Making Machines that Make: Object-Oriented Hardware Meets Object-Oriented Software

by

Nadya Peek

MS, Media Arts and Sciences, MIT (2010)
BSc, Artificial Intelligence, University of Amsterdam (2008)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

Author .....
Program in Media Arts and Sciences
July 25th, 2016

Certified by .....
Neil Gershenfeld
Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by .....
Pattie Maes
Academic Head
Program in Media Arts and Sciences

Making Machines that Make: Object-Oriented Hardware Meets Object-Oriented Software

by

Nadya Peek

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on July 25th, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Media Arts and Sciences

Abstract

Rapid prototyping has been in the limelight for the past decade. 3D printers have an evocative name that promises production of complex parts on demand. Yet current practice doesn't quite deliver on these promises of advanced manufacturing. Existing digital fabrication tools enable repeatability and precision by using codes to describe machine actions. But the infrastructure used for digital fabrication machines is difficult to extend, modify, and customise. It is very difficult for the end-user to incorporate more forms of control into the workflow. Machine design today is largely the same as it was 50 years ago, despite decades of progress in other fields such as computer science and network engineering.

I argue that we need to transition from rapid prototyping to *rapid prototyping of rapid prototyping*. To make diverse goods, we need diverse tools. To develop diversity in digital fabrication tools, we need reconfigurable and extensible infrastructure for machine building.

Using insights from object-oriented programming, end-to-end principles in network design, and the open system interconnection model, I propose a new paradigm for machine building called *object-oriented hardware*. In this paradigm, software objects and hardware objects are peers that have procedures, methods, ports, and presentations. Machine building modules are available as software libraries are to programmers. A machine instantiation is an assembly of objects situated in a particular context.

Using this approach, a thing *together with* the machine that makes it becomes an application. This method transcends the additive versus subtractive manufacturing comparisons by considering both types of rapid automation. Development work is di-

vided into infrastructural engineering, which develop modules for use in any machine, and application development, which develop specific machine instantiations.

Here I present technical implementations of machine building infrastructure first. These include distributed networked controls, reconfigurable software interfaces, and modular mechanical machine components. Then I present machine instantiations that use this infrastructure to demonstrate its capability. Finally to evaluate the object-oriented hardware paradigm in the wild, I observe machine building novices using these tools in both a workshop format and in the Fab Lab network for machine building. To make the modular components for machine building accessible in this context, I developed an extensible toolkit for machine building—the Cardboard Machine Kit. Using this toolkit, novices were able to make a wide range of machines, demonstrating the power of this method.

Thesis Supervisor: Neil Gershenfeld
Title: Professor of Media Arts and Sciences, Program in Media Arts and Sciences

Making Machines that Make: Object-Oriented Hardware Meets Object-Oriented Software

by

Nadya Peek

The following people served as readers for this thesis:

Thesis Reader .....

Erik Demaine
Professor of Computer Science and Engineering
MIT Department of Electrical Engineering and Computer Science

Making Machines that Make: Object-Oriented Hardware Meets Object-Oriented Software

by

Nadya Peek

The following people served as readers for this thesis:

Thesis Reader .....

Jennifer Lewis
Wyss Professor of Biologically Inspired Engineering
Harvard School of Engineering and Applied Sciences

Acknowledgements

I'd like to thank my advisor, Neil Gershenfeld.
He has given me knowledge, time, space, courage, and motivation.
His role as my advisor is irrevocably appointed with tenure for life.
I'd like to thank my thesis readers, Erik Demaine and Jennifer Lewis.
Their research has given me shoulders to stand on and shown me sky to reach for.
I'd like to thank my collaborators James Coleman, Ilan Moyer, and Jonathan Ward.
They have given me a beautiful history to look back on and now give me a lifetime of projects, discussions, manifestos, machines, and friendship to look forward to.
I'd like to thank all of the Center for Bits and Atoms as well as its affliates.
In particular, I'd like to name Ara Knaian, Kenny Cheung, Kerry Lynn, Gonzalo Rey, Bunnie Huang, Bill Young, Max Lobovsky, Matt Likens, and Breht O'Hearn.
I'd like to thank all of the Fab Lab network, and the Fab Academy students,
especially Sherry Lassiter and Tomas Diez.
I'd like to thank the *How to make (almost) anything* students, and the *How to make something that makes (almost) anything* students.
I'd like to thank the Media Arts and Sciences Academic Program,
especially Linda Peterson.
I'd like to thank 41W (for never talking to the press).
I'd like to thank my friends, and my family.

I have truly found paradise.

