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

Signed distance measure δ:

A signed distance measure δ between a pair of shape attributes S_i and S_j is computed as the shortest distance between their closest points using Euclidean norm $\|\cdot\|$, when they are not overlapping. Otherwise, we compute δ as the negative of penetration depth (PD), which is a natural extension of Euclidean distance when the elements d_i and d_j are overlapping. PD is defined as the minimum translation distance that one of them undergoes to make the interiors of their shape attributes S_i and S_j disjoint [25] (see Fig. 7A). Detailed mathematical equations are defined in Appendix B. We use a publicly available implementation based on the Expanding Polytope Algorithm (EPA) to compute δ [49, 1].

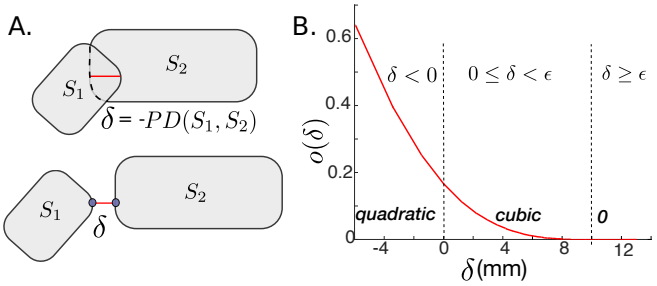

Figure 7. (A) Signed distance measure δ defines distances between overlapping and non-overlapping shapes. When two shapes S_1 and S_2 overlap each other, δ is computed using the minimum translational length that will separate them (called penetration depth (PD)). Otherwise, δ is calculated as the Euclidean distance between the closest points of S_1 and S_2. (B) We use a C^2 continuous cost $o(\delta)$ to penalize overlapping shapes.

Smooth overlap cost $o(\delta)$:

Our signed distance measure δ is not amenable to gradient-based methods. We therefore define a smooth overlap cost $o(\delta)$ to penalize overlapping elements in $\mathbb{D}$. The overlap cost $o(\delta)$ is a function which is quadratic when distance between shapes (δ) is less than zero, and is zero when shapes are sufficiently far from each other (see Fig. 7B, Appendix B). To ensure smoothness, a cubic function is defined over the intermediate distance range $0 \le \delta < \varepsilon$, where $\varepsilon > 0$ determines the minimum separation between the elements in a device. ε allows us to define a safety distance margin between elements, which further aids easy assembly. We set it empirically.

Collision and bounding penalties:

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

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

$$J_c \equiv \sum_{i<j} overlap(S_i, \widetilde{S_j}) \tag{13}$$

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

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

$$J_b \equiv \sum_c overlap(S_e, S_c) \tag{14}$$

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

Note that the enclosure shape used in the collision penalty is the shape without lids (as used in assembly validity), while the enclosure shape used in the bounding penalty is the shape with lids (as used in layout validity).

Numerical Optimization

To optimize the cost function $J(\mathbb{D})$ as defined in eq. (12), we develop an efficient algorithm that combines heuristics inspired by the CAD design community, and powerful optimization strategies.

Heuristics:

We interviewed an expert with 5 years of CAD experience in designing mechanical assemblies to understand the design practices in the community. The expert supported simultaneous reasoning for assembly and layout of components during design. However, the expert highlighted that the expert would approach such concurrent assembly-layout design in an incremental manner. Instead of adding all components in a device at once, the expert would add one component at a time and focus on finding valid layout and assembly for this latest addition, before adding any more components. Similar incremental approaches have also been applied for automatic computer-aided design of VLSI [14, 12], architectural floor plans [15], specifically to deal with high design complexity, and to improve algorithm run times. Inspired by these, we adopt an incremental approach to ensure interactivity during design. Instead of searching for valid configurations (layout and assembly) of all components at once, we incrementally create partial device designs by adding and properly configuring one component at a time to the device. Incrementally adding components to the device optimization ensures that the search space complexity increases gradually, aiding interactivity. Finally, an additional component may be accommodated in a partial design with small reconfigurations of its existing components, if the component makes the best use of available empty space. We therefore reward the use of empty space during our incremental optimization.

Choice of optimization strategy:

Our cost function $J(\mathbb{D})$ is highly multimodal. Further it has a mixture of discrete and continuous optimization variables. Determining the assembly ordering is a combinatorial problem while layout optimization is continuous. Markov Chain Monte Carlo (MCMC) based stochastic optimization methods have been successfully used in the past for combinatorial problems [22, 26]. However, standard MCMC methods tend to get stuck in a single mode while sampling from a multimodal probability distribution. Approaches based on multiple markov chains such as Parallel Tempering have been proposed to overcome this issue [19, 4]. These approaches however, do not offer a mechanism to exploit the availability of gradient information for continuous optimization variables. Recent approaches have shown the benefit of combining gradient based optimization with sampling for both continuous and mixed optimization problems [7, 27]. Using gradient information increases the efficiency of sampling by ensuring less-random walks of the markov chains in the parameter space. Inspired by these approaches, we combine gradient-based methods with Parallel Tempering (PT) for our problem.

