An End-to-End System for Designing Mechanical Structures for Print-and-fold Robots

Ankur M. Mehta and Daniela Rus[1]

Abstract— **This work presents a script-based development environment aimed at allowing users to easily design and create mechanical bodies for folded plastic robots. The origami-inspired fabrication process is inexpensive and widely accessible, and the tools developed in this work allow for open source design sharing and modular reuse. Designs are generated by recursively combining mechanical components – from primitive building blocks, through mechanisms and assemblies, to full robots – in a flexible yet well-defined manner. This process was used to design robotic elements of increasing complexity up to a multi-degree-of-freedom compliant manipulator arm, demonstrating the power of this system. The developed system is extensible, opening avenues for further research ultimately leading to the development of a complete robot compiler.**

I. Introduction

Robots can come in a variety of form factors with a number of uses, displaying potential to become indispensable for purposes such as academic research, educational outreach, or general household use [1]–[3]. However, the power of robotics comes from customizability in the system design. To enable the general public to personally design and create individualized robots, non-specialized users must be able to go from problem specification to device fabrication rapidly and repeatedly.

Though there are many computer aided design (CAD) packages focusing on various stages of this process, the creation of a robotic system still requires a multitude of such tools, along with the specialized skills needed to navigate each one. This currently leaves robotics within a domain of experts. To bring personalized robots into the homes of the general public, the complete design process needs to be reworked.

This paper presents a toolbox-like system to simplify and streamline the design and manufacture of printable robot bodies. Mechanical structures are fabricated in an origami-inspired process wherein precision patterned and cut 2D sheets of plastic are folded into 3D elements. Designs for these structures are generated by a collection of Python scripts which allow a user to define complex geometries by hierarchically composing simpler building blocks, starting from a library of basic primitives. These designs are then fabricated using an inexpensive desktop paper/vinyl cutter. The resulting process is intuitive, versatile, and extensible, allowing quick and easy design of sophisticated robot bodies. It uses cheap and easily available software and hardware tools and raw materials, making it accessible to a casual hobbyist.

The contributions presented in this paper include:

- a method of abstracting 2D fold patterns via scripted objects,
- a composition paradigm to generate hierarchical print-and-fold mechanical designs,
- a collection of mechanical elements forming a library of building blocks, and
- complex multi-degree-of-freedom robot bodies designed using the aforementioned system.

II. Origami inspired folding

There are many tools and processes available to be used in the fabrication of mechanical structures, optimized for a variety of applications. When considering the task of creating custom designed robot bodies, several concerns take precedence. The design of mechanical structures often involves trial-and-error or successive refinement. In order to facilitate such a process, the fabrication method must be relatively fast and cheap, while still versatile enough to produce the variety of structures necessary for arbitrary robot bodies.

A common rapid prototyping method is 3D printing, made accessible to home users by products such as the MakerBot [4] and popularized through user communities like Thingiverse [5]. Structures are designed by creating 3D solid models, which are then fabricated through an additive manufacturing process using plastic stock material. Current 3D printers can make a wide variety of solid body rigid objects, but are significantly limited in permitting structures with mechanical degrees of freedom. Furthermore, the additive process can often take hours to build a complete model.

Instead, this work focuses primarily on an origami-inspired print-and-fold process for creating 3D structures from 2D patterned sheets of plastic, as presented in e.g. [6], [7]. Such patterns can be quickly printed using an inexpensive desktop paper/vinyl cutter, with raw material cheaply and readily available. This process inherently allows for constrained motion by using patterned folds as hinges. Similar to the diversity displayed by origami creations, the print-and-fold process

*This work was supported by the National Science Foundation, grant numbers 1240383 and 1138967.

[1]A. Mehta and D. Rus are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. email: {mehtank, rus}@csail.mit.edu

allows for a wide range of mechanical designs. This process has already been used to manually design a number of successful standalone robots [6]–[9].

III. Design principles

To enable personal robotics to gain widespread traction, the process by which desired devices are designed must be greatly simplified. With a potential target audience of school children or the general public, the system must be usable by those without a background in engineering design. In particular, there are a number of guiding principles to help translate users' visions into mechanical structures as easily and directly as possible.

Most importantly, the system environment should be intuitive, allowing an inexperienced user to easily understand and implement the design process. A "what you see is what you get" (WYSIWYG) model is ideal, wherein the starting state and all operations directly correlate to their real-world implementation. Generated designs should be easy to share, modify, adapt, and extend. Free and open source tools are preferable to expensive and esoteric computer aided design (CAD) programs, and designs should be similarly unencumbered by proprietary standards.