Nadya Peek

Colophon

This dissertation was written in Cambridge, MA, and Deer Isle, ME.
It was written in gvim on a Lenovo Thinkpad W530 running Ubuntu 14.04.
It was typeset in *Computer Modern* using pdflatex and the MIT thesis template.
It uses the hyperref package for links, the pdfpages package for adding pages to the appendix, the minted package for typesetting Python, C, and JavaScript code, and the clrs package for pseudocode.
James Coleman staged many graphical interventions; I owe him many thanks for my dope figures.
In-image text was typeset in *Info Display Bold* and *Akkurat Bold*.

Code and design files are maintained at
`http://mtm.cba.mit.edu`.
A digital copy of this document is available at
`http://infosyncratic.nl/peek-making-machines.pdf`.

Contents

List of Figures

List of Tables

Chapter 1

Introduction: Rapid Prototyping of Rapid Prototyping Machines

Personal fabrication sounds like something one might buy in the future. *Get your personal fabricator, available at any store that sells personal computers!* It certainly evokes the convenience promised by Star Trek's 'Replicator'. Being a digital fabrication researcher at a well-supplied lab, I already inhabit part of that future. I can theoretically already make *almost anything* on existing industrial machines [26]. Parts of digital fabrication research are trying to distribute that future more evenly—a future where *anyone* can make almost anything. In that future we have faster, cheaper, better tools. We have better curricula for educating a society of makers.

That future sounds pretty good. But I think we need something more.

Since starting my graduate work, I have set up many Fab Labs, such as the one depicted in Figure 1-1, in five different continents. I have taught in more than 50 Fab Labs. I have set up labs as part of a tribal council in Alaska, at NASA, in farm schools in rural India, in navy shipyards, at amputee clinics, in midwestern community colleges, in Saudi Arabian corporate visitor centres, in design schools,

in libraries, in museums, and also pop-up labs at the UN General Assembly, the White House, Capitol Hill, and the World Economic Forum. I have seen all kinds of people interact with all kinds of digital fabrication machines and software, from young children to Nobel laureates. I speak from extensive experience working with both many kinds of machines and many kinds of people when I say that our current digital fabrication technology does not encourage wide participation.

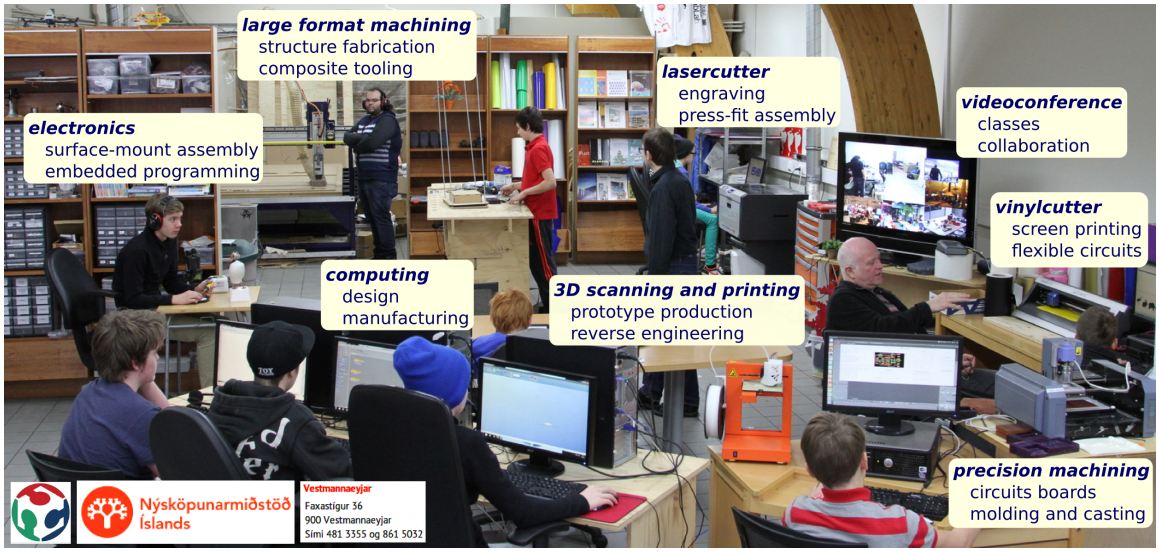


Figure 1-1: A Fab Lab in Vestmannaeyjar, Iceland. Featuring the fab lab standard inventory of milling machines, electronic workstations, laser cutters, 3D printers, and more. Image courtesy Frosti Gislason.