Parallel Tempering (PT):

We briefly summarize PT for sake of clarity. Typical MCMC methods perform a memoryless, random walk in the space of parameters θ by simulating a markov chain that generates samples from a function $f(\theta)$. These samples can be generated using a Boltzmann-like probability distribution such as:

$$P(\theta) = \frac{1}{Z} e^{-f(\theta)\beta}, \tag{15}$$

where Z normalizes the distribution, and $\beta \leq 1$ is known as an inverse-temperature constant. β controls the amount of exploration, which reduces as temperature decreases (increasing β values). In PT, independent markov chains are run in parallel on a set of N distributions (such as in eq. 15) with inverse temperatures defined as $0 \leq \beta_N < \beta_{N-1} < \cdots < \beta_1 < \beta_0 = 1$. Periodically, the configurations of these chains are swapped probabilistically. This allows chains at higher temperature that tend to explore more, to pass information about better configurations to exploitative chains at lower temperatures, thereby allowing colder chains to escape local minima. Each chain propagates over time based on the Metropolis-Hastings (MH) update procedure, using easy to sample proposal distributions Q, and the corresponding acceptance probability [34, 20]. The performance of PT is dependent on the proposal distributions Q that are used to update the chains's configurations, and sequence of their inverse-temperatures. For exploring the space of possible layouts and assembly plans effectively, we define proposal distributions that generate chain configurations by perturbing the layout and assembly parameters. These perturbations allow local adjustments around the current values of these parameters as well as create global design changes. We describe our proposal distributions, and procedure for selection of chain temperatures in the Appendices C, D respectively.

Interleaving gradient optimization with PT:

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

$$\theta_t^k = \theta_t^k - \gamma \frac{\partial J(\mathbb{D})}{\partial \theta}\Big|_{\theta=(p^l,p^a)\in\theta_t^k}, \tag{16}$$

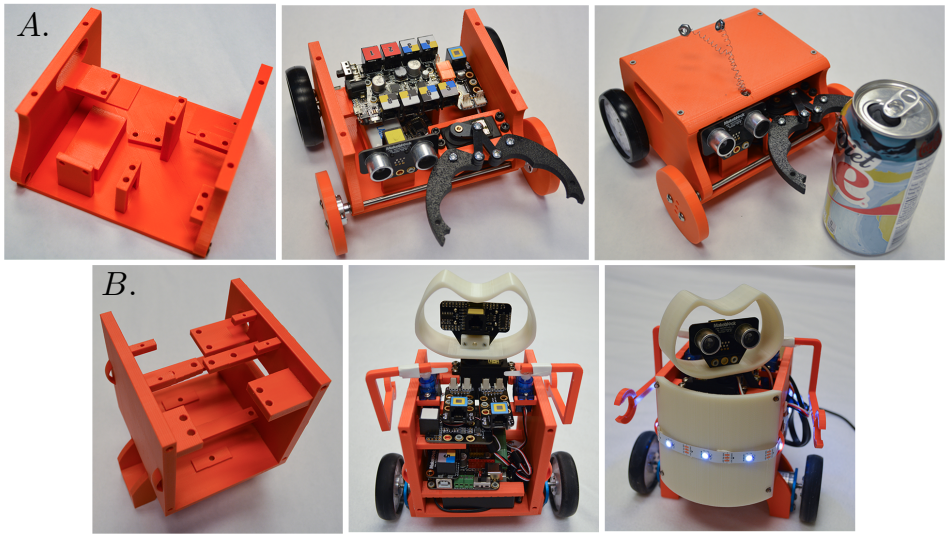

Figure 8. Fabricated devices – (A) Crusher, and (B) Clumsy are shown with their 3D printed enclosures, and their final assembled design. Each enclosure has custom mounts and fastener geometries created by our system for their components, based on the optimized layout. Our video shows these robots in action.

where $\frac{\partial J(\mathbb{D})}{\partial \theta}$ is a numerically computed gradient, and γ is the gradient step-size determined by line search. p^l and p^a are the continuous layout and assembly DOF respectively (eq. 10, 11).

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

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

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

Such incremental design enables the overall optimization process to be much more effective. This is because partial designs have fewer parameters to optimize (smaller design space), and a less constrained volume available for layout. Further, when the optimization is re-run with the next set of components, partially optimized designs result in more favorable initial conditions. Interleaved optimization without our heuristics (incremental design, and use of empty space) lead to much longer optimization times. For instance, the average design time for one of our test devices – Clumsy (Fig. 8) comes out to be 706.7s over 10 runs using only interleaved optimization, instead of 214.14s with incremental interleaved optimization.

