A Design Environment for the Rapid Specification and Fabrication of Printable Robots

Ankur Mehta*, Nicola Bezzo†, Peter Gebhard†, Byoungkwon An*,
Vijay Kumar †, Insup Lee†, and Daniela Rus*

*Massachusetts Institute of Technology,
†University of Pennsylvania

Abstract. In this work, we have developed a design environment to allow casual users to quickly and easily create custom robots. A drag-and-drop graphical interface allows users to intuitively assemble electromechanical systems from a library of predesigned parametrized components. A script-based infrastructure encapsulates and automatically composes mechanical, electrical, and software subsystems based on the user input. The generated design can be passed through output plugins to produce fabrication drawings for a range of rapid manufacturing processes, along with the necessary firmware and software to control the device. From an intuitive description of the desired specification, this system generates ready-to-use printable robots on demand.

Keywords: Design co-generation, Personal robots, Printable robots

1 Introduction

Creating a robotic system generally requires broad engineering expertise to successfully manage the interplay between constituent electromechanical subsystems. For an average user to be able to create their own robots for personal use, a system is needed to abstract away technical details in favor of easy to understand functionality specifications. To parallel a typical solution methodology in which a problem is broken into atomic tasks, we have developed a design environment in which robots are assembled by simply connecting self-contained components. These modular building blocks encapsulate mechanical, electrical, and software subsystems without relying on a specific manufacturing process. A full robot is then defined by the component hierarchy required to achieve each necessary behavior, and can be directly fabricated from design files automatically co-generated after passing the specification through a suitable output plugin.

This definition is specified by the user in an intuitive and simplified high-level graphical programming language. Blocks describing distinct parts of a robot are graphically connected together on a workspace to describe the robot design. From this, Python code is automatically generated which, when executed, creates the hardware design description of printable robots. This graphical frontend provides intuitive user interfacing while providing versatile cus-

tomizability, while the backend is compatible with a variety of 2D and 3D manufacturing processes suitable for automated home fabrication. The overall system, outlined in figure 1 takes as input a schematic outlining a high-level breakdown of the required components, and outputs a fabricated ready-to-use robot, with minimal required user intervention. This system paves the way towards realizing smart programmable cyber-physical systems on demand towards a future of pervasive personal robots.

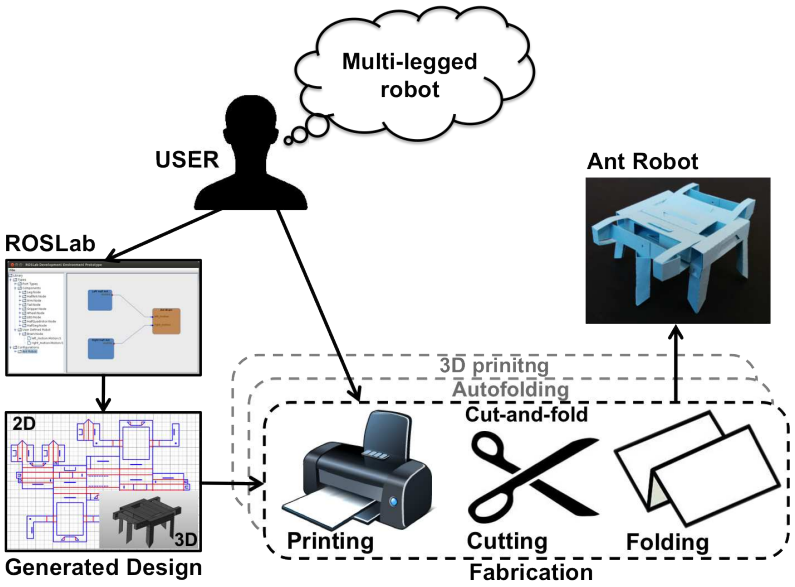


Fig. 1. Workflow diagram showing the steps necessary for a user to design, create, and fabricate a robot

2 Related Work

2.1 Fabrication

There are a variety of fabrication methods to create printable devices. Arbitrary 3D structures are generally achievable by additive manufacturing using 3D printers; advances in printer technology have made desktop printers available to the general public. However, while complex solid geometries are easily manufactured with 3D printing, achieving the required compliance and mobility necessary for general robotic systems is nevertheless difficult to achieve using most common techniques [1]. Limited workarounds do exist [2, 3]; these often lack robustness or reliability, though current technology has been improving.