If anyone should be able to make almost anything, who makes the tools for them to do that, and why? If we want everyone to have access to fabrication, why don't we expect the variety of tools to reflect the variety of applications from that variety of people? Does making tools accessible just mean making tools we are already familiar with faster, cheaper, and easier to use?

I believe the future holds a whole new class of machines and tools that are designed, modified, and reconfigured by those who use them.

There are two perspectives—two movements that have shaped the research I present

here. One is 'classic' engineering and technology. MIT is an ideal stage for observing this world view—a place destined to develop technology that can change the world. The other is the 'maker movement', the excitement over which has been propelling experts and novices alike [2]. Fab Labs, hackerspaces, and maker faires provide a foundation for broad-based engagement with technology, digital fabrication, and science show-and-tell.

Both of these perspectives focus attention and energy; both have tangible outcomes. They give us self-driving cars, solar energy, digital literacy, educational outcomes. Both are at times also problematic—exclusive, political, erroneous, gendered, commercial, hyped [21; 4]. In my work, I see problems with both 'engineering' and 'making'. I don't believe simple technological interventions alone can provide solutions to complex social issues. I don't believe that a maker revolution has democratised technology such that anyone can come up with their own solutions. However, I'm optimistic about what comes after technosolutionism [46].

A driving force for digital fabrication research is developing the ability to manufacture quantities of one without high cost or loss of precision and complexity. I believe that to make this happen, we need to make *the making of tools* accessible at a personal level. We need accessible ways to make machines. This does not only include making machine building technologies, but also includes developing skilled communities of practice. Involving people in developing the technology they use does not only transform the technology, but transforms the people as well. This inclusion is not currently a common social practice; our technological infrastructures poorly support it.

It is worthwhile to consider the historical context of the digital fabrication equipment we know today. The first computer-numerically controlled (CNC) milling machines were developed so that the highly precise parts that were being produced during the Cold War could be made more quickly and reliably. CNC mills removed a precarious

reliance on skilled workers to produce precise parts. By making a machine capable of producing the kinds of parts that previously only highly skilled and experienced machinists could make, managers were able to shift costs from the wages of skilled workers (who might leave) to investments in machines that would run around the clock. Machines were developed to increase technical capacity, but also as a political tool to restructure labour relations [59; 63].

Thus the machine became the most important asset for manufacturing, not the humans who ran them. The original user model for digital fabrication assumed the machines would be operated by an insubordinate workforce with no interest in improving the technology they were being replaced with. The tool-maker was separated from the tool-user.

Since the 1950s, little has changed in this model. For example, the purchase of a wire EDM (electric discharge machining) in our lab came with several days of training by the vendor's technicians, who had a tough time explaining when to use the keypad, the mobile keypad, the keyboard, the touchscreen, or the extra buttons that for some reason are on the side. The machine only takes certain types of .dxf files (a specific file type for specifying geometry, used here for toolpaths) with filenames that are shorter than eight characters. Despite these arbitrary usability limitations, the height of the workbase of the EDM is guaranteed to be within 30μm at all points, which demonstrates the extremely high precision of this tool. The expectation was that there would be an expert who would be in charge of making the toolpaths for the EDM and a machinist who would be in charge of running the EDM, and that they would get used to the quirks of the machine. The machine is not designed for a group of casual users who also use all the other digital fabrication equipment we have in our lab. These digital fabrication machines are designed to have dedicated staff, and those staff are not expected (or able) to modify the machines.

1.1 If it's cheaper, more people will buy it

Digital fabrication machines are not only hard to use. They are also inaccessible because of their cost. Many of our lasers, EDMs, or waterjets cost hundreds of thousands of dollars. How could people without big research lab budgets still gain access to the precision that digital fabrication affords? The maker movement has primarily focused attention on making digital fabrication equipment more accessible by making it cheaper.

For additive manufacturing Jones et al. [39] developed a Replicating Rapid Prototyper, or RepRap, to help grow the number of 3D printers in the wild in 2006. Many of the RepRap parts can be printed inexpensively on a RepRap, which not only helped the exponential growth of RepRaps, but also grew the number of design iterations by the printer's open source community [17].

Malone et al. developed the Fab@Home, another open source 3D printer, at around the same time [51]; it was not self-replicating, but it did provide users with design files and bills of materials for making their own instantiations. The MIT Center for Bits and Atoms class *How to make something that makes (almost) anything*[1] provided another rich space to develop personal fabrication machines (not only limited to 3D printing) such as the *MTM Multifab* by Ilan Moyer and Max Lobovsky in 2009 or a small format milling machine Jonathan Ward and I developed out of HDPE kitchen cutting boards called the *MTM Snap* in 2011 (see Figure 1-2). But despite freely available design files, sourcing the right parts and assembling the machines was still a high barrier to entry. Cost is only one factor in accessibility, and these machines were still fairly inaccessible in terms of expertise required.

