Integrated Codesign of Printable Robots

Ankur Mehta
Postdoctoral Associate
Email: mehtank@csail.mit.edu

Joseph DelPreto
Graduate Student
Email: delpreto@csail.mit.edu

Daniela Rus
Professor
Email: rus@csail.mit.edu

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

This work presents a system by which users can easily create printable origami-inspired robots from high level structural specifications. Starting from a library of basic mechanical, electrical, and software building blocks, users can hierarchically assemble integrated electromechanical components and programmed mechanisms. The system compiles those designs to cogenerate complete fabricable outputs: mechanical drawings suitable for direct manufacture, wiring instructions for electronic devices, and firmware and user interface software to control the final robot autonomously or from human input. This process allows everyday users to create on-demand custom printable robots for personal use, without the requisite engineering background, design tools, and cycle time typical of the process today. This paper describes the system and its use, demonstrating its abilities and versatility through the design of several disparate robots.

1 Introduction

Robotic devices have gained widespread traction within research and industrial environments, yet they are still comparably underrepresented in personal everyday life. Creating a new robotic system typically requires domain-specific expertise across a range of disciplines, including mechanics for the structural body, electronics to connect sensors and actuators, and software to specify behaviors. This presents a knowledge barrier for on-demand robot creation, which is further compounded by the need for a variety of computer-aided design tools for implementation. This entire process often involves repeated design iterations, and must be rerun for each new robot desired, keeping the design and fabrication of new robots beyond the realm of casual users.

Computational tools that can create robots from high-level descriptions would allow the general public to obtain custom devices able to accomplish specified functions on-demand, The long-term objective is to develop a hardware compiler that can automatically design and fabricate a robot to accomplish desired goals from a functional specifications of the required tasks. This paper takes a step towards that vision with a system that allows non-experts to simultaneously generate mechanical, electrical, and software designs from a custom structural specification, and then quickly and inexpensively fabricate the designed robot.

The system presented here begins with a database of mechanical, electrical, and software components, encapsulated in a common abstraction suitable for modular composition. Expert users can directly generate new low level building blocks, while both expert and casual users can make custom electromechanical devices by hierarchically connecting existing blocks. The component abstraction allows for parametrization in terms of geometric and physical properties to allow further fine-grained customizability.

Component hierarchies are compiled by the system into a collection of files necessary for a user to manufacture the specified design: the mechanical structure is made using 2D or 3D rapid fabrication processes from generated fabrication drawings, the user assembles the electrical subsystem onto that structure guided by a bill of materials and wiring instructions, and firmware gets loaded onto the central microcontroller. The resulting robot can be wirelessly controlled from a generated user interface (UI), autonomously controlled from generated application software, or user programmed with custom behaviors with help from a generated control library.

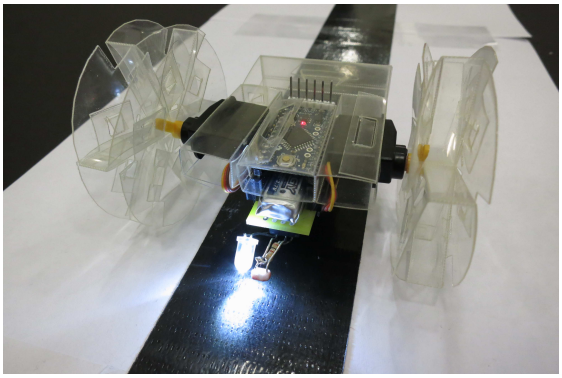


Fig. 1. Complete mechanical, electrical, and software subsystem designs for an autonomous line-following wheeled robot are generated from a functional description of the logical flow of information from a light sensor to the wheels.

This work extends the previous work reported in [1–3] by unifying and integrating mechanical, electrical, and software subsystem designs under a common functional specification based on information flow. In particular, this paper presents a modular, hierarchical, component based abstraction for integrated electromechanical robot design specification, composed of the following:

1. a scripted code object which encapsulates mechanical, electrical, and software designs into self-contained parameterizable components,
2. a process to hierarchically compose such elements along well-defined interfaces into electromechanical mechanisms of arbitrary complexity,
3. methods to render the component design as directly manufacturable fabrication specifications,
4. a library of base and derived components, and
5. several robots designed, fabricated, and operated using the proposed system.

2 Related Work

This paper joins a body of work including rapid fabrication technologies, modular design methods, and robotic system specification.

There has been much work to enable the rapid fabrication of arbitrary geometries. On-demand 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 [4]. Limited workarounds do exist [5,6]; these often lack robustness or reliability, though current technology has been improving. 3D printers are also plagued by long fabrication times – though quicker than conventional manufacturing processes, parts still take on the order of hours to build.

Mechanical structures can alternatively 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 [7], single layer plastic film [8], or more exotic materials [9, 10]. These designs can be manually folded by hand, folded by embedded or external active stimuli, or passively folded by controlled environmental conditions [11, 12]. These processes have been used to create passive 3D structures [11, 13] as well as active programmable robots [9, 14, 15]. The system presented in this paper employs this fabrication process.

The 2D fabrication methods have been employed for rapid prototyping, being able to manufacture devices in a time frame of minutes. However, creating the fabrication drawings for these processes typically requires careful hand design by experienced engineers using sophisticated 2D CAD programs, and were difficult to visualize as 3D objects. Custom electronic circuitry and software was required to drive the actuators. The robots were created as monolithic integrated designs, and so these issues would compound as designs grow in size and complexity. There have been attempts to automate the decomposition of 3D shapes, notably in [16–18], to generate 2D fold patterns. These tools and algorithms, however, focus mostly on solid objects, employing various heuristics to generate polyhedra obeying certain rules. Compliant and kinematic structures are not addressed.

The use of modular methods can greatly simplify, clarify, and speed up system design [19], and has been widely adopted throughout the software development communities. Modular design can also be applied to robot creation [20–22] to achieve the same benefits over ad-hoc custom design, to the point of inspiring commercial offerings, e.g. [23–25]. However, these all call upon the use of a discrete set of specially designed modular building blocks, adding expense with limited configuration space. This work adapts such a design method to use discrete but off-the-shelf electronics along with the 2D cut-and-fold fabrication process above, enabling a much broader range of customizability from cheaper raw materials.