Earlier work on origami inspired robotics [6]–[8], [10] were lacking in one or more of those goals. Designs are often manually drawn from scratch in sophisticated 2D CAD programs, and can be difficult to visualize as 3D objects. Most robots are created as monolithic integrated designs, and so these issues are compounded as designs grow in size and complexity.

The design package presented in this paper consists of a collection of Python scripts that automate the end-to-end process of designing and fabricating the mechanical body of a folded plastic robot. The simple text-based representation of these designs enables many of the same benefits of the Open Source Software movement such as incremental modification, modular reuse, and community based sharing of designs. Expert users can generate new modules and edit the scripts directly, while novice users need only be concerned with designed modules as building blocks, connecting them through a simple interface that hides the details of the underlying software.

IV. Design environment implementation

The system presented in this paper is outlined in figure 1.

A. Software abstraction

3D folded plastic geometries are defined by their fold patterns – the 2D drawings specifying edges to either cut or fold. However, because of the non-intuitive mapping between the fold pattern and its resulting 3D geometry, mechanical elements are first abstracted into their constituent components, hiding the underlying

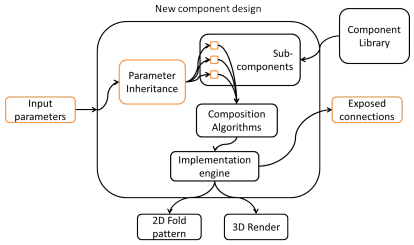

Fig. 1. When creating a new print-and-fold design in the proposed system, a designer needs only to be responsible for specifying the blocks highlighted in orange: which subcomponents are required and how their parameters are set, and what parameters and connections to expose to higher designs.

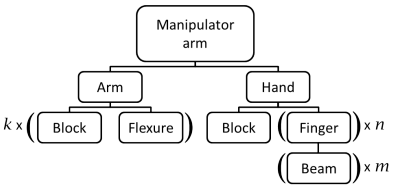

Fig. 2. A robotic manipulator arm can be hierarchically decomposed into modular building blocks; each block in the diagram represents a component in the Python-based design system and can generate the required design files for fabrication. Only the leaf nodes must be coded by an expert designer – the rest can be implemented by simply combining their constituent subcomponents.

2D fold pattern in favor of a building-block representation as in figure 2. These components are self-contained software abstractions of specific mechanical parts represented in the system by Python objects, and are responsible for generating their own fold patterns. While expert users can edit the code directly to fine-tune the resulting geometry, the typical user can instead treat each component atomically as the physical entity it generates, combining them in a modular fashion similar to Lego or Tinkertoys.

This system enables the rapid design and fabrication of robot bodies such as the one diagrammed in figure 2 and shown in figure 10. Given an initial component library populated with some expert-designed primitives, a typical user need only assemble the components as per a specified hierarchy graph in order to design the complete mechanical structure.

B. Component library

The hierarchical design paradigm of the system presented in this work is meant to recursively build up to the desired 3D geometry. The lowest level building blocks of these designs are primitive components representing mechanical structures directly defined by their 2D unfolding. Derived components are then formed by combining primitive components or other derived components. Because primitives must be designed in their entirety, their creation falls under the expert user domain.

Each component implements a common set of methods defined in the parent class, providing a structured implementation for parametrization and composition. These methods can then be called from the casual user domain to enable customization and extensibility within the expert-defined bounds. The parameters of a component are defined by the expert designer to enumerate user-configurable degrees of freedom in the geometric specification of a design, an example of which is shown in lines 6-10 of listing 2. Typical parameters include size, shape, or repetition counts. The component designer is responsible for ensuring that the assigned parameter values are then reflected in the final design of the component. Each component also specifies a list of connections – locations identified by the designer to which other components can safely be attached. This can be seen in lines 12-15 of listing 2.

C. Composition

A design implemented in this system is a constrained combination of components (and is itself another component); specifying a design in this system consists of recursively composing the right collection of components to generate the desired robot body. Composing a new design requires the user to follow a prescribed set of steps to generate the Python class representing the new component. These steps could be carried out by manually generating the required Python script, or an automated user interface could generate the code directly from user input.