Commercially available 3D printer kits inspired by these early collaborative designs, such as the *Makerbot Thing-o-matic* or *Ultimaker 1*, were released soon after the

[1] *MAS.865*, documented at `http://mtm.cba.mit.edu`.

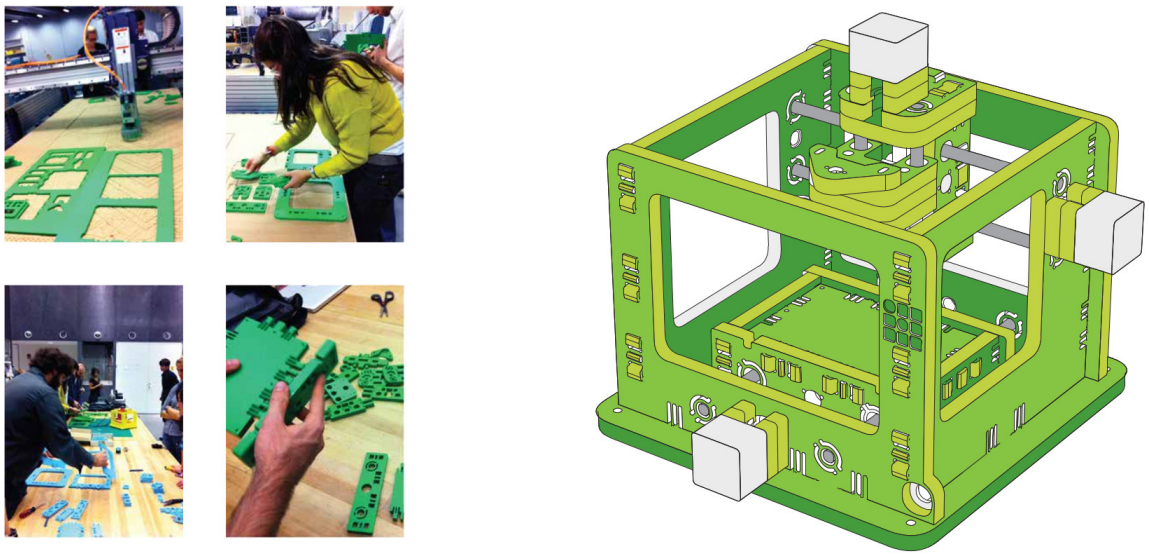


Figure 1-2: The MTM-Snap: A milling machine that can be made on another milling machine, such as the ShopBot (shown left). The ShopBot is included in the standard Fab Lab inventory `fab.cba.mit.edu/about/fab/inv.html`, accessed Feb 2016. The MTM Snap designs are available at `http://mtm.cba.mit.edu/machines/mtm_snap-lock/index.html`, accessed Feb 2016.

first open designs. This mitigated the issues with part sourcing and also created a commercial interest in developing clear assembly instructions. At a Maker Faire in 2012, O'Reilly editor Shawn Wallace counted a total of fifty-five distinct 3D printer designs being presented [86], many of them for sale as kits. At the same time, existing fabrication companies such as ShopBot Tools (whose tools we used to bootstrap the *MTM Snap*, see Figure 1-3), Epilog Laser, and Roland Digital released lower cost, smaller format machines such as the *ShopBot Desktop*, the *Epilog Zing*, and the *iModela*. Again, accessibility here was mostly addressed through cost.

The intent of the *MTM Snap* project was to produce a low-cost, modifiable milling machine design that anyone with access to a Fab Lab could make. We failed at that goal. We made a low-cost, modifiable milling machine *we* could make. It was too hard for someone without prior experience in electrical and mechanical engineering to source the parts and build the machine, and for people with sufficient experience