Finally, while physical systems are often designed in an interactive graphical environment, there has been work on creating domain specific programming languages to specify hardware designs using software for electrical circuits [26] and rigid bodies [27]. This software-defined-hardware paradigm has been used to define robotic designs for simulations using a scripted modular language in [21, 28]; the system presented in this paper employs a similar design method for user input, but the focuses on physical device creation, compiling into directly fabricable outputs.

3 Design Paradigm

Traditionally, robots are created over a sequence of phases during which mechanical, electrical, and software subsystems are designed and then integrated. Because of the deep interplay between the separate subsystems, the entire process must be largely recreated for each distinct robot. If personal robotics is to gain widespread traction, however, the process by which devices are designed must be greatly simplified. Furthermore, with school children or the general public as target audiences, the system must be usable by those without engineering backgrounds. The system presented in this work therefore follows a number of guiding principles to help translate users' visions into mechanical structures as easily and directly as possible.

The presented system leverages a modular paradigm that allows electrical, mechanical, and software components to be coupled at the lowest level by experts, and then abstracted into functionally defined blocks usable by novices. As a user combines these electromechanical modules, subsystem designs are assembled behind the scenes to maintain an integrated design throughout the modular composition. Complexity is managed by nesting hierarchical constructions in an intuitive design abstraction, allowing an inexperienced user to easily understand and utilize the design process. Ultimately, this high-level assembly of modules can be directly compiled to generate fabrication specifications to manufacture the robot.

This paradigm allows a high degree of modular reuse, allowing for incremental adaptation from earlier designs. Furthermore, the generated designs take the form of text-based code scripts, and are therefore easy to share, modify, adapt, and extend using free and open source tools, unencumbered by proprietary standards.

3.1 Modular encapsulation

The fundamental unit of abstraction in this design system is the component object. This represents an individual design element that can accomplish a self-contained set of functionality, and provides the required encapsulation to define and create that device. The simplest components are the basic building blocks of electromechanical structures, such as a mechanical beam, discrete servomotor, or code method. Complex components can be hierarchically built by combining existing components as described below in Sec. 3.3. These higher order components can represent anything from mechanical assemblies and control systems to integrated electromechanical mechanisms and full robots.

Components can be parameterized, allowing for fine-tuned design customization. These parameters define adjustable values that quantitatively but not qualitatively change the functionality of design elements. Examples of parameters include geometric dimensions, electronic device models, or feedback loop gains. They therefore provide a means by which casual users can customize a component's behavior without changing its overall function. Assigning values to all parameters of a component serves to fully specify that element, and is sufficient to generate fabricable design files to manufacture that object.

3.2 Information flow

In order to make the behavior of a design readily apparent, the design process focuses on describing the flow and manipulation of information among the various components. Each component is an encapsulated module that can be conceptually replaced by a parameterized "black box" mapping a set of inputs to outputs. These ports conceptually transmit information related to the behavior defined by that component. Ports can take on a number of different types, depending on the nature of the information transmitted therein. Mechanical ports transmit information in the form of spatial position and orientation, and a connection along mechanical ports is realized by a physical attachment of mechanical patches. Electrical ports transmit electrical signals, and their connections are realized by wires or other communication channels. Data ports represent the flow of conceptual information such as software values, and are realized by code functions or variables.

Components can provide ports of multiple types, and in fact this forms the basis for cogenerating robotic subsystems across a design. For example, electromechanical transducers such as sensors and actuators translate between electrical and mechanical ports, while hardware drivers translate between data and electrical ports. Controls in a User Interface (UI) can be seen as data information sources, while end effectors provide mechanical information sinks. Each component is defined by the specific mapping between the information at its inputs to the information at its outputs, and this mapping is often a function of the component's parameters.

Several ports can be collected into a single interface, allowing a logical grouping of functionally related ports to be connected simultaneously. For example, a plug can provide both electrical and mechanical connectivity, or a communications transceiver can bundle several data ports into a single channel.

3.3 Hierarchical composition

Each design made in this system is itself a component. A design library is initially populated with basic building blocks designed by experts to provide a core set of functionality from manually defined specifications. From there, the general design environment allows users to create new components by attaching existing components from the library along their exposed interfaces. A newly created component is then defined by its collection of sub-components and how their interfaces are connected, as diagrammed in Fig. 2.

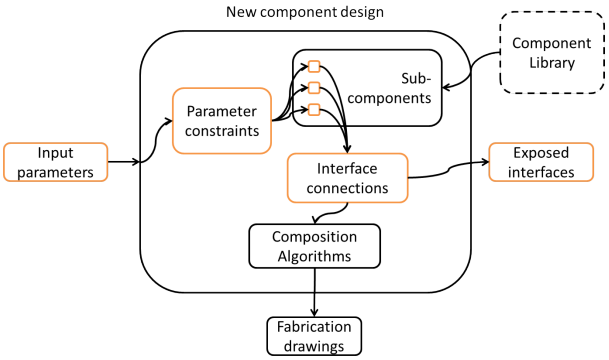


Fig. 2. When creating a new electromechanical component, a designer needs only to be responsible for specifying the blocks highlighted in orange: which subcomponents are required from the library, how their parameters and interfaces are constrained, and what parameters and connections to expose to higher designs.

The new super-component is also parameterized; all sub-component parameters are then either manually specified or defined as functions of the super-component's parameters. Similarly, the new super-component can expose interfaces inherited from the sub-components for future connections. In this way, newly designed components get added to the library and can be used in higher order designs.

As components from the library get connected along their interfaces, an information path is traced from a set of inputs, through various transformations, to a set of outputs. The overall input/output relationship defines the functionality of the designed mechanism, while the specific path defines its implementation.

4 Scripted hardware design

The design system described in the above sections was implemented as a Python package, with the designs themselves defined and generated by Python scripts. This purely software-defined-hardware paradigm allows for general cross-platform compatibility, and inherits many of the benefits inherent in software development.

4.1 Component object

The component object is a Python class that implements the functionality described in the previous section. Every component contains a list of its parameters and interfaces; a derived component additionally contains references to its constituent subcomponents with functions constraining their parameters, as well as a list of connections defining which pairs of subcomponent interfaces are connected. Each component includes a collection of executable script objects that, when run, generate fabricable designs to implement its specified input / output functionality. For basic building blocks, these code elements must be manually written in Python by an expert designer. In a derived composition, however, these scripts are autogenerated by composing the respective elements of its subcomponents.