The required subcomponents must first be identified and collected, as in line 4 of listing 2. If any do not yet exist in the library, they must be created either by drawing a new primitive component or by composing a new derived component in the same manner being presented here.

By default, the parameter set of the new derived component begins as simply a union of the parameters of all its subcomponents. However, an assembled design often imposes constraints between related parameters of its constituent parts, and so these constraints must be specified relative to the desired parameters of the assembly. The final set of parameters must exactly cover the available geometric degrees of freedom of the composite design, and specify all the parameters of the subcomponents. The parameters in listing 2 are declared in lines 6-10, and set in lines 17-20.

Finally, the subcomponents must be attached to each other along allowable connections to generate the final geometry, as accomplished by lines 22-40 of listing 2. As each subcomponent is actualized as a 2D unfolding of the desired 3D structure, attaching two unfoldings must result in a new planar unfolding of the composite structure. With careful design, this can sometimes be achieved by simply concatenating the two unfoldings along a shared edge. Other times, however, a more rigorous algorithm such as that presented in [11] must be used to generate the composition.

D. Fabrication

Each component in this system is an instance of a common parent class, stored as an executable Python script. When run, this class calls a method to generate the final 2D design file to be sent directly to the cutting machine, thus providing the fabrication path for the printable robot. The class also has methods to generate 3D representations to aid in the design process, demonstrating the static and dynamic appearance and behavior of the resulting structure. Instantiating the final design consists of setting the required parameters, generating a 2D design file, sending it to a cutter to pattern a sheet of plastic, then folding it to the final 3D geometry, using generated models for guidance.

V. Results

The proposed system was used to design and fabricate a number of mechanical structures.

A. Primitive components

There are many possible primitives, a subset of which are presented below. They are typically limited to simple structures; more complex structures can often be subdivided into compositions of these simpler designs.

1) Basic shapes: The simplest non-trivial geometries required to build 3D structures are convex polyhedra. Building blocks such as cubes (shown in figure 3, from the code in listing 1) and beams (shown in figure 4) form the structural elements of most folded plastic robots. The unfolded geometries are specified by Python scripts as first presented in [9]: and encapsulated in the component framework presented above. They are parametrized to allow for user specified geometries. For example, the length, diameter, number of sides, and face angles can all be adjusted for a beam component. The components also publish their available connections – edges along which other components are allowed to connect.

```
1  class Cube(Drawing):
2    def __init__(self, edge, top=True, bot=True):
3      Drawing.__init__(self)
4
5      r = Rectangle(edge, edge)
6      self.append(r, 'rm')
7      if top:
8        self.attach('rm.e0', r, 'e2', 'rt', 90)
9      if bot:
10       self.attach('rm.e2', r, 'e0', 'rb', 90)
11
12     self.renameedge('rm.e1', 'e1')
13     for i in range(4):
14       self.attach('e1', r, 'e3', 'r%d' % i, 90)
15       self.renameedge('r%d.e1' % i, 'e1')
```

Code Listing 1. A cube is formed by joining square faces at right angles.

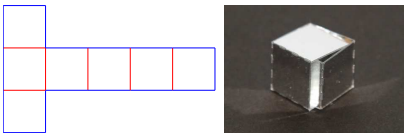

Fig. 3. The code to generate a cube creates the 2D fold pattern on the left with cut edges in blue and folded edges in red. The resulting folded structure is shown on the right. There is an extra overlapping side panel to enable allowable connections on each of the four edges on the top and bottom of the cube.

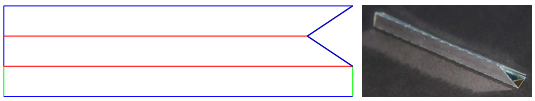

Fig. 4. The fold pattern of a beam with an angled face is shown with available connections at either end of the beam highlighted in green.

2) Joints: Solid components such as the above shapes can then be combined in a number of ways to generate the desired motion profiles for robot bodies. The simplest joint does not require additional components at all. Instead, two solids can simply be connected along an unconstrained folded edge, thus allowing 1 degree of freedom (DOF) rotational motion. A 2 DOF pivot can be formed by combining solid components along two orthogonal edges, or with a dedicated primitive component as seen in figure 5. A more compliant flexure can be created to allow for both rotational and translational motion, as shown in figure 6.