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

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

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

Initialize N chains with $\beta_1, \ldots, \beta_N$
for *chain k* **do**
  if $k < \frac{N}{2}$ **then**
    $\theta_0' \leftarrow$ Sample 50 configurations for c_{add}, keeping $\mathbb{D}_{partial}$ fixed
    $\theta_{0*} \leftarrow$ Pick the best configuration out of θ_0'
    Intialize chain k at θ_{0*}
  else
    Intialize chain k at random
  end
end
while *not converged* **do**
  foreach *chain k* **do**
    gradient step for continuous variables (eq. (16))
    Metropolis-Hastings(MH) update of chain k
    if $J(\mathbb{D})$ *at* $\theta_t^k < threshold$ **then return** converged
  end
  swap a random pair of adjacent chains
  update chain's β_k periodically
end

RESULTS

Fabricated examples

We designed and fabricated three devices with very different functionalities to demonstrate the utility of our system – a four wheeled robot called *Crusher*, a two wheeled balancing robot - *Clumsy*, and a smart crib monitoring toy owl –*Chirpy* (shown in Fig. 1, 8). Crusher is a bluetooth controlled recycling robot that can detect and grab soda cans with its gripper arm. Clumsy balances its way through obstacles, and asks for help by waving its hands when needed. Chirpy can detect and soothe a crying baby by flapping its wings and singing a song. Chirpy also alerts the baby's parents by sending a message when the baby starts crying. Each device has a variety of components and unique enclosures. While our system is not restricted to any particular type of kit or modules, we use electromechanical components from Makeblock kits for our examples [29]. Relevant information about fasteners and mounts for each component is pre-processed manually, but one can envision scanning a catalog to gather this information.

To endow devices with a desired functionality, we selected a set of components for each device. The configurations of certain components may be limited within an enclosure owing to the functional and aesthetic requirements of the device. For example, Crusher's motors need to be configured so as to connect to its wheels, while the LEDs for Chirpy should be placed near its eyes. We therefore pre-specify and lock the configurations of such components before generating the assemblable layout of other components using our interleaved optimization. The optimization process maintains the layout of locked components, and optimizes for their assembly while concurrently optimizing the layout and assembly parameters for all other components. For each layout, our system also generated mounting structures and integrated them into the original 3D model of the enclosure using CSG operations. We fabricated our designs using a Stratasys uPrint SE Plus, a filament based 3D printer using ABSP430 plastic as the model material and a dissolvable support material. The fabrication time varied from a few hours to a day, depending on the geometric size of the enclosure. In contrast, the average time to design a device with our system was less than 5 minutes (more details in Fig. 10), and the assembly time of each device was less than 15 minutes. Each enclosure also had single or multiple lids that were printed separately and attached to the enclosure after assembly, through rivets. Overall, it was easy to assemble these devices owing to the DFA guidelines and the assembly animation. However, some parts of devices were bit cumbersome to handle, since we did not model the space needed for hands during the assembly.

While all the devices we fabricated appear to be simple with relatively fewer components, it is worth noting that most designs that casual users make have on average less than 7 components. We base this observation on an informal survey of 3D printable devices that we conducted on two such popular platforms – Instructables and Thingiverse [21, 48]. We found that these designs used on an average 7 components (averaged over 70 designs of robots, IoT devices etc). While these designs may not be from 'casual makers', they reflect what the 'maker' community is interested in building. Further, based on the findings of our user-study (discussed later), devices with 8 components are already quite difficult and time-consuming for non-experts to design manually.

Virtual device examples

Chirpy, Crusher and Clumsy, have cuboidal enclosures. However our system works for any convex-shaped enclosure. Fig. 9 shows devices with a polygonal enclosure, a trapezoidal enclosure with slanted walls, and a bunny-shaped enclosure. Each device contains arbitrary electronic components of varied sizes selected randomly. The trapezoidal enclosure has openings on the top and the bottom, while the polygonal and bunny-shaped enclosures have one opening on the top, and one on the sides. While our current implementation only supports convex-enclosures, our framework will work for concave enclosures as long as we have a way to compute distances from the enclosure. Note that manual layout design for devices such as our trapezoidal and polygonal examples is difficult, owing to the large number of components, and corresponding constraints. This increase in design complexity and cost with increasing number of components prevents non-experts from using more components in their device designs.

USER STUDY

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

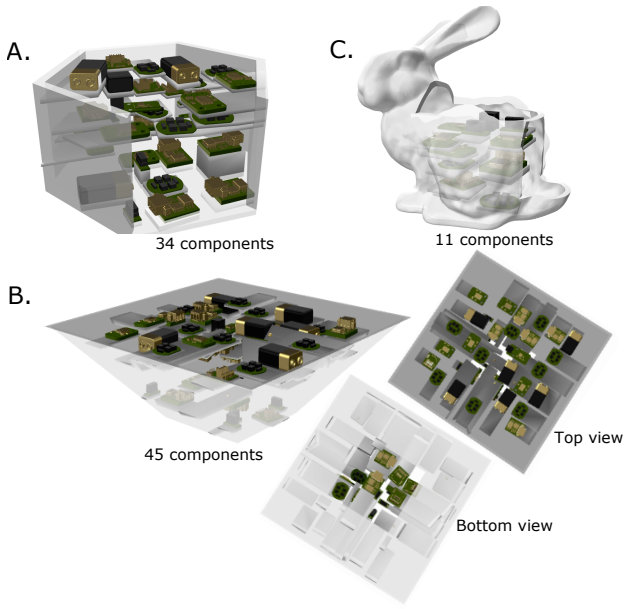

Figure 9. Devices with differently-shaped enclosures and components – (A) a polygonal device, (B) a trapezoidal device, and (C) a bunny-shaped device, each with arbitrary electronic components are shown.