Component objects, when written directly by experts, also represent a hierarchy through a structure of class inheritance. Any component may serve as the superclass for a more specialized component, allowing for new definitions to inherit rules, ports, types, and design principles from previously designed components. This makes it easier for experts to design new components since most of the tedious details have already been dealt with by pre-existing, higher level components.

4.2 Composable script elements

The composable script elements form the software that defines the hardware specified by the components, with the hardware comprising various mechanical, electrical, and software subsystems of the complete electromechanical device. Not all components will have all subsystems, but a derived component will contain scripts for every subsystem contained in the components comprising its hierarchy. These scripts are described in more detail in Sec. 5.

Executing the script in a Python interpreter generates output files that specify the creation of the respective subsystem. Mechanical output files include a 2D vector drawing that can be directly sent to a laser or vinyl cutter to create a cut-and-fold structure, and a solid object file that can be built using a 3D printer. Electrical subsystem output files include a bill of materials and wiring instructions to assemble the desired circuit. Software subsystems can include a set of program files to be loaded onto the central microcontroller, off-board UI apps for human control of the robot, and code libraries that simplify the creation of custom-written control programs.

4.3 Component ports

The interfaces of a component can contain a number of ports which represent pathways by which information is passed to and from other components in a hierarchical design. Like components themselves, parameters can be used to quantitatively customize ports during design and implementation. The Python class defining a port specifies its type, describing what kind of information it passes and in which direction. When instantiating a connection between interfaces of two components, ports can ensure that they are connected to ports of appropriate reciprocal types, allowing information to logically flow from one component into the other. When establishing such a connection, the component class delegates to each port the task of combining the attached composable script elements in each subcomponent into a single set of scripts that generate

integrated designs for the composite device.

When connecting two components together, port requirements can be used to automatically determine appropriate interfaces to join. For example, when connecting an analog sensor to a microcontroller, the sensor's electrical output port must connect to an electrical input port that supports analog readings, and so the respective interfaces will be automatically selected. Similarly, the data, electrical, and mechanical subsystems will automatically be joined as appropriate. Additional rules for port matching can be programmed to provide more sophisticated design guidance and automation, and can be used to provide compiler-level verification of design decisions.

To achieve these rules for automatic connections and design principles, the port classes form a structure of class inheritance similar to that described above for the components. At the most general level, the Port class allows for the specification of port types to which it should connect and port types to which it cannot connect, as well as basic rules for the verification and determination of connections. New types of ports defined by experts then inherit from existing port types, and can extend their lists of recommended or forbidden types as well as their rules for verifying and forging connections. For example, a general class for an output port may simply specify that it cannot connect to another output port and recommend connecting to an input port. A slightly more specialized port type such as a PWM output can inherit from the general output type and add that a PWM input is a recommended type. The rule for not connecting to other outputs is already specified (and will apply to any port types that inherit from the general output type). Moreover, the system can automatically determine that the PWM input inherits from the more general input type, and thus the new rule will be applied first when searching for ports with which to make new connections. In this way, rules for verification and determining connections can quickly become quite sophisticated with little additional effort on the part of the experts.

4.4 Software infrastructure

The Python software package that implements the integrated co-design environment is divide into three main collections: 1) the classes that define the underlying code architecture of the software-defined-hardware and contain the algorithms used to compose and instantiate designs are collected into an API; 2) that API is used by expert designers to come up with a set of basic components to populate a library; and 3) a collection of utilities and builder applications to aid a casual user to assemble library components into higher order electromechanical designs (which can then also be added to the library).

The library is a folder that stores all components created in this environment; a small subset of the components currently defined in the library are displayed in Fig. 3. An expert-designed basic component takes the form of a Python script in which all the component script elements are manually specified, and is saved in this folder. Derived components, on the other hand, are specified only in terms of its subcomponent breakdown, and so do not need to be written as a Python script. Instead, the design can be stored in plain text using the YAML markup language [29]. A user can write the YAML by hand, or use a number of utilities to generate the YAML in a more interactive manner. Currently both a text-based console interface and a simple graphical interface exist to allow non-expert users to build robotic designs by intuitively assembling building blocks; a web interface is currently under development. Of course, derived components can also be created with a python script, giving greater configurability to an expert user.

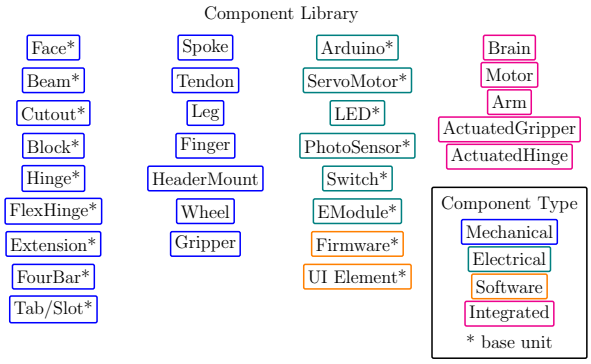


Fig. 3. A library of modular components enables robotic design to be reduced to hierarchical composition of pre-designed elements. The starred components are basic building blocks defined from scripts by experts; the rest have been assembled within the design system and added to the library.

5 Subsystem implementation

5.1 Mechanical system

Mechanical building blocks are used to define the physical structure and degrees of freedom of the robot body. These components also present input/output ports, defining the physical positions and orientations of a subset of the mechanical design, often a face or an edge, which can interface with other components or the environment. To maintain universality, designs are generated and stored in a process-independent data structure; process-specific plugins can then be used on those designs to generate fabrication-ready outputs.

5.1.1 Implementation encapsulation

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

Fig. 4, generated from the code in listing 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.

```
import library
b = library.getComponent("Beam")
b.setParameter("length", 100)
b.setParameter("beamwidth", 10)
b.setParameter("shape", 3)
b.makeOutput()
```

Listing 1. Python script defining a mechanical beam

5.1.2 Mechanical ports

The ports of a mechanical structure describe locations along which additional mechanical elements can be physically attached; the information that flows through them is the spatial configuration of that patch. For rigid elements, the information that is assigned to the output port is an affine transform applied to the location to the input. For example, the beam described in the previous section can have one input and one output port, defined to be the two ends of the beam. The value of the output port is a location set to be that of the input port, offset by a distance equal to the length of the beam.