A special type of joint can also be used to combine components without permitting motion. Typically this would be done by combining adjacent faces into a single continuous sheet, but there are many cases where the geometry does not permit that. Instead, a tab and slot can be used to fix two disjoint faces relative to each other. This can be used on a single component as well, as seen in figures 5 and 7, allowing a folded structure to rigidly hold its shape.

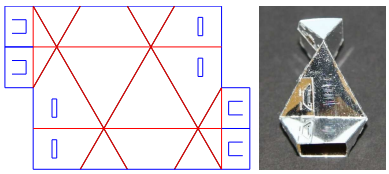

Fig. 5. A robust two degree of freedom pivot allows arbitrary angular motion by forming two orthogonal pivots. Note the tabs and slots added on to the component to join adjacent edges in the folded structure.

Fig. 6. This flexural element permits longitudinal compression as well as in-plane and out-of-plane rotations.

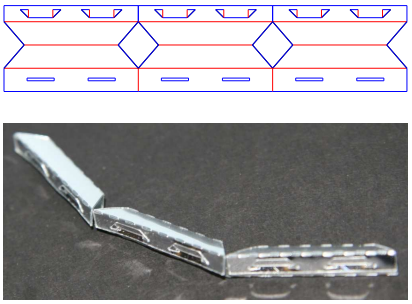

Fig. 7. A basic robotic finger element is simply composed of multiple beams joined with simple folds along their available connections. Again, tabs and slots are used to hold the structure together.

B. Derived components

1) Compound structures: The primitives from above can be combined to make more complex mechanisms. Joining two beams along an edge creates a hinge, permitting the relative rotation of the two solids, akin to a knuckle. A finger can be generated from a number of these knuckles, as seen in figure 7, generated by the code in listing 2.

```
class Finger(Component):
  def defComponents(self):
    ### Identify subcomponents
    self.components.setdefault("knuckle", Beam())

  def defParameters(self):
    ### Declare parameters
    # knuckle parameters inherited by default
    self.parameters.setdefault("cnt", 2) #no. of knuckles
    self.parameters.setdefault("len") #length of finger

  def defConnections(self):
    ### Define edges for connections
    self.connections.setdefault("topedge")
    self.connections.setdefault("botedge")

  def setSubParameters(self):
    ### Impose constraints on parameters
    self.components["knuckle"].parameters["len"] = 1.0 *
      self.parameters["len"] / self.parameters["cnt"]

  def assemble(self):
    ### Assemble the object
    k = self.components["knuckle"]

    # Insert the first knuckle
    self.drawing.append(k.drawing, "k0")
    # Identify the current edges of the finger
    for e in ["topedge", "botedge"]
      self.connections[e] = "k0.%s" % k.connections[e]

    for i in range(1, self.parameters["cnt"]):
      # Attach the next knuckle
      self.drawing.attach(self.connections["botedge"],
        k.drawing,
        k.connections["topedge"],
        "k%d" % i, FOLD)
      # Update the bottom edge of the finger
      self.connections["botedge"] =
        "k%d.%s" % (i, k.connections["botedge"])
```

Code Listing 2. A simple finger is composed of a number of beam components joined end-to-end.

Hybrid structures can also be designed from a collection of different primitives. By alternating cubes with flexural joints, a compliant multi-DOF soft arm can be quickly designed, as shown in figure 8.

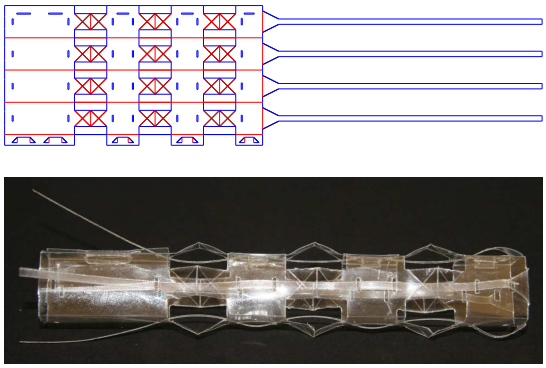

Fig. 8. Joining sequential cube faces with flexural joints allows 3 DOF motion between each pair of cubes, creating a soft robot arm. Additional components, namely the tendons and their slots, were also added into the design to allow for actuation of the arm.