Participants

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

Study structure

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

Qualitative analysis

Participants found our highlighted guidance, automatic mount creation, and animation features highly useful (Fig. 3). For instance, a participant P7 reported that *"assembly animations were very useful, without them layout would be hard."*. Aided by these features majority of participants succeeded in creating device designs (Fig. 10). Inspite of this, all 24 participants rated the design task to be difficult or very difficult, and supported the utility of automatic design mode (see Appendix E for more details on user experience). Participants reported that they would either do physical mockups with iterations (4/24), use pen-paper (2/24), attempt to learn CAD (9/24), or wouldn't know what to do (9/24); for designing such devices if not for our tool. While all participants reasoned well about the space in the device for layout, most of them struggled to make assembly considerations (e.g., P1: *"at first I was only focusing on layout and only then I realized that I should think about [assembly] order."*).

Quantitative analysis

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

A discussion on device complexity:

Even though Clumsy has the same number of components as Crusher, there is a significant difference in their complexity, as evident in the design time and success rate statistics. We

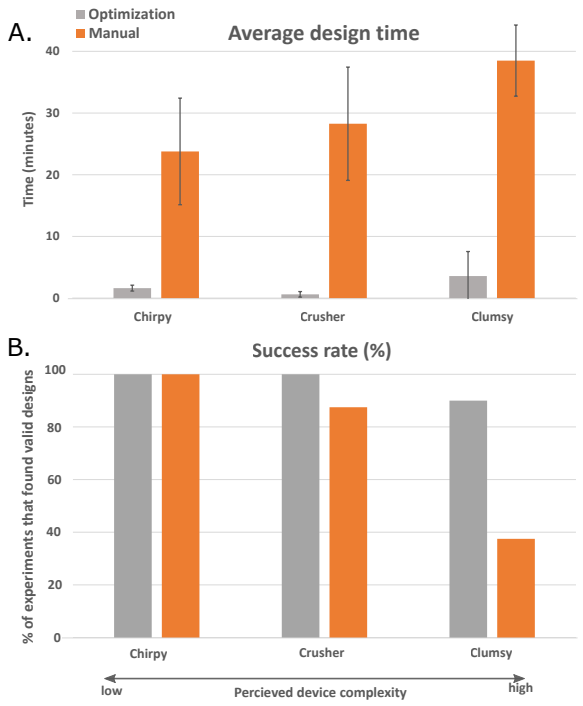

Figure 10. (A) The average design time in minutes, and (B) the success rate for each of our 3 devices are shown. The orange bars represent these statistics averaged over 8 participants per device during our study, while the gray bars correspond to those averaged over 10 runs of incremental interleaved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

therefore attempt to quantify the approximate complexity of these devices using a set of features (table 1). In particular, we use – a) the number of parameters, and b) the fill ratio which is defined as the relative volume occupied by all components and mounting structures. The fill ratio is computed for each valid design. Note that the number of parameters and the fill ratio capture different aspects of complexity.

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

Table 1. The number of components ($\mid C \mid$), and fasteners ($\mid F \mid$), fill ratio, and number of discrete and continuous parameters to optimize (# p) indicate approximate complexity of each design.

While the features in table 1 provide an approximate idea of device complexity, it is hard to precisely quantify the difficulty of finding an assemblable layouts. This is because the complexity depends upon not just the number of components, volume filled, and parameter count, but is also a function of many other factors. In particular, the shape of the enclosure dictates possible layouts and assembly paths for the components. However, the enclosure shape is not characterized by its volume. Out of two enclosures with the same volume, the enclosure with a shape that provides more surface area to mount the components may be more amenable for layout. Further, when a component is added to an enclosure, the component and its mounts partition the space available for other components in a non-trivial manner. Fasteners, and configurations of locked components such as LED further shape the remaining available space. The role of these factors becomes apparent in the design of Clumsy. Even though Clumsy has only 8 components, 2 of its components – a battery pack, and a controller board are very large and can be arranged in only certain configurations so as to fit within the enclosure. If their configurations are badly initialized, the smaller components may block them from achieving these valid configurations (as reflected in the less than 100% success rate of our optimization (gray) in Fig. 10).

LIMITATIONS AND FUTURE WORK

We introduced a computational design system that enables users with no CAD experience to design fabricable and assemblable 3D printed devices with embedded electromechanical components. With the help of a parameterized model that captures the spatio-temporal aspects of the assembly process, and an efficient concurrent optimization scheme, our system allows users to design complex devices within minutes. Our spatio-temporal model is purely geometric and general. Further, our system encodes Design for Assembly (DFA) principles that ensures the resultant designs to be easily assemblable. We evaluated our system by creating a variety of designs, and with a user-study. Our system could find valid designs for devices in less than 5 minutes, while the users took longer than 30 minutes on average with lower success rates. The lack of CAD experience and resulting design challenges faced by our participants further confirmed the usefulness of our system.