Mechanical components can also include degrees of freedom; in that case setting an input value can result in a non-rigid deformation of the mechanical device. This is useful in generating motion for robotic mechanisms.

5.1.3 Composition

Mechanical components can be connected along mechanical ports to generate more complex geometries. Depending on the nature of these connections, additional mechanical ports may be opened up in a composition if the resulting geometry has additional unconstrained degrees of freedom. A simple composite structure is demonstrated in Fig. 5 from the YAML definition in listing 2. In contrast to the primitive Beam component described above, this new design is entirely defined by its subcomponent structure, and does not need to be designed in Python. The subcomponent ports are edges, connected by a flexible joint for compliant motion. This hinge defines an additional mechanical port for the angle of the flexure.

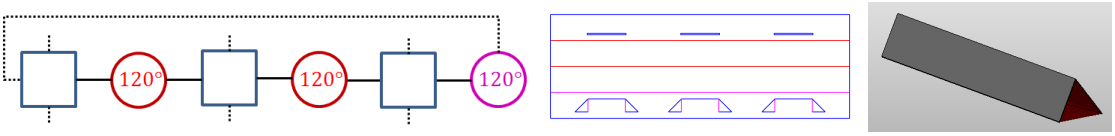
```
# Set parameters of new Component
parameters: {width, fullLength, jointAngle}

# Define subcomponents and
#   constrain subcomponent parameters
subcomponents:
  beam1:
    object: Beam
    parameters:
      angle: 45
      beamwidth: {parameter: width}
      length:
        parameter: fullLength
        function: x * .4
      shape: 3
  beam2:
    object: Beam
    parameters:
      angle: 45
      beamwidth: {parameter: width}
      length:
        parameter: fullLength
        function: x * .6
      shape: 3

# Connect subcomponents along interfaces
connections:
- - [beam1, botedge]
  - [beam2, topedge]
  - {angle: {parameter: jointAngle, function: 'x'}}

# Expose interfaces inherited from subcomponents
interfaces:
  botedge: {interface: botedge, subcomponent: beam2}
  botface: {interface: botface, subcomponent: beam2}
  topedge: {interface: topedge, subcomponent: beam1}
  topface: {interface: topface, subcomponent: beam1}
```

Listing 2. YAML specification of the hierarchical design of a finger consisting of two beams with a kinematic degree of freedom

5.2 Electrical system

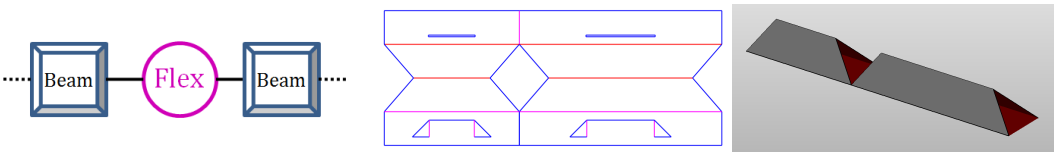
Within the hierarchical composition of elements, the electrical subsystem is determined by the topology of the electrical devices and connections. Each device added to the design may contain electrical ports, and connections between these ports represent physical connections that describe how electrical information flows throughout the design. A library of basic components has been developed which addresses the typical electrical needs of a robotic system, namely various forms of sensing, actuation, processing, communication, and user interfacing. Yet this subsystem is not designed in isolation, since many electromechanical devices required to accomplish physical tasks are often distributed throughout the robot. Hardware modules have been developed to facilitate this codesign and allow the electrical layout to mirror the mechanical structure. Additionally, components may directly serve as interfaces between various subsystems by containing ports of many different types in addition to electrical.

The library has been populated with discrete electronic components that have standardized header connectors, allowing for simple plug-in connections between devices. This eliminates the requirement for custom printed circuit boards (PCBs) to handle electrical interconnects. However, the system does not preclude such design elements – the extensible nature of the component abstraction can allow an expert designer



(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. 4. Outputs generated from the code in listing 1



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

Fig. 5. Outputs generated from the YAML definition in listing 2

to implement a PCB composable to enable more complex electrical devices and circuits, at the expense of in-home fabricability for a casual user.

5.2.1 Information Flow

As in the mechanical layout, the sources and sinks of electrical information can reveal the underlying structure of the design. In the case of electrical signals, units such as sensors or communication modules can source electrical information, and devices such as servos or LEDs can sink electrical information. Note that the overall information flow does not necessarily start or stop at these devices, but the electrical information does – for example, a communication module may take in a conceptual value and convert it to electrical information, and the servo takes in electrical information and converts it to mechanical information. These devices may therefore serve as electrical sources while being sinks for other types of information, and vice versa. By only considering the electrical sources and sinks though, the electrical sub-design can be made apparent.

Less informative ports such as power connections, and details such as particular pins used, are abstracted away from the user during the design process. At fabrication time, the system automatically creates power connections, chooses particular pins and pin types, and inserts devices such as microcontrollers or power converters if necessary so that only the informational flow needs to be considered during design.

5.2.2 Electrical Hardware Modules

The modularity and scalability of the electrical system is enhanced by plug-and-play hardware modules that serve as interfaces between electrical devices and the main controller. Each module uses an ATtiny85 microcontroller to drive three general ports, as shown in Fig. 6, which can be independently configured as digital outputs, PWM outputs, digital inputs, or analog inputs. Since these modules are designed to be plug-and-play, however, the code loaded on the modules does not change according to the robot design; on startup, the main controller sends the modules any necessary design-specific data such as what pin types to use. Communication is established between a module and the main controller via a one-wire serial protocol, and messages are then exchanged such that devices can be attached to the modules as if they were being attached to the main controller. Modules can also be chained together, in which case messages are passed along the chain until they reach the desired module. In this way, the number of possible devices is no longer limited by the number of pins on the main controller. This configuration also facilitates the physical distribution of devices across the robot while reducing the wiring complexity, thus allowing the electrical layout to more naturally mirror the mechanical layout. The flexible nature of the hardware modules can also be leveraged during automatic design since the system can insert them where needed in order to join various devices together.