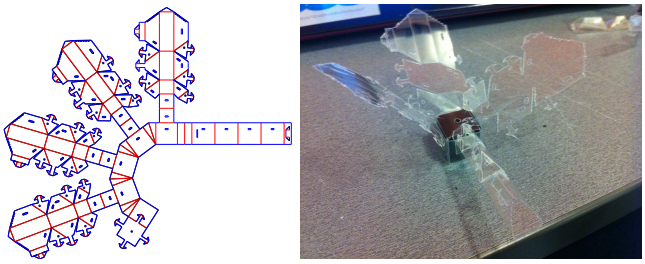

Fig. 9. A number of fingers joined to the edges of a cube forms a gripping hand. A 3D printed model of the central cube, generated alongside the 2D fold pattern and assembled into the structure during the folding process as shown here, can provide rigidity to the device.

2) Robots bodies: These derived mechanisms can be further refined and combined into complete robot bodies. By attaching a number of fingers onto the edges of a solid, a gripper is created as shown in figure 9. Variants of the tabs, cube, and finger components were designed to produce this more sophisticated hand structure. Attaching this gripper onto the end of the compliant arm from above then yields a highly controllable manipulator arm as shown in figure 10.

Designing this final complex robot took no more than combining the relevant building blocks into structures of ever increasing complexity, each step of the way combining only a small number of previously designed components to make the next mechanism. It is here that the power of this system is evident. The first pass for each derivative component took less than an hour to design, code, print, and fold, with subsequent design iterations happening much quicker. A full robot body like the manipulator arm could be assembled from a component library of only primitives in a small handful of hours. If the library included any of the derivative elements along the way, perhaps obtained through an open source sharing community, the final design could have been reached even faster. For an expert designer familiar with the system, the primitives themselves are similarly quick to implement.

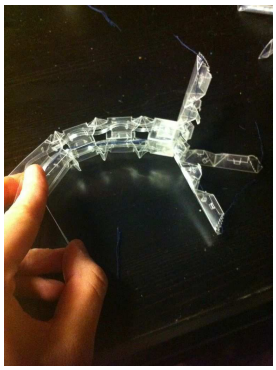

Fig. 10. Putting a gripper at the end of a flexible arm yields a true robot body, a multi-degree of freedom manipulator arm, shown here being manually actuated.

C. Current component database

Over the course of creating bending, crawling, rolling, and flying robots, a number of components were designed and added to the component library to form an ever expanding database of designs. Most components in the library are quite simple, containing only a few subcomponents. However, many contain several levels of hierarchy, recursively building up from set of primitives. In the current library, there are fewer than 10 primitives defining solid objects, joints and hinges, connectors, and specialized geometries such as mounts and wings. However, simple mechanisms form the first tier of compositional hierarchy, and full featured robot bodies are already beginning to appear at the second tier. More complex robots, such as that as diagrammed in figure 2 and shown in figure 10, may take more levels of nonetheless similarly simple components.

VI. Extensions

The system as presented above is capable of generating designs for printable robots of arbitrary complexity using the folded plastic fabrication process. However, the system is not limited there. The scripted nature of the underlying robot representation allows for additional functionality via further software development.

A. 3D printing

Though there are many benefits of the folded plastic process, it is not optimal for some applications. In particular, while lightness and flexibility is valuable for creating moving parts, it is a detriment in situations where rigidity is necessary, for example when considering energy efficiency in flying robots [9] or grip strength in hand-like end effectors. However, since this system produces models of the 3D structures generated by an unfolding, it can also extract solid models to send to a 3D printer. For example, in figure 9, a 3D printed model of the cube forming the palm is co-designed along with the fold pattern. The hybrid process can produce robots more versatile than either fabrication method alone.

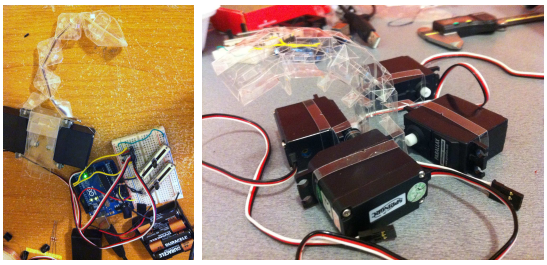

Fig. 11. Mechanical bodies can include mounts for electromechanical devices; complete robots are then formed by installing a processor controlling servomotors to actuate the printed structures. Sensors can also be incorporated to enable feedback control.