However, our system is currently limited to devices without transmission or moving parts. Extending our framework to account for moving parts that take a range of configurations once assembled will enable the design of more diverse devices. Including other higher level design requirements (such as a desired center of mass for Clumsy) while optimizing for component layout may further increase the space of designs. We also do not account for strengths of mounts generated. Our system's mounts were sufficient for the majority of electronic components that are lightweight. However, including structural optimization of mounts within the system will be important for making robust designs in the future. Exploring design possibilities with flexible mounts, and supporting novices through the use of device mockups in virtual reality are other interesting directions for future work.

Finally, we found that it is challenging to quantify design complexity of such devices. As a result, guaranteeing a solution for any arbitrary device is non-trivial. Further, in our preliminary scalability experiments (see Appendix F), we found that our optimization needed much longer time to find valid designs for devices with more than 15 components, or higher than 50% fill ratio. The number of components, and fill ratio in devices made by our target audience of makers and artist is well below these bounds. However, more research is necessary to enable our system to support experts, or other target audiences. To this end, a promising approach entails putting user-in-the loop during incremental optimization, as well as exploring better ways to incorporate user intuition during design.

REFERENCES

1. 2015. *Bullet Physics Library*. http://bulletphysics.org/.
2. 2017. *Sketch-up*. https://www.sketchup.com/.
3. Daniel Ashbrook, Shitao Stan Guo, and Alan Lambie. 2016. Towards Augmented Fabrication: Combining Fabricated and Existing Objects. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 1510–1518.
4. Yves F Atchadé, Gareth O Roberts, and Jeffrey S Rosenthal. 2011. Towards optimal scaling of Metropolis-coupled Markov chain Monte Carlo. *Statistics and Computing* 21, 4 (2011), 555–568.
5. Autodesk 2014. *Autodesk 123D Design*. Autodesk. http://www.123dapp.com/design.
6. Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. 2014. Spin-it: optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 96.
7. Gaurav Bharaj, David IW Levin, James Tompkin, Yun Fei, Hanspeter Pfister, Wojciech Matusik, and Changxi Zheng. 2015. Computational design of metallophone contact sounds. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 223.
8. Geoffrey Boothroyd. 2005. *Assembly Automation and Product Design, Second Edition (Manufacturing Engineering and Materials Processing)*. CRC Press, Inc., Boca Raton, FL, USA.
9. George Casella and Edward I George. 1992. Explaining the Gibbs sampler. *The American Statistician* 46, 3 (1992), 167–174.
10. Andrea Censi. 2017. A Class of Co-Design problems with cyclic constraints and their solution. *IEEE Robotics and Automation Letters* 2, 1 (2017), 96–103.
11. Duygu Ceylan, Wilmot Li, Niloy J Mitra, Maneesh Agrawala, and Mark Pauly. 2013. Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 186.
12. Jason Cong and Majid Sarrafzadeh. 2000. Incremental physical design. In *Proceedings of the 2000 international symposium on Physical design*. ACM, 84–92.
13. Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 83.
14. Olivier Coudert, Jason Cong, Sharad Malik, and Majid Sarrafzadeh. 2000. Incremental cad. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*. IEEE Press, 236–244.
15. Jim Crenshaw, Majid Sarrafzadeh, Prithviraj Banerjee, and Pradeep Prabhakaran. 1999. An incremental floorplanner. In *VLSI, 1999. Proceedings. Ninth Great Lakes Symposium on*. IEEE, 248–251.
16. Ruta Desai, Ye Yuan, and Stelian Coros. 2017. Computational abstractions for interactive design of robotic devices. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 1196–1203.
17. Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. 2016. Computational multicopter design. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 227.
18. Ran Gal, Lior Shapira, Eyal Ofek, and Pushmeet Kohli. 2014. FLARE: Fast layout for augmented reality applications. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*. IEEE, 207–212.
19. Charles J Geyer. 1991. Markov chain Monte Carlo maximum likelihood. (1991).
20. W Keith Hastings. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57, 1 (1970), 97–109.
21. Instructables. 2017. *Instructables - How to make anything*. https://www.instructables.com/.
22. Mark Jerrum and Alistair Sinclair. 1996. The Markov chain Monte Carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems* (1996), 482–520.
23. Pablo Jiménez. 2013. Survey on assembly sequencing: a combinatorial and geometrical perspective. *Journal of Intelligent Manufacturing* 24, 2 (2013), 235–250.
24. Michael D Jones, Kevin Seppi, and Dan R Olsen. 2016. What you sculpt is what you get: Modeling physical interactive devices with clay and 3d printed widgets. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 876–886.
25. Young J Kim, Miguel A Otaduy, Ming C Lin, and Dinesh Manocha. 2002. Fast penetration depth computation for physically-based animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM, 23–31.
26. Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, and others. 1983. Optimization by simmulated annealing. *science* 4598 (1983), 671–680.
27. Dingzeyu Li, David IW Levin, Wojciech Matusik, Changxi Zheng, Timothy R Langlois, Changxi Zheng, Doug L James, Gaurav Bharaj, David IW Levin, James Tompkin, and others. 2016. Acoustic Voxels: Computational Optimization of Modular Acoustic Filters. *ACM Transactions on Graphics* 33 (2016), 2.
28. Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. 2014. Build-to-last: strength to weight 3D printed objects. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 97.
29. Makeblock. 2015. *Makeblock electronic modules*. Available at http://www.makeblock.com/electronic-robot-kit-series-STEM.

30. Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. 2015. Interactive design of 3D-printable robotic creatures. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 216.

31. Ankur M Mehta, Joseph DelPreto, Kai Weng Wong, Scott Hamill, Hadas Kress-Gazit, and Daniela Rus. 2018. Robot creation from functional specifications. In *Robotics Research*. Springer, 631–648.

32. Paul Merrell, Eric Schkufza, and Vladlen Koltun. 2010. Computer-generated residential building layouts. In *ACM Transactions on Graphics (TOG)*, Vol. 29. ACM, 181.

33. Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. 2011. Interactive furniture layout using interior design guidelines. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 87.

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

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

36. Radford M Neal and others. 2011. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo* 2 (2011), 113–162.

37. Jorge Nocedal and Stephen J Wright. 2006. Numerical optimization. (2006).

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

39. Gerhard Pahl and Wolfgang Beitz. 2013. *Engineering design: a systematic approach*. Springer Science & Business Media.

40. Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. Retrofab: A design tool for retrofitting physical interfaces using actuators, sensors and 3d printing. *Proc. of SIGCHI. ACM* (2016).

41. Rohit Ramesh, Richard Lin, Antonio Iannopollo, Alberto Sangiovanni-Vincentelli, Björn Hartmann, and Prabal Dutta. 2017. Turning coders into s: the promise of embedded design generation. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*. ACM, 4.

42. Gareth O Roberts, Andrew Gelman, Walter R Gilks, and others. 1997. Weak convergence and optimal scaling of random walk Metropolis algorithms. *The annals of applied probability* 7, 1 (1997), 110–120.

43. Bruce Romney, Cyprien Godard, Michael Goldwasser, G Ramkumar, and others. 1995. An efficient system for geometric assembly sequence generation and evaluation. *Computers in Engineering* (1995), 699–712.

44. Valkyrie Savage, Sean Follmer, Jingyi Li, and Björn Hartmann. 2015. Makers' Marks: Physical markup for designing and fabricating functional objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, 103–108.

45. Stefan Schneegass, Alireza Sahami Shirazi, Tanja Döring, David Schmid, and Albrecht Schmidt. 2014. NatCut: an interactive tangible editor for physical object fabrication. In *Proceedings of the extended abstracts of the 32nd annual ACM conference on Human factors in computing systems*. ACM, 1441–1446.

46. A Schug, T Herges, and W Wenzel. 2004. All-atom folding of the three-helix HIV accessory protein with an adaptive parallel tempering method. *Proteins: Structure, Function, and Bioinformatics* 57, 4 (2004), 792–798.

47. Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Měch. 2012. Stress relief: improving structural strength of 3D printable objects. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 48.

48. Thingiverse. 2017. *Thingiverse - Digital Designs for Physical Objects*. `https://www.thingiverse.com/`.

49. Gino Van Den Bergen. 2001. Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference*, Vol. 170.

50. Marynel Vázquez, Eric Brockmeyer, Ruta Desai, Chris Harrison, and Scott E Hudson. 2015. 3d printing pneumatic device controls with variable activation force capabilities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1295–1304.

51. Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. 2012. . NET gadgeteer: a platform for custom devices. In *Pervasive Computing*. Springer, 216–233.

52. Voxel8. 2015. *Voxel8: 3D Electronics Printing*. `http://www.voxel8.co`.

53. Christian Weichel, Manfred Lau, and Hans Gellersen. 2013. Enclosed: a component-centric interface for designing prototype enclosures. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. ACM, 215–218.

54. Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. 2012. Printed optics: 3D printing of embedded optical elements for interactive devices. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 589–598.

55. Randall H Wilson. 1992. *On Geometric Assembly Planning.* Technical Report. DTIC Document.

56. Jan D Wolter. 1991. On the automatic generation of assembly plans. In *Computer-Aided Mechanical Assembly Planning*. Springer, 263–288.

57. Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D Goodman, and Pat Hanrahan. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 56.

58. Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30, no. 4, July 2011, article no. 86* (2011).

APPENDIX

ASSEMBLY PATH REPRESENTATION

The linear piece-wise assembly paths P_i for components and fasteners can be precisely represented as functions of time. The assembly path function for components consisting of two piecewise linear steps $\vec{x_1}$ and $\vec{x_2}$ and a final position x can be written as:

$$P_c(t) \equiv \begin{cases} x - \vec{x_2} - (1-2t)\vec{x_1} & \text{if } t < 0.5 \\ x - (2-2t)\vec{x_2} & \text{otherwise} \end{cases} \tag{17}$$

The assembly paths for fasteners are single segmented and depend upon their associated component's configuration and corresponding socket positions. Such an assembly path with step $\vec{x_1}$ is equivalent to:

$$P_f(t) \equiv x_c + R_c\left(x - (1-t)(\vec{x_1})\right), \tag{18}$$

where x_c and R_c define configurations of the fastener's associated component.