5.3 Software System

In general, electrical systems on a robot are controlled by processors such as microcontrollers, and thus the design of an electrical subsystem must directly interact with the design of a software subsystem. This subsystem includes driver firmware for controlling devices, higher level microcontroller code, UI generation, and the ability to automatically generate code for robot behaviors. Within this abstract subsystem, components may pass information such as a desired servo angle or a UI slider position as conceptual data values. Components contain software snippets written by experts which represent the code needed for the block to perform its required function, and can contain code tags that reference design-specific information. At fabrica-

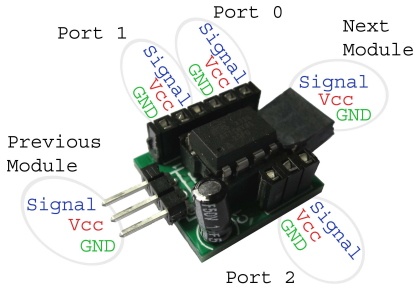


Fig. 6. Each electrical module features connections for an upstream and downstream module as well as three ports for connecting devices such as servos, LEDs, or digital and analog sensors. These modules are designed to be plug-and-play and do not require reprogramming based upon location or connected devices.

tion time, the data network can be analyzed and all of the software snippets can be pooled together to generate software which reflects the designed data flow. The collection of provided components allows users to design at an abstraction level with which they are comfortable; expert users can use low-level code directly, intermediate users can use automatically generated code libraries to aid the writing of custom code, and novice users can intuitively link ports to specify behaviors and generate a graphical UI.

5.3.1 Hardware Drivers

At the lowest level, code must be generated which allows the main controllers to directly interact with the electrical devices. Towards this end, components called drivers perform conversions between software, abstract data, and electrical signals; for example, a servo driver accepts as input a conceptual data value such as an angle, and outputs a software snippet representing the knowledge of how to realize that value as an electrical signal. This output can also adapt to the design topology through the use of parameters. Drivers are therefore sources for the software subsystem and sinks for abstract data – they serve as indirect interfaces between the conceptual software realm and the physical electrical realm. Such examples illustrate that the designed sub-systems are not isolated from each other, but rather interact both through the types of information they process as well as through design parameters that affect how the information processing takes place.

5.3.2 User Interface Elements

While hardware drivers are necessary abstraction barriers between the software and electrical realms, they are often included at a low level of the design hierarchy and not made transparent to the novice user. Other data sources such as UI elements, however, can be intuitively included in higher level designs and allow for humans to become information sources. In this case, elements can represent UI elements such as joysticks, buttons, switches, or sliders. These then generate conceptual data values that can be processed by other software blocks and ultimately control actuators or otherwise affect the robot's behavior. In this way, the user interface can be designed in parallel with the robot itself, such that the design process for the robot subsystems can interact with the design process for its human interface.

5.3.3 Data Manipulation

Although drivers and UI elements serve as conceptual sources by translating data or human interaction into software and thereby allow for the direct control of various devices, a robot should also be able to perform some autonomous behavior. An intuitive way to design such behavior is to link data sources and sinks together – for example, linking a light sensor output to a servo angle input through some simple function can create a line following robot. To facilitate such information flow, various library components can manipulate data within the conceptual realm. For example, such a block may take in data from a sensor and scale it to a value that is meaningful to a servo driver. By serving as an interface between sensors and actuators, this conversion enables autonomous behavior to be easily described in the design environment. Similarly, data may be converted from a human-readable version to a machine-readable version, facilitating human interaction with the final design. Thus the flow of conceptual data within the design largely describes the resulting behavior of the robot.

5.3.4 Programming Blocks

While the data manipulation components allow for the direct linking of devices throughout the design and the seamless integration of the conceptual and physical realms of the design, more advanced users may want to specify robot behaviors in a more arbitrary manner. Library components are therefore provided which allow for graphically writing arbitrary code. These blocks include if/else statements, loops, and the declaration and definition of variables or methods. Using these blocks, arbitrary code can be created to specify robot behavior. Such blocks also include data ports which allow the software to directly utilize the information flow of the design; for example, the block to set a variable may be connected to the output of a sensor. Details of how the data signals are converted into software (such as how the sensor is read) can be encapsulated lower in the hierarchy and thus abstracted away from the user.

5.3.5 Software Sinks

The various elements described above translate conceptual data into software to realize the abstract flow of data defined by the design. Ultimately, these

software outputs must be processed and pooled together into a coherent library for a particular device. Towards this end, a microcontroller such as an Arduino may be a sink for the drivers software, or an Android device may be a sink for the User Interface software.

The software snippets written by experts and included in the components can include various code tags that are processed once the design is complete. These may include pin numbers, device indices, counts of other devices in the design, device types, or other design parameters. These allow experts to write code snippets that are flexible and dependent upon the final design topology. In addition, they may write multiple code snippets and provide rules for choosing between them based upon design parameters – this allows the software sources to adjust their generated software according to the type of sink to which they are ultimately connected.

Once the flow of software is well defined, the sinks can pool the code from all of the connected inputs into usable code. This includes processing the aforementioned code tags so that the code reflects the final design. This may also include generating code for interfacing with the hardware modules, if any are present, by abstracting away the implementation details from users of the final code library. When the system analyzes the overall topology, it assigns each device a "virtual pin number," as shown in figure 7, and this list of virtual pins is presented to the user along with the building instructions. If the user then opens a generated Arduino file, for example, they can interface with the attached devices by simply using the virtual pin numbers – if a sensor was assigned a virtual pin number of 3, a user can simply call robot.analogRead(3) as if it was connected directly to the brain. The generated robot library will determine the corresponding module and physical pin, and send the command along the appropriate chain. The user can therefore program as they normally would program an Arduino, and all of the work for interfacing with the actual electronic layout is done behind the scenes.

5.4 Integrated components

Because of the common API used by each component, design elements can be integrated across subsystems. A typical combination connects the electrical output of a software driver block to the input of an electromechanical transducer to give a logical actuator element driven by data signals. Higher levels of integration can further connect that block's data input to a UI data source and mechanical output to a structural degree of freedom to yield a component representing self-contained robotic mechanism. Such integrated components autonomously cogenerate mechanical structures, electronic wiring diagrams, microcontroller firmware, and user interfaces as shown in Fig. 8.