Alternatively, mechanical structures can be realized by patterning then folding 2D sheets to define the shell of the desired geometry. A variety of substrates are possible, including cardboard laminates [4], single layer plastic film [5], or more exotic materials [6, 7]. These designs can be manually folded by hand, folded by embedded or external active stimuli, or passively folded by controlled environmental conditions [8, 9].

2.2 Design

The fabrication processes listed in the previous section require a multitude of computer-aided design (CAD) tools to specify mechanical structures or electromechanical assemblies. Though some automated design tools have been developed, especially to translate 3D geometries into 2D unfoldings [10, 11], these are often limited in scope, resulting in custom designs needing to be manually drawn by experienced designers. Instead, the work presented in this paper builds off of the system presented in [12, 13], wherein mechanical designs for a particular cut-and-fold fabrication process are abstracted into code objects.

The use of graphical languages is common in the engineering and academic community. Simulink [14] and LabView [15] are two well known examples widely used for general purpose engineering computing. Authors in [16] use a formal graphical language to program medical operation between different medical devices. Standing more from a robotic perspective, only a few graphical programming environments are available and usually limited and constrained to specific platforms [17, 18]. To overcome this limitation, in [19] a high level programming language called ROSLab was proposed to generate C++/ Python code for general robotic applications involving different type of platforms such as aerial, wheeled, and multi-legged robots.

3 Technical Approach

3.1 Scripted Hardware Programming

The system presented in [12, 13] provides a scripted programming language to specify designs for printable robots realized using a cut-and-fold fabrication process. To enable greater versatility, that system was overhauled in this work to abstract designs into a process-agnostic representation of the component hierarchy.

In this system, parametrized components are defined by code objects, with scripted functions representing physical manipulations and design steps. Basic building blocks can define mechanical, electrical, or software elements, while composite blocks can be integrated across subsystems. Parameters can be used to customize variable geometric measurements, alternate electric components, or other design-time configurable quantities.

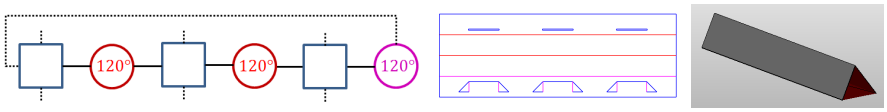
The code objects expose an interface allowing them to be hierarchically composed: component modules specify predetermined connections along which

other components can be attached. Attaching modules via their connections forms a new higher-order module, establishing constraints on their free parameters, merging their mechanical structures, and wiring together their electrical components.

Complex designs, from electromechanical mechanisms up to full robots, can then be represented as simple software scripts implementing the above steps. Executing a script co-generates fabricable design files for the complete device, including mechanical drawings to be sent to a fabrication tool, electrical component requirements and wiring information, and firmware and software libraries and application code.

The composition of building blocks into higher order components is carried out using an internal graph-based representation to generate structurally specified robot designs. This graph specifies the connections and parameter constraints between constituent modules. To generate fabricable drawings, then, an interpreter plugin realizes the geometries along the design graph into a format required by the manufacturing process of choice. Plugins have been written for a number of fabrication methods as described in section 4.2 below.

Mechanical geometries are stored using a face-edge graph that can be resolved to both 2D and 3D shapes as required by specific fabrication processes. A basic example of this is shown in by the beam in figure 2, generated from the code in listing 1.1. The blue squares in the graph represent the rectangular faces of the beam, connected to each other along folded edges represented by red circles. The unconnected dashed lines represent connections along which future components can be attached. A cut-and-fold pattern can be generated from the face graph, requiring the dotted edge to be replaced by a tab-and-slot connector. A 3D solid model can also be generated to display the structure resulting from folding the 2D pattern, or to directly generate a 3D object via 3D printing.