SIGNED DISTANCE MEASURE AND SMOOTH OVERLAP COST

Mathematically, the signed distance measure δ between a pair of shape attributes S_i and S_j (as in Fig. 7) is defined as:

$$\delta(S_i, S_j) = \begin{cases} -PD(S_i, S_j), & \text{if } S_i \cap S_j \neq \emptyset \\ \Delta, & \text{otherwise} \end{cases} \tag{19}$$

where S_i and S_j are shape attributes. The shortest distance Δ is defined as $\min(\|x_i - x_j\|, \mid x_i \in S_i, x_j \in S_j)$, where $\|\cdot\|$ denotes Euclidean norm. Likewise, penetration depth $PD(S_i, S_j) = \min(\Delta \mid interior(S_i + \Delta) \cap S_j = \emptyset)$.

$$o(\delta) = \begin{cases} a_1\delta^2 + b_1\delta + c_1, & \text{if } \delta < 0 \\ a_2\delta^3 + b_2\delta^2 + b_1\delta + c_1, & \text{if } 0 \leq \delta < \varepsilon \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

where a_1 corresponds to the stiffness of the quadratic cost. $b_1 = -\frac{a_1\varepsilon}{2}$, $c_1 = \frac{a_1\varepsilon^2}{6}$, $a_2 = -\frac{a_1}{6\varepsilon}$, and $b_2 = \frac{a_1}{2}$ are constant weights that are determined such that the resultant overlap cost function is C^2 continuous. a_1 and ε are empirically set.

METROPOLIS-HASTINGS UPDATE AND PROPOSAL DISTRIBUTION Q

Generating markov chains corresponding to a distribution P (as in eq. 15) is generally intractable due to the required computation of the normalization constant Z in eq. 15. Instead, Metropolis-Hastings approach allows exploring distributions without computing Z, in the following manner. Starting with a random configuration in the parameter space θ_1, a sample θ' is proposed at each time step, from an easy to sample proposal distribution $Q(\theta \mid \theta_t)$. θ' is accepted in the chain with a probability:

$$\alpha(\theta') = \min\left(1, \frac{P(\theta')}{P(\theta_t)} \frac{Q(\theta_t \mid \theta')}{Q(\theta' \mid \theta_t)}\right), \tag{21}$$

where α is called the MH acceptance probability. If θ' is accepted, $\theta_{t+1} = \theta'$. We refer to this as the MH-update step (line 14 in Algorithm 1). The proposal distributions Q that are used to propose samples for MH-update of each chain typically use perturbation mechanisms. These mechanism underlying proposal distributions for layout and assembly parameters are described below:

Layout perturbation: The layout parameters p_i^l of components $c_i \in \mathbb{D}$ (as defined in eq. (1)) consisting of 3D position x_i and orientation R_i of c are perturbed in 4 ways:

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

The first two perturbations are 'local', while the last two perturbations allow the markov chains to jump to different parts of the parameter space. We employ rejection sampling from $\mathcal{N}(0, \sigma_x)$ to ensure that the resultant component configuration is within the bounds of the enclosure. σ_x is auto-tuned to achieve 23% acceptance rate during MH-updates of each chain (eq. (21)). This is based on the theoretical evidence that suggests this rate to be a good general setting [42].

Assembly plan perturbation: Based on our early experiments, we develop a set of heuristics to perturb the assembly parameters p_i^a corresponding to the assembly path, and the assembly order i (defined in eq. (11)).

- Instead of sampling the assembly path parameters p_i^a according to a Gaussian distribution, we uniformly sample these parameters for each component $c_i \in \mathbb{D}$ around a set of main directions that correspond to removable panels (lids) of the enclosure. Such biased sampling of assembly paths allows us to filter out paths that are blocked by enclosure walls, and speed up computation considerably.

- To perturb assembly order, we adopt a greedy strategy that allows for occasional exploration. We swap assembly order i (index in device tuple in eq. 1) of two randomly selected components with a small probability, or generate a heuristic ordering otherwise. This strategy is based on the observation that out of $n!$ assembly orderings for a set of n components, many orderings have the same outcome. For example, when a group of small components is blocked by a larger component, swapping the order between components in this group is counter-productive. Therefore, instead of resorting to un-informed sampling in the assembly order space, our heuristic ordering is generated by considering the layout parameters. It is decided based on the distance of components from the main opening of the enclosure, with the farthest component getting assembled first. This approach of sampling a parameter given other parameters is similar to Gibbs sampling [9].

Note that our proposal distributions are symmetric ($Q(\theta_t \mid \theta') = Q(\theta' \mid \theta_t)$), which further simplifies eq. 21.

TUNING CHAIN TEMPERATURES β

Inspired by [46], we adapt the temperature of chains during sampling to achieve this swap rate of 23% amongst each adjacent pair of chains. For this adaptation, we first initialize inverse-temperatures with a geometric temperature sequence: $\beta_{j+i} = \rho\beta_j$, where ρ is a constant. ρ can be easily determined given the number of chains N and maximum chain temperature β_N. We use $N = 10$, and $\beta_N = 0.001$ for all our experiments (empirically determined).