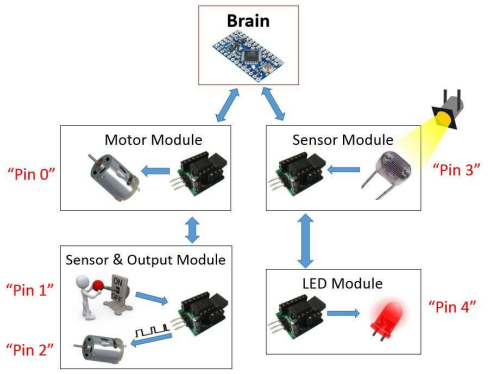


Fig. 7. Each device is automatically assigned a virtual pin number. Users can then control the robot using the virtual pin numbers, so that knowledge of the actual chain configuration is not required.

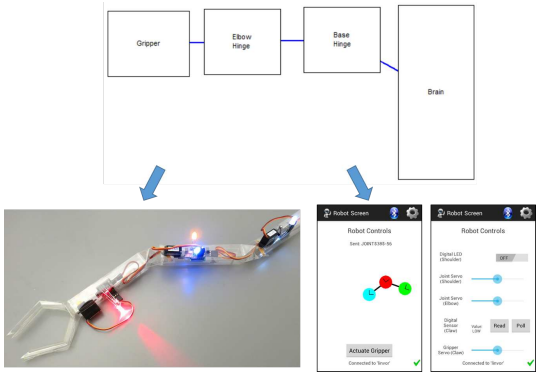


Fig. 8. An intuitive connection of integrated components simultaneously produces a collection of outputs for immediate fabrication, producing designs across all required subsystems.

6 Case studies*

The design environment presented herein was used to create a variety of different robots. Because the system is process agnostic, any of a number of rapid prototyping manufacturing techniques can be used to realize the generated designs. The robots presented in this section were all laser cut from 0.010" (0.25mm) thick polyester (PET) sheet, then folded to their final 3D geometries. The electronic components were incorporated into the structure during the folding process. Generated drawings guide the user along the steps in the folding process, aiding a novice designer in the fabrication of these robots. More hands-off fabrication processes can be used to reduce the skill requirements on the user – the same designs were made using e.g. 3D printed structures and origami self-folding laminates in [30].

6.1 Two-Wheeled Roller

A two-wheeled mobile robot base is shown in Fig. 10. The robot, nicknamed the Seg, is specified by

*Some work in this section was previously published in [3].

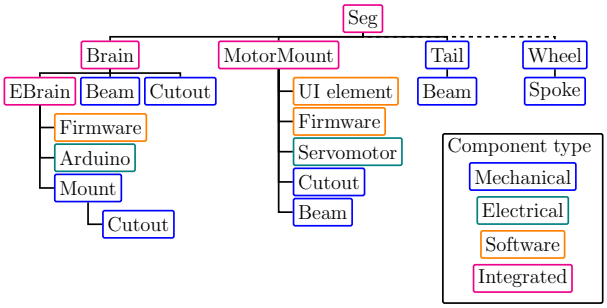


Fig. 9. Each node on this tree represents a component in the design of the two-wheeled robot, generated solely by composing its child nodes. The leaf nodes were design by expert designers, but every higher level of the design can be assembled from its children by a casual user.

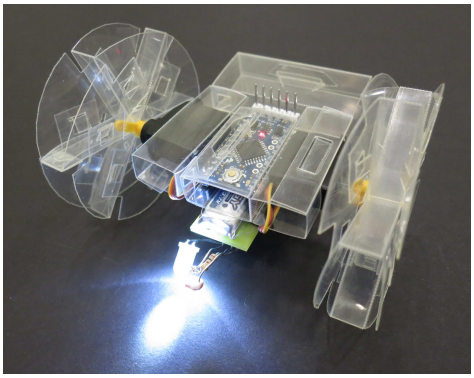


Fig. 10. The Seg, a two-wheeled mobile robot, was compiled from modular electromechanical components. Electrical components are directly connected to the brain using the modular software interface.

three parameters:

1. the specific microprocessor module used (in this case the Arduino Pro Mini) and its dimensions,
2. the specific continuous rotation servos used as drive motors (in this case Turnigy TGY-1370's) and its dimensions, and
3. the desired ground clearance (in this case set to be 25mm).

The user can design a Seg from an extended electromechanical component library by attaching two motor mounts to a central body, along with a tail for stability. The functionality of the robot is defined by the flow of information from a human source to the drive wheels as a mechanical sink. A component defining a UI slider is the information source, generating information at a data output port from human interaction. A firmware driver is the next component in the chain, converting the data value from the UI element into an electrical signal on a microcontroller output pin. An electrical component defining the servomotor actuator takes the electrical signal and generates a mechanical output angle of the servo horn. This finally gets connected to the wheel for a mechanical sink, achieving the desired robot functionality.

In practice, the design process is greatly simplified by breaking the design into a multilayer hierarchy. For instance, the components defining the servomotor firmware driver and discrete electronic device are combined into a higher level integrated component that translates a data input value to a mechanical output, abstracting away the internal details until the final design outputs are generated. The final component-based hierarchical design is presented in Fig. 9.

Since the necessary mechanical, electrical, and software designs are encapsulated within the components, the compilation of the complete design creates mechanical drawings for the body and wheels as well as code for the central microcontroller. The electrical subsystem gets resolved into a wiring diagram, software and firmware snippets get pooled together, and the mechanical mounts get physically linked. Instructions for connecting the modules and devices are then displayed to the user, and an auto-generated smartphone app containing the controlling UI blocks can immediately be used to drive the robot. A summary of the robot's characteristics are provided in table 1.

Table 1. Performance of two-wheeled Seg robot folded from laser-cut 0.010" (0.25mm) PET film. *The same design made from 0.005" (0.13mm) PET film weighs 36g, while a paper version weighs 31g. Other metrics remain unchanged.