(a) Face-edge graph representation of a beam geometry (b) Generated drawing to be sent to a 2D cutter (c) Generated 3D solid model

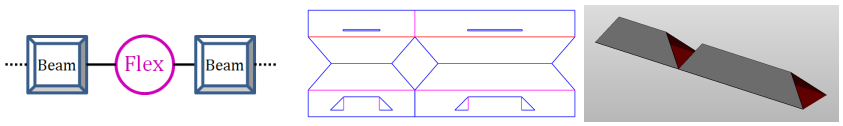
Fig. 2. Outputs generated from the code in listing 1.1

```
1 import Beam
2 b = Beam.Beam()
3 b.setParameter("length", 100)
4 b.setParameter("beamwidth", 10)
5 b.setParameter("shape", 3)
```

Listing 1.1. Scripted design of a mechanical beam

A simple composite structure is demonstrated in figure 3 from the code in listing 1.2. As above, constituent objects are instantiated and their parameters

are set. However, these building blocks are attached along exposed connections in a new higher-order component. The connection type is set to be a flexible joint to allow for compliant motion, and geometric constraints are imposed on objects' parameters.



(a) Component-connection graph representation of finger design hierarchy (b) Generated drawing to be sent to a 2D cutter (c) Generated 3D solid model

Fig. 3. Outputs generated from the code in listing 1.2

```
from api.Component import Component
from api.Edge import Flex
from Beam import Beam

finger = Component()

finger.addSubComponent("beam1", Beam)
finger.addSubComponent("beam2", Beam)

finger.setSubParameter("beam1", "length", 60)
finger.setSubParameter("beam1", "beamwidth", 10)
finger.setSubParameter("beam1", "shape", 3)
finger.setSubParameter("beam1", "angle", 45)

finger.setSubParameter("beam2", "length", 40)
finger.setSubParameter("beam2", "angle", 45)

finger.add("beam1")
finger.connect(("beam1", "topedge"),
               ("beam2", "botedge"),
               Flex())
```

Listing 1.2. Scripted design of a composite finger

As described in [13], electrical components, including sensors, actuators, and processors, can also be encapsulated in the same component framework of the mechanical designs described above. Purely electrical devices can be combined with mechanical structures to form integrated electromechanical mechanisms. Software drivers and UI elements can similarly be included within a component to define a fully self-contained robotic subsystem. The connections for such integrated components algorithmically combine electrical wiring and software blocks as well as mechanical geometries, preserving the modular design abstraction across robotic subsystems.

When compiled, the design scripts produce device specifications and wiring diagrams for the complete electronic subsystem, as well as integrated software packages to operate the designed electromechanical system. In this way, complete designs are automatically co-generated for the design hierarchy, thus en-

abling a user to design a print-and-play robot by scripting the composition of elements.

3.2 ROSLab for Hardware Design Generation

To further simplify the design flow for casual users, a graphical programming tool was adapted to generate the hardware specification scripts. ROSLab [19] provides an intuitive and simplified high-level development environment that builds software through a drag-and-drop interface. Code components are represented as blocks which can be imported from a library into a workspace, then connected along available interfaces to generate new designs.

In this work, we extend ROSLab to provide a design environment for creating printable robots with the scripted infrastructure described above. A predesigned collection of Python scripts representing parametrized robotic building blocks are available to a user as a component library in the ROSLab environment. Desired blocks can be dragged into a workspace, and parameters can be set by the user based on target specifications. Exposed interfaces on each robot component are represented by ports on the ROSLab block; these ports can be wired together to specify electromechanical connections. Compositions can themselves be saved as components in the library to be used in future higher order designs. In this way, a full robot can be hierarchically composed from its constituent blocks.

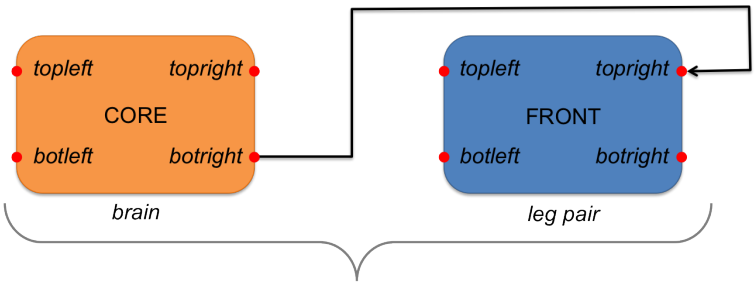
Code Generation and User Interface (UI) The programming workflow in ROSLab follows the intuitive logic of how one would construct a robot. Generally speaking, a robot is made of a main body part, which we call *brain* because it contains the processing unit (e.g., microcontroller, microprocessor, etc.), locomotion components (e.g., legs, wheels, propellers, etc.), and other extremities that hold sensors and actuators (e.g., gripper). A typical ROSLab program for fabrication starts by defining the brain node followed by the desired locomotion technique. For instance, a user that is interested in designing an ant-like robot would need to specify the <*brain*> component and two <*motion*> type <*leg-pair*> blocks. Similarly for a two wheeled robot (e.g., a Segway), the <*brain*> component will be connected to two <*wheel*> modules of type <*motion*>. The user will then place blocks in the ROSLab workspace to symbolize the brain and leg-pairs (or wheels for the Segway) and will connect these blocks together. Once the design is finalized, it will be automatically generated by ROSLab.