B. Electronics

Finally, to generate complete robots instead of robot bodies, the electrical subsystem needs to be included with the mechanical elements. In the robot examples presented above, the designed degrees of freedom were actuated by servomotors winding up cables tied to body elements, as seen in figure 11. The mounts for these servomotors, naturally, are designed as components attached to the final body design. However, the hardware controller, drivers, and application software are currently ad-hoc solutions added manually (in the case of the above, the servos are driven by an Arduino or a GINA board [12] running custom written firmware). Since the degrees of freedom of the robot along with the specific drive actuators are specified by the mechanical body design, it should be possible to automatically design these components as well, incorporating electronic mounting and wiring into the generated design files that get fabricated.

VII. Conclusions

A. Contributions

The primary contribution of this work is a new end-to-end rapid design and fabrication paradigm that specifies mechanical robot bodies by hierarchically composing simpler components. An abstraction was developed allowing for mechanical components to be described by scripts; a few designed primitive components plus rules and algorithms for composition were then sufficient to generate a wide array of robot mechanisms and bodies. This system was implemented in Python and used to generate design files for an origami-inspired plastic print-and-fold process. A database of primitive components was generated and used by the system to design and fabricate a number of robot mechanisms of increasing sophistication and complexity.

B. Future work

In addition to extending this system to create robot designs of ever increasing functionality and complexity, there are additional avenues for research opened up by the work initiated herein.

1) Design decomposition: The basis for this system is the hierarchical modular design of robots by non-experts. However, there are many robot experts that can and do design custom robots (see for example [6]–[9]) to a much higher precision than achievable by a general multi-purpose system. It then becomes interesting to see whether it is possible to decompose experts' designs into parametrizable modules for use in this system. Big data techniques could potentially be applied to extract useful reusable design components from complete designs, thus increasing the depth of the component library and expanding the scope of designable robots.

2) Automated recommendations: Further analysis of generated robot designs could also reveal details of engineering principles. By evaluating existing robot structures, it might be possible to build a recommendation engine into the design system, further reducing the burden on a robot designer. As the corpus of designed robots grows, so too does the engineering knowledge contained in those designs, and the system could evolve from a tool to realize conceptual designs to a tool to propose new designs. Eventually, this leads us down a path to a full fledged robot compiler, able to produce complete robot designs from a specification of the problem to be solved.

Acknowledgments

The authors would like to acknowledge the contributions of Roger Lo and Ian Trase, whose assistance in this project was invaluable.

References

[1] J. Forlizzi and C. DiSalvo, "Service robots in the domestic environment: a study of the roomba vacuum in the home," in *Proc. ACM SIGCHI/SIGART conf. on HRI*, 2006, pp. 258–265.
[2] G. V. Lauder, "Flight of the robofly," *Nature*, vol. 412, no. 16, pp. 688–689, 2001.
[3] African robotics network (AFRON) - design challenge. [Online]. Available: http://www.test.org/doe/
[4] 3D printing — MakerBot. [Online]. Available: http://www.makerbot.com/
[5] Thingiverse - digital design for physical objects. [Online]. Available: http://www.thingiverse.com/
[6] C. D. Onal, M. T. Tolley, K. Koyanagi, R. J. Wood, and D. Rus, "Shape memory alloy actuation of a folded bio-inspired hexapod," in *in ATBio Workshop, IROS*, 2012.
[7] C. D. Onal, R. J. Wood, and D. Rus, "An origami-inspired approach to worm robots," *Mechatronics, IEEE/ASME Transactions on*, vol. 18, no. 2, pp. 430–438, 2013.
[8] D. E. Soltero, B. J. Julian, C. D. Onal, and D. Rus, "A lightweight modular 12-DOF print-and-fold hexapod," in *IROS*, 2013 (to appear).
[9] A. Mehta, D. Rus, K. Mohta, Y. Mulgaonkar, M. Piccoli, and V. Kumar, "A scripted printable quadrotor: Rapid design and fabrication of a folded MAV," 2013 (in review).
[10] S. M. Felton, M. T. Tolley, C. D. Onal, D. Rus, and R. J. Wood, "Robot self-assembly by folding: A printed inchworm robot," in *ICRA*, 2013 (to appear).
[11] C. Sung, E. D. Demaine, M. L. Demaine, and D. Rus, "Joining unfoldings of 3-D surfaces," in *ASME IDETC/CIE*, 2013 (to appear).
[12] A. M. Mehta and K. S. Pister, "Warpwing: A complete open source control platform for miniature robots," in *IROS*. IEEE, 2010, pp. 5169–5174.