Approximate design time	1	hr
Approximate fabrication time	20	min
Approximate cost	20.00	USD
Weight	42	g*
Maximum speed	23	cm/s
Turning radius (both wheels driven)	0	cm
Turning radius (one wheel driven)	4	cm

The design environment is also able to auto-generate autonomous driving code for this robot. Additional electronic components comprising an LED and a photosensor (each an integrated derived components containing both a pure electrical subcomponent and a firmware driver) can be added to the design. Their data outputs can be wired through data manipulation blocks into the data input ports of the integrated motor component and replace the previous UI elements. Since there is no UI, app code is no longer generated; instead the on-board microcontroller runs autogenerated code to autonomously drive the robot in a line-following pattern. The resulting system is shown in Fig. 1.

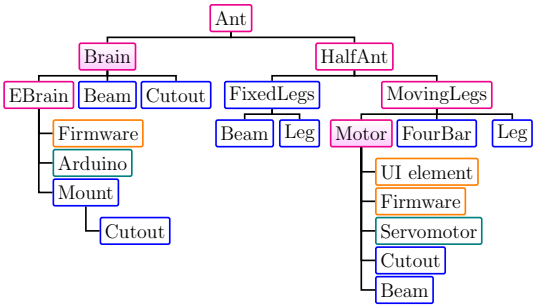


Fig. 11. The design of the walking robot is similar to that of the Seg, with the addition of mechanical leg and flexure components. The higher-level brain and motor components, shaded in the diagram, can be reused from the earlier design.

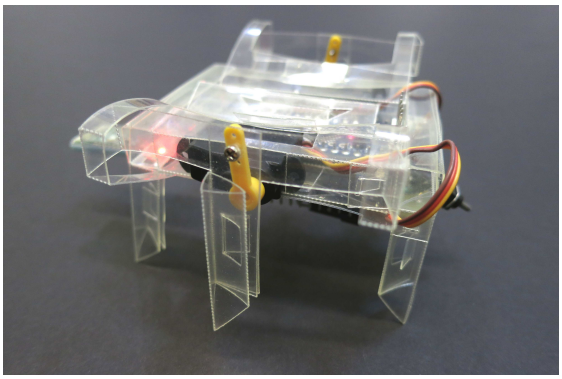


Fig. 12. A complex hexapod walker can be generated adapting existing library elements generated from past designs.

6.2 Hexapod Walker

An insect-like legged robot can be created using compliant joints to add kinematic degrees of freedom for a more complex design. A stationary base is formed from four non-moving legs, while two other legs are circularly actuated by drive motors to provide a walking gait. The moving legs remain parallel and are constrained to move in a plane by flexural four-bar linkages. The design of this robot was adapted from the earlier Seg design, with many components directly taken from that. This was enabled by the modular design paradigm, greatly simplifying and speeding up the creation of the hexapod.

An information flow similar to the wheeled robot above defines the robot design, with an additional mechanical component defining a four bar linkage translating the circular mechanical output of the motor shaft into the walking gait of the moving legs. The component hierarchy can be seen in Fig. 11, and the resulting structure can be seen in Fig. 12.

6.3 Grasping Arm

A markedly different robotic configuration is created for the multi-segment manipulator arm shown in

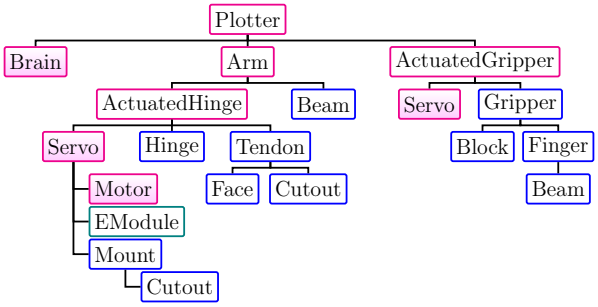


Fig. 13. The design tree for the gripper arm shows how a complex electromechanical device can be hierarchically assembled from simpler mechanisms. The integrated brain and servo modules are adapted from the earlier robots with slight modifications to enable daisy chained electronic modules, and the servo module is shared between the hinge and gripper mechanisms.

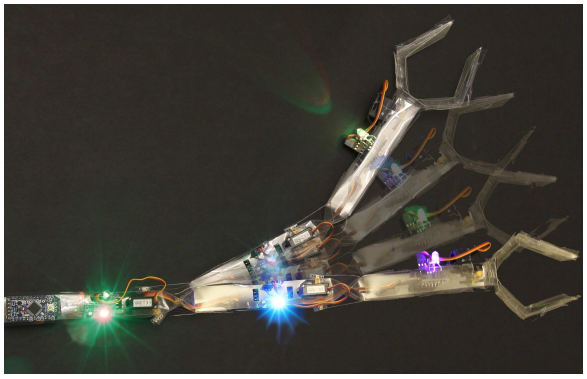


Fig. 14. A robotic manipulator arm was generated by serially connecting integrated actuated hinge and gripper modules.

Fig. 14. In this robot, an actuated gripper is positioned by a sequence of actuated hinge joints. The design tree, shown in Fig. 13, illustrates the hierarchical composition; for example, the end effector itself is an integrated electromechanical mechanism included in the higher level assembly. This robot also employs the electrical hardware modules, such that each actuated hinge and gripper module contains an independent integrated mechanical structure, actuator, drive circuit, and control logic. The plug-and-play electrical modules enable a distributed electrical system along the arm.

The designed arm automatically generated a smartphone UI to allow immediate human control. Some performance metrics of the fabricated robot are presented in table 2.

7 Conclusions and Future Work

The system presented in this paper implements a unified design environment allowing users to create robotic mechanisms from a library of integrated mechanical, electrical, and software components. The building blocks can be created by expert designers, al-

Table 2. Performance of an arm folded from laser-cut 0.010" (0.25mm) PET film.

Approximate design time	1	hr
Approximate fabrication time	30	min
Approximate cost	27.00	USD
Weight	60	g
Maximum joint angle (actuated)	$\pm$35	deg
Maximum joint angle (mechanism)	$\pm$110	deg
Gripper Strength (on 1.5 cm object)	100	mN

lowing casual users to quickly and easily create custom programmed electromechanical mechanisms. The value of this paradigm is demonstrated by the various robots created using the system presented above. In a matter of hours, the high-level structural specification of a desired device was able to be realized into automatically generated fabrication files, control software, and user interfaces, creating immediately usable robots complete with driver interfaces and autonomous behavior. This system brings into reach the goal of a complete robot compiler to incorporate custom robotics into the domain of personal on-demand use.