ROSLab is written in Java and is designed to follow a template logic. The templates are characterized by fields which are filled by specific code snippets according to how a component is connected to another component. Prior to use by a casual user, a ROSLab developer must first carefully construct the templates by finding common code primitives among the possible output files. Each component is described by (*i*) a set of ports which for a printable robot consist of foldable tabs; and (*ii*) a set of parameters which define the dimension and interface with the electronic hardware. These parameters can be easily

edited within the ROSLab UI, thus giving the user the ability to create unlimited designs for the robot under development.

The code generation process is initiated by calling a function which parses the UI workspace, checks the connections and components in the design, and fills the template holes with code associated with each used component. Specifically, the code generation creates a Python script that contains details about the assembly of the different components.

Figure 4 shows an example of the blocks and code associated with the assembly of the brain to a leg-pair.




```
self.attach(("brain", "core", "botright"),
            ("legpair", "front", "topright"), Fold(-180))
```

Fig. 4. Pictorial representation of ROSLab code generation. The bottom right (botright) of the <brain> component named *core* by the user is connected to the top right (topright) of the <half> component named *front*. The last field inside the parenthesis is used to define how the the half will be folded during the assembling

The user can create different designs however not all configurations are guaranteed to generate a stable mechanical system. The user can easily change connections and adapt the design from the ROSLab UI.

3.3 Robot Fabrication

Once a robot has been designed, it can be compiled to generate manufacturing specifications. Because the system can generate both a 2D representation of the surface of the robot body as well as the 3D volume, a number of rapid fabrication processes, as described in section 2.1 above, can be used to create the specified mechanical structures. Output scripts in the system are used to translate the internal representation of the robot design into fabrication files suitable for manufacture. When compiling the design, the user can select from the available fabrication processes to create the robot body.

Since the system can generate the 3D geometry of the final mechanical body, a solid model output by the system can be sent to a 3D printer for fully autonomous fabrication of the desired structure. Though this process is generally

versatile enough to make even the most complicated shapes, it often lacks the compliance needed for robot mobility. Instead, the 3D volume can also be realized by folding its surface from a patterned flexible 2D sheet, providing both structure and compliance. The underlying geometric data stores the face and edge geometries of the 2D unfolding, and so the body can be automatically self-folded by uniform heating of a patterned 3 layer laminate, as presented in [20]. Alternately, the cuts and folds can be patterned onto a plastic sheet using a laser cutter [21] or desktop vinyl cutter [12], and then manually folded to the final 3D geometry. Finally, it is possible to realize the mechanical body without any custom tools by printing the fold pattern onto a sheet of paper. A user can cut the design out with scissors, and fold the structure according to the printed instructions.

Once the body has been fabricated, the specified electronic components and electromechanical transducers must be mounted onto the body and wired together as specified by the system, with auto-generated firmware loaded onto the core microcontroller. The device then simply needs to be powered on and paired with a user interface for the process to be complete, delivering a fully functioning custom printed robot on demand.

4 Experiments and Results

4.1 Design

We use the ROSLab interface to create robots. A library of basic components was imported from [13], forming the building blocks from which new systems were designed. These systems were then manufactured and operated.