USER DESIGN EXPERIENCE

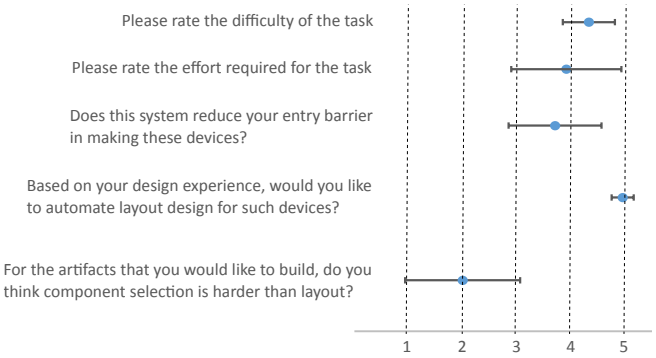

Figure 11. Users feedback about the design task, and our system are highlighted by their responses to our survey. 1 - strongly disagree/very low, 3 - neutral, 5 - strongly agree/very high. Error bars indicate standard deviation.

Fig. 11 shows the responses to our survey on Likert scale.

ADDITIONAL VALIDATION EXPERIMENTS

We perform two additional experiments for validating our approach. First, we validate the interleaved optimization approach for our problem, by comparing with other standard sampling and gradient based methods. Next, we describe a benchmarking experiment that shows the scalability of our framework with increasing number of components, and fill ratio.

Comparison with other optimization approaches

In order to validate the advantage of interleaving gradient optimization with Parallel Tempering (PT), we compare against other possible variants – 'PT only', 'PT followed by gradient optimization', and 'gradient optimization only'. We use the BFGS quasi-Newton method for gradient optimization in the last two test conditions [37]. 'PT only' has been successfully used for layout problems in the past such as for furniture layout [33], while BFGS is widely used for continuous optimization problems [37]. 'PT followed by gradient optimization' combines stochastic and gradient based optimization in a naive manner. Table 2 shows the comparison of our interleaved optimization against these variants for the design of Crusher averaged over 10 runs. Considering that the interleaved optimization for Crusher design found a valid solution in 154s on average, we ran PT for 500s or until the cost threshold of collision-free design was reached, for 'PT only', and 'PT followed by gradient optimization' test conditions. This was followed by 100s of gradient optimization for the latter. In order to replicate the random initializations and N parallel chains of PT, we execute $N = 10$ parallel gradient optimizations, each starting with a random configuration for the last test condition. Since the gradient optimization cannot be used to find a discrete assembly ordering, we set the ordering based on the initial configuration in a heuristic manner (as described in Appendix C).

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

Table 2. Comparing optimization techniques based on how often they find valid designs, and the time they take to find them.

The success rate of this experiment is an indicator of the probability of finding an assemblable layout given a fixed ordering. It reaffirms the need to search for an assembly ordering and layout concurrently. The interleaved optimization strategy finds valid designs in lesser time with higher success rate compared to other methods. Combining interleaved optimization with incremental design reduces the design time even further (Fig. 10). 'PT only' does not manage to find acceptable designs in 500s for any runs.

Scalability experiments

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

For each experiment, a virtual device is created with fill ratio m, and n cuboidal components of arbitrary sizes. To reduce the affect of enclosure and component shapes on the device complexity, we use simple cuboidal enclosure and cuboidal components, for all our experiments (Fig. 12(A)). Further, we assume that the enclosure's size and thereby volume can be

appropriately scaled as needed. Scaling the enclosure in this manner allows us to easily control a device's fill ratio for our test scenario.

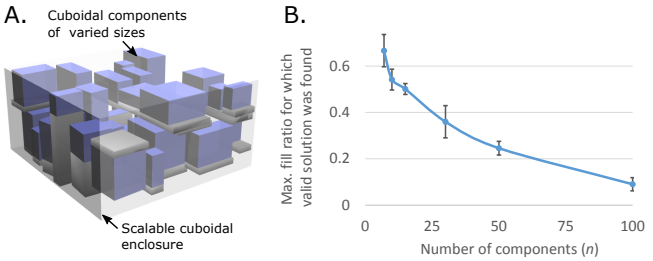

Figure 12. We perform scalability experiments that measure the performance of the algorithm as the number of components, and fill ratio increases. (A) shows an example virtual device with arbitrary sized cuboidal components that we create for these experiments. A scalable cuboidal enclosure is used to easily increase the fill ratio during the experiments. (B) gives an intuition about the maximum fill ratio of devices with n components for which the algorithm was able to find a solution in the allotted time.

In order to test the scalability of our framework with increasing fill ratio, we keep the number of components n in a device constant as we change the device's fill ratio m. Similarly, to test how the framework scales with increasing number of components in a device, we keep the fill ratio fixed as we increase the number of components in the device. During the course of first experiment, we gradually scale down the enclosure volume to increase the fill ratio for a device with n components. We run incremental design optimization to find valid assemblable layout designs, for an hour for each fill ratio. We continue increasing the fill ratio till the optimization fails to find a valid device design in an hour, and record the maximum fill ratio for which the optimization succeeded. For testing scalability with number of components, we repeat the above experiment for different values of n (number of components). Fig. 12(B) shows a plot of maximum fill ratio for which the optimization found a valid design in the allotted time vs. number of components for our virtual cuboidal device, calculated over set of 3 experiments. As the number of components or fill ratio increases, the optimization needs more time, and hence finds fewer valid designs in the allotted time.