This work demonstrates an infrastructure for automated robot design, opening up a large body of future research to extend and enhance the system. The next steps can focus on assisting a user with design decisions. A recommendation engine can analyze existing components to suggest possible connections or components to add to a design in progress. The system can also help closing the design loop by incorporating behavioral analysis. Component definitions can include a composable model of their kinematics and dynamics, generating outputs suitable for simulations and analytic characterizations. The complete behavior of a design can be verified and validated against the functional requirements of the user.

In the long run, an independent design loop can iterate through automatically generated robot compositions, analyzing and updating the design based on the characterization output, thus leading to an intelligent compiler that can autonomously generate a custom robot design based on a high level task description.

Acknowledgments

This work was funded in part by NSF grants 1240383 and 1138967 and NSF Graduate Research Fellowship 1122374, for which the authors express thanks.

References

[1] Mehta, A. M., et al., 2013. "A scripted printable quadrotor: Rapid design and fabrication of a folded MAV". In 16th International Symposium on Robotics Research.

[2] Mehta, A. M., and Rus, D., 2014. "An end-to-end system for designing mechanical structures for print-and-fold robots". In Robotics and Automation (ICRA).

[3] Mehta, A. M., DelPreto, J., Shaya, B., and Rus, D., 2014 (to appear). "Cogeneration of mechanical, electrical, and software designs for printable robots from structural specifications". In Intelligent Robots and Systems (IROS).

[4] Mavroidis, C., DeLaurentis, K. J., Won, J., and Alam, M., 2001. "Fabrication of non-assembly mechanisms and robotic systems using rapid prototyping". *Journal of Mechanical Design,* **123**(4), pp. 516–524.

[5] Richter, C., and Lipson, H., 2011. "Untethered hovering flapping flight of a 3d-printed mechanical insect". *Artificial life,* **17**(2), pp. 73–86.

[6] Rossiter, J., Walters, P., and Stoimenov, B., 2009. "Printing 3d dielectric elastomer actuators for soft robotics". In SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring, International Society for Optics and Photonics, pp. 72870H–72870H.

[7] Hoover, A. M., and Fearing, R. S., 2008. "Fast scale prototyping for folded millirobots". In Robotics and Automation (ICRA), 2008., IEEE, pp. 886–892.

[8] Liu, Y., Boyles, J., Genzer, J., and Dickey, M., 2012. "Self-folding of polymer sheets using local light absorption". *Soft Matter,* **8**, pp. 1764–1769.

[9] Shimoyama, I., Miura, H., Suzuki, K., and Ezura, Y., 1993. "Insect-like microrobots with external skeletons". *Control Systems, IEEE,* **13**(1), pp. 37–41.

[10] Brittain, S., et al., 2001. "Microorigami: Fabrication of small, three-dimensional, metallic structures". *Journal of Physical Chemistry B,* **105**(2), pp. 347–350.

[11] Hawkes, E., et al., 2010. "Programmable matter by folding". *Proceedings of the National Academy of Sciences,* **107**(28), pp. 12441–12445.

[12] Tolley, M., Felton, S., Miyashita, S., Xu, L., Shin, B., Zhou, M., Rus, D., and Wood, R., 2013. "Self-folding shape memory laminates for automated fabrication". In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE.

[13] Onal, C. D., Wood, R. J., and Rus, D., 2011. "Towards printable robotics: Origami-inspired planar fabrication of three-dimensional mechanisms". In Robotics and Automation (ICRA), IEEE, pp. 4608–4613.

[14] Birkmeyer, P., Peterson, K., and Fearing, R. S., 2009. "Dash: A dynamic 16g hexapedal robot". In Intelligent Robots and Systems (IROS), IEEE, pp. 2683–2689.

[15] Onal, C., Wood, R., and Rus, D., 2013. "An origami-inspired approach to worm robots".

Mechatronics, IEEE/ASME Transactions on, **18**(2), April, pp. 430–438.

[16] Demaine, E. D., and Tachi, T., 2009. Origamizer: A practical algorithm for folding any polyhedron.

[17] Lang, R., 2012. *Origami design secrets : mathematical methods for an ancient art*. A K Peters/CRC Press.

[18] Pepakura designer. `http://www.tamasoft.co.jp/pepakura-en/`. [Online; accessed 26-May-2014].

[19] Parnas, D. L., 1972. "On the criteria to be used in decomposing systems into modules". *Commun. ACM*, **15**(12), Dec., pp. 1053–1058.

[20] Farritor, S., and Dubowsky, S., 2001. "On modular design of field robotic systems". *Autonomous Robots*, **10**(1), pp. 57–65.

[21] Hornby, G., Lipson, H., and Pollack, J., 2003. "Generative representations for the automated design of modular physical robots". *Robotics and Automation, IEEE Transactions on*, **19**(4), Aug, pp. 703–719.

[22] Davey, J., Kwok, N., and Yim, M., 2012. "Emulating self-reconfigurable robots-design of the smores system". In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE, pp. 4464–4469.

[23] LEGO Mindstorms. `http://mindstorms.lego.com`. [Online; accessed 01-Nov-2014].

[24] MOSS. Modular Robotics. `http://www.modrobotics.com/moss`. [Online; accessed 01-Nov-2014].

[25] VEX Robotics. `http://www.vexrobotics.com`. [Online; accessed 01-Nov-2014].

[26] Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J., and Asanović, K., 2012. "Chisel: constructing hardware in a scala embedded language". In Proceedings of the 49th Annual Design Automation Conference, ACM, pp. 1216–1225.

[27] OpenSCAD. The programmers solid 3D CAD modeller. `http://www.openscad.org`. [Online; accessed 01-Nov-2014].

[28] Freese, M., Singh, S., Ozaki, F., and Matsuhira, N., 2010. "Virtual robot experimentation platform v-rep: A versatile 3d robot simulator". In *Simulation, Modeling, and Programming for Autonomous Robots*, N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, and O. von Stryk, eds., Vol. 6472 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 51–62.

[29] YAML. `http://www.yaml.org/`. [Online; accessed 01-Nov-2014].

[30] Mehta, A. M., Bezzo, N., An, B., Gebhard, P., Kumar, V., Lee, I., and Rus, D., 2014 (to appear). "A design environment for the rapid specification and fabrication of printable robots". In International Symposium on Experimental Robotics (ISER).