A simple wheeled robot can be specified as two motors attached to a central core. To add stability, a third point of contact, such as a tail, can be added to a free end. Symbolically, this is represented by the following relation:

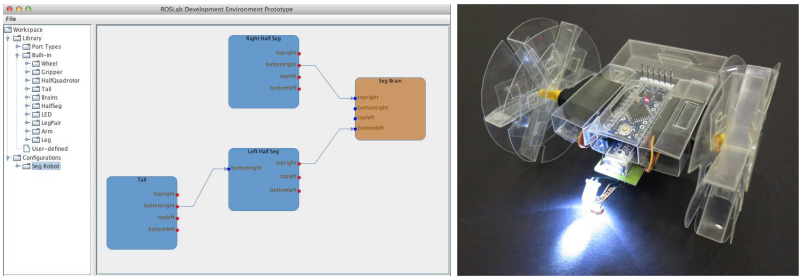
$$Seg = \textit{left wheel} + \textit{core} + \textit{right wheel} + \textit{tail}. \tag{1}$$

The core, motors, and tail were all basic components from the library, and so the design of this simple robot consisted simply of dragging blocks into the workspace and connecting them as per equation 1. The ROSLab source along with the resulting printed robot are shown in figure 5. Similarly designed and fabricated wheels, also from the library, are added to complete the device.

Within the programming environment, labels can be added to the blocks: the central core provides a *brain* containing the low level computational electronics (e.g. processing, communications, and control), while the motor blocks provide *motion*, in particular mobility. These tags can be used to quickly generate alternate, functionally similar designs.

Instead of generating *motion* through wheels, an insect-like crawler can use a leg-pair block from the library to generate a walking motion. Since a tail is no longer needed to provide stability, the new design can be simply expressed as:

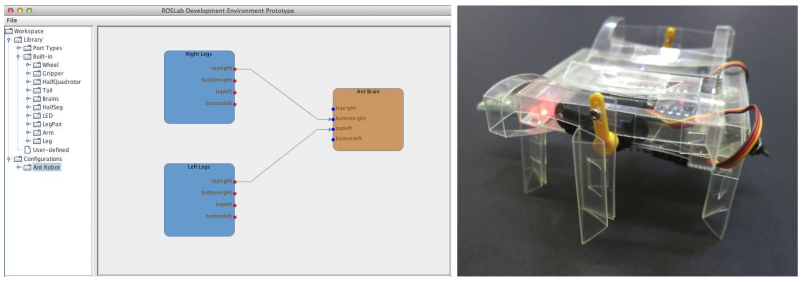
$$Ant = \textit{left leg-pair} + \textit{core} + \textit{right leg-pair}. \tag{2}$$



(a) A simple two wheeled robot design (b) The resulting robot

Fig. 5. A Seg robot designed within the ROSLab programming environment and fabricated in a cut-and-fold process

The ROSLab source and generated robot are shown in figure 6.



(a) Making an insect-like crawler with ROSLab (b) A cut-and-fold robotic ant

Fig. 6. The Ant robot generated by adapting the earlier Seg design

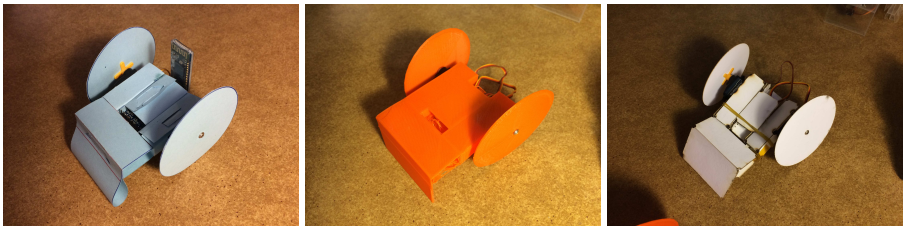
The base components, and therefore the derived designs, are all parametrized; by using this framework, a user has the freedom to adjust design geometries in a variety of ways by tweaking the exposed parameters. As an example, some of the free user-defined parameters for a multi-legged robot are the leg length, the body length, the servo motor type, and the desired microcontroller in the brain.

4.2 Fabrication

The user can also select a manufacturing process. The Seg design from equation 1 was fabricated with four different methods:

(I) a 2D edge unfolding can be encoded in a drawing to laser cut a plastic sheet, as in figure 5(b);

(II) the edge unfolding can also be printed on paper for manual cutting and folding, as in figure 7(a);
(III) the geometry can instead be formed into a solid model, which can get fabricated on a 3D printer as in figure 7(b);
(IV) the geometric information can be used to preprogram an active patterned laminate for self-folding through uniform heating, as in figure 7(c).



(a) Cut and folded by hand (b) 3D printed (c) Self-folded in an oven

Fig. 7. Fabrication files for three additional processes can be generated from the same source as in figure 5 above

These four methods generated robots with the same geometry as specified by the robot design, but with significantly varying tradeoffs.

The traditional cut-and-fold process (I), wherein the structure was folded out of a patterned compliant plastic sheet, was overall a middle-of-the-road process. Though it took the shortest time to fabricate, needing only a single layer 2D cut, its required post-process folding step took a fair amount of user time and skill to assemble. The flexible source material was very useful in creating compliant degrees of freedom for moving parts such as in the legs of the crawler, but needed designed structural reinforcement when stiffness was desired as in the wheels of the Seg. It required an expensive laser cutter for fabrication.

The manual cut-and-fold process (II) was by far the cheapest process, requiring only a printed sheet of card stock and a pair of scissors or a knife. The generated design as shown in figure 8 can be printed using any standard color printer, fully defining the geometry. The lack of tooling was made up for by the labor necessary for fabrication and assembly as the user had to cut then fold the 2D design by hand, as shown in figure 9. The stiffer card stock made the structural elements simpler to design than the plastic substrate above, though the compliant folds were more prone to fatigue.

The 3D printed process (III) took the least user input, directly building the generated 3D geometry as shown in figure 10, but took the longest overall time to fabricate. It produced the most rigid structural body, but could not generate flexible hinges necessary for compliant joints. Though flexible materials are slowly becoming available for 3D printers [22], they are still new and not well characterized.

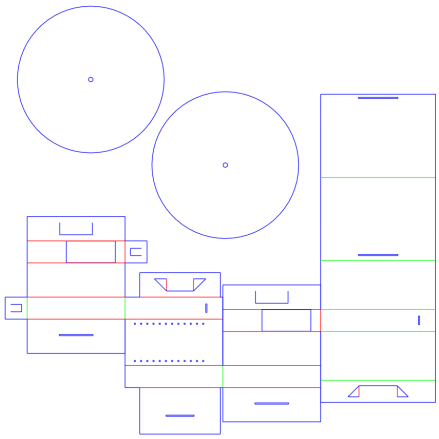
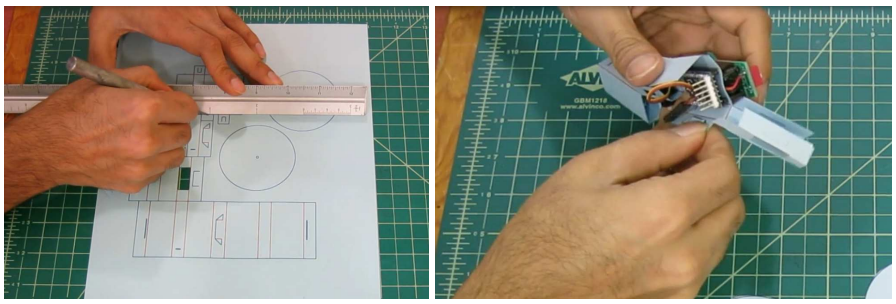


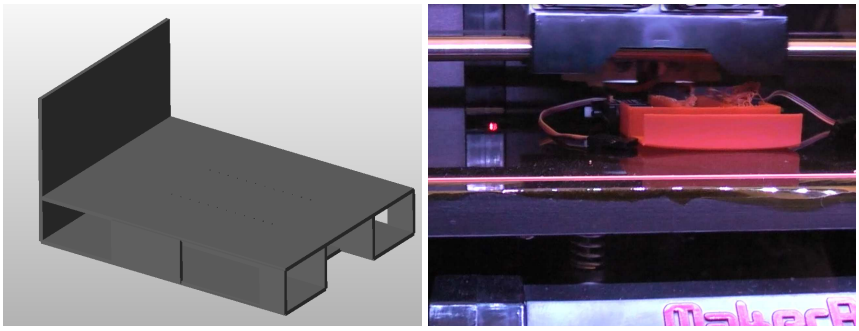
Fig. 8. The design file for a manual cut-and-fold process consists of a color diagram to be printed onto a sheet of cardstock. The user is responsible for cutting along the blue lines; red and green lines represent mountain and valley folds, respectively.



(a) The designs are cut using a knife or scissors.

(b) Electronic components are assembled while folding the body.

Fig. 9. The manual cut-and-fold fabrication process is labor intensive, but requires cheap tools and minimal infrastructure.

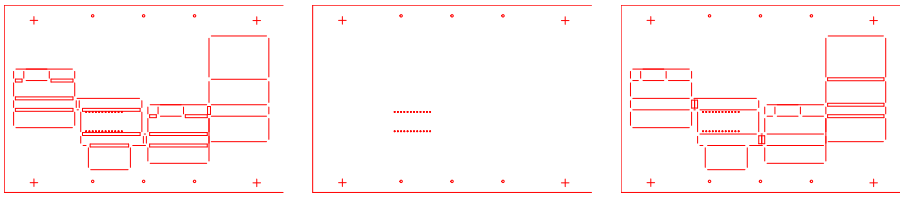


(a) The generated solid model is sent to 3D printer software to manufacture the desired geometry.

(b) Printing can be paused to incorporate embedded electronics, or they can be mounted afterward.

Fig. 10. The designed structure can be directly fabricated on a desktop 3D printer.

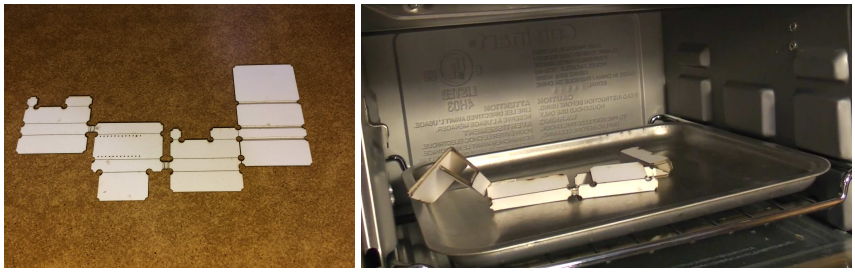
Finally, the self-folding process (IV) combined the benefits of minimal user intervention from the 3D printing with the quicker fabrication of 2D layers. The generated layer designs are shown in figure 11. Though user labor was necessary to laminate the three laser-cut layers, this process was the quickest to fabricate the mechanical body, seen in figure 12. However, the final structure generated by the self-folded laminate occasionally had notable deviations from the designed geometry as the self-folding process halted before achieving to it's final configuration. The structural elements provided excellent rigidity, coming at the expense of less flexible compliant structures.



(a) Top cardboard (b) Shape-memory polymer (c) Bottom cardboard

Fig. 11. A three layer laminate can be assembled to form a self-folding structure that forms the designed geometry under uniform heating.

The comparison of these fabrication methods is summarized in table 1.



(a) Laser cut layers get assembled into a self-folding laminate.

(b) Uniform heating in a toaster oven generates the desired folded geometry.

Fig. 12. Self-folding can automatically generate 3D geometry from a 2D unfolding.

Table 1. Analysis of various rapid fabrication methods

Process	Fabrication time	Assembly time	Body weight	Process Strength	Process Weakness
(I)	2 min	15 min	15.2 g	Controlled compliance	Flexible structure
(II)	10 min	15 min	3.6 g	Low cost	Labor intensive
(III)	90 min	1 min	14.9 g	Structural integrity	Minimal compliance
(IV)	5 min	5 min	12.2 g	Construction speed	Large tolerances

5 Conclusions

The key development presented in this work was a design infrastructure which encapsulated process-independent definitions of robotic building blocks, allowing custom electromechanical systems to be hierarchically designed once for a range of fabrication methods. This was overlaid with the graphical interface of ROSLab allowing for direct transcription of design ideas into mechanism definitions. Together, these form a process flow for personal robot creation from vision to operation that is simple and intuitive. By abstracting and encapsulating the various stages of design, this modular pick-and-place design environment brings custom robot design to the realm of non-expert users. Simple robots such as the Seg and the Ant require only minutes to design; more complex robots can be hierarchically designed in similarly easy stages.

The unified design process highlighted important differences, summarized above, between the various rapid fabrication methods used for printable robotics. As user specifications for custom robots vary widely, so too do desired optimization targets, and thus the design paradigm presented in this work provides a valuable asset for personal robot creation.

This work suggests an important next step towards more autonomous robot design: developing the system to guide design decisions at both ends of the pipeline. Information about the tradeoffs among the fabrication processes can be incorporated into the system, and a recommended fabrication method can

be then presented based on optimization goals on a user specified design. Conversely, given optimization goals and fabrication constraints, components can be suggested to generate a design based on a functional specification.

A full robot compiler can be built upon this framework, allowing casual users to design desired robots from a very high-level functional specification of the problems to be solved. The system presented in this work thus represents a major step forwards towards programmable cyber-physical systems on demand.

Acknowledgments

This material is based on research sponsored by the National Science Foundation awards EFRI-1240383 and CCF-1138967, for which the authors express thanks.

References

1. Constantinos Mavroidis, Kathryn J DeLaurentis, Jey Won, and Munshi Alam. Fabrication of non-assembly mechanisms and robotic systems using rapid prototyping. *Journal of Mechanical Design*, 123(4):516–524, 2001.
2. Charles Richter and Hod Lipson. Untethered hovering flapping flight of a 3d-printed mechanical insect. *Artificial life*, 17(2):73–86, 2011.
3. Jonathan Rossiter, Peter Walters, and Boyko Stoimenov. Printing 3d dielectric elastomer actuators for soft robotics. In *SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*, pages 72870H–72870H. International Society for Optics and Photonics, 2009.
4. Aaron M Hoover and Ronald S Fearing. Fast scale prototyping for folded millirobots. In *Robotics and Automation (ICRA), 2008.*, pages 886–892. IEEE, 2008.
5. Y. Liu, J. Boyles, J. Genzer, and M. Dickey. Self-folding of polymer sheets using local light absorption. *Soft Matter*, 8:1764–1769, 2012.
6. Isao Shimoyama, Hirofumi Miura, Kenji Suzuki, and Yuichi Ezura. Insect-like microrobots with external skeletons. *Control Systems, IEEE*, 13(1):37–41, 1993.
7. S. Brittain et al. Microorigami: Fabrication of small, three-dimensional, metallic structures. *Journal of Physical Chemistry B*, 105(2):347–350, 2001.
8. Elliot Hawkes et al. Programmable matter by folding. *Proceedings of the National Academy of Sciences*, 107(28):12441–12445, 2010.
9. M. Tolley, S. Felton, S. Miyashita, L. Xu, B. Shin, M. Zhou, D. Rus, and R. Wood. Self-folding shape memory laminates for automated fabrication. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013.
10. Robert Lang. *Origami design secrets : mathematical methods for an ancient art*. A K Peters/CRC Press, 2012.
11. E. Demaine, S. Fekete, and R. Lang. Circle packing for origami design is hard. In *Proc. 5th Int'l. Conf. on Origami in Sci., Math. and Edu.*, pages 609–626, 2010.
12. Ankur M. Mehta et al. A scripted printable quadrotor: Rapid design and fabrication of a folded MAV. In *16th International Symposium on Robotics Research*, 2013.
13. Ankur M. Mehta, Joseph DelPreto, and Daniela Rus. Cogeneration of mechanical, electrical, and software designs for printable robots from structural specifications. In *Intelligent Robots and Systems (IROS)*, 2014 (to appear).

14. Mathworks Simulink. http://www.mathworks.com/products/simulink/. [Online; accessed 26-May-2014].
15. NI Labview. . http://www.ni.com/trylabview/. [Online; accessed 26-May-2014].
16. Andrew L King, Lu Feng, Oleg Sokolsky, and Insup Lee. Assuring the safety of on-demand medical cyber-physical systems. In *1st Int'l. Conf. on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, pages 1–6. IEEE, 2013.
17. Lego mindstorms. http://mindstorms.lego.com. [Online; accessed 26-May-2014].
18. VEX Robotics. http://www.vexrobotics.com. [Online; accessed 26-May-2014].
19. Nicola Bezzo, Junkil Park, Andrew King, Peter Geghard, Radoslav Ivanov, and Insup Lee. Demo abstract: Roslab a modular programming environment for robotic applications. In *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), (to appear)*. IEEE, 2014.
20. Byoungkwon An et al. An end-to-end approach to making self-folded 3d surface shapes by uniform heating. In *IEEE International Conference on Robotics and Automation (accepted)*. IEEE, 2014.
21. C.D. Onal, R.J. Wood, and D. Rus. An origami-inspired approach to worm robots. *Mechatronics, IEEE/ASME Transactions on*, 18(2):430–438, April 2013.
22. Makerbot flexible filament. https://store.makerbot.com/flexible-filament. [Online; accessed 26-May-2014].