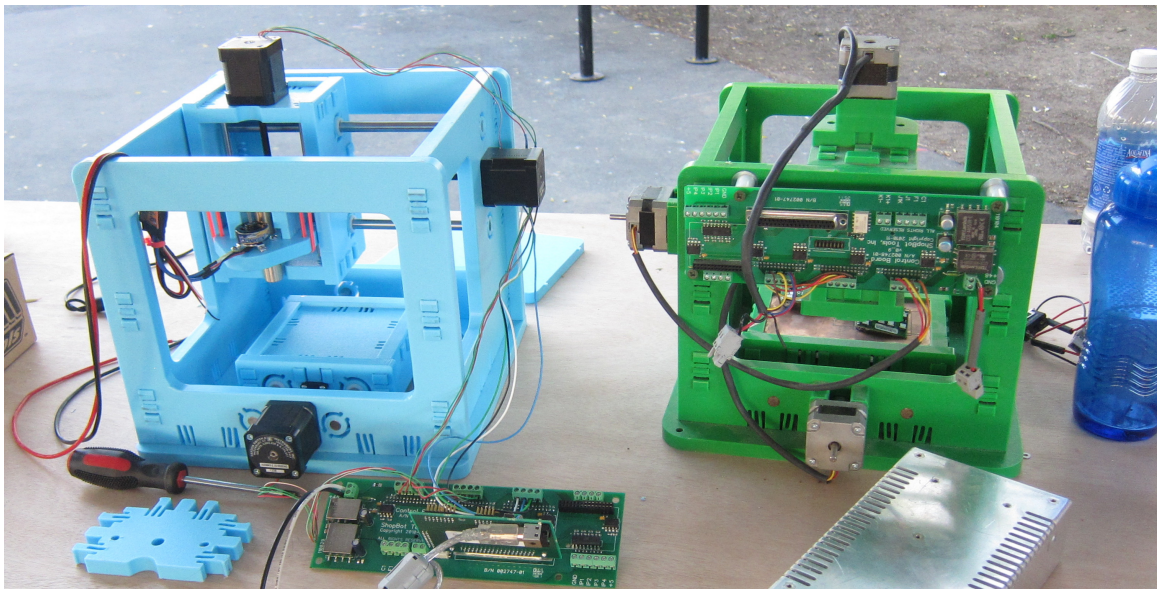


Figure 1-3: Two versions of the *MTM Snap* Jonathan Ward and I demoed at the World Maker Faire with ShopBot in 2012.

it was often easier (and likely more satisfying) to develop machines made with locally available hardware, such as the *Black Mamba* developed at MISiS by Dmitry Sukhotskii, Vladimir Kuznetsov, et al.[2] [79]. The *MTM Snap* was inaccessible in that it lacked both documentation and user-friendliness[3] and did not make up for it in the fabrication capacity it offered. After all that work, it was still just a simple circuit board mill with a work volume of under $15 \times 15 \times 10$ cm. Nonetheless, many of core design principles of the *MTM Snap* were reproduced later in others' machines. For example, the snap construction method of HDPE was reused widely in other machine designs, as was the control circuit.

[2]The *Black Mamba* was partially based on the *MTM Snap* with great enthusiasm for the open source design, but none of the design files were actually reused. The *MTM Snap* design was not sufficiently universal to be able to accommodate the very different markets and parts that our Russian counterparts had access to. Nonetheless, it was an object that we could base our cross-atlantic collaboration on.

[3]Our method for supplying toolpaths involved downloading C and Python source code plus a bunch of dependencies, compiling the control software, and running it with a custom-flashed Arduino, with an added breakout board for motor control you had to solder together yourself. Once you got it running, the machine could make more of the custom breakout boards, but as far as we know no users besides us, the designers, got to this point.

Meanwhile, the commercial success of maker-oriented digital fabrication machines began to rely heavily on their usability and reliability—factors in accessibility other than cost. Makerbot Industries and Ultimaker released plug-and-play 3D printer models instead of build-it-yourself kits. Features such as auto-calibration and one-button interfaces became commonplace. By 2013, the personal fabrication machine space was no longer occupied predominately by the self-produced *RepRaps* or *Lasersaurs*, but by machines that were commercially available. This commercial market has only grown, with, for example, the preorders for the personal-use-oriented laser cutter *Glowforge* exceeding 28 million USD in late 2015 [76]. By now, the personal fabrication market is a serious contender for machine development, alongside the previously dominating industrial fabrication markets.

1.2 Versatile, yet easy

User interface and user experience work being done on commercially available machines such as Glowforge's interactive laser cutter is a very good thing; I hope one day I never have to use a buggy 32-bit Windows driver to run a machine again. I applaud machines getting increasingly faster, cheaper, and more precise [29]. However, the most user-friendly of interfaces still contain—or even design in—many limitations.

As Anthony Dunne writes about electronics: "*User-friendliness helps naturalize electronic objects and the values they embody. For example, while electronic objects are being used, their use is constrained by the simple generalized model of a user these objects are designed around: the more time we spend using them, the more time we spend as a caricature.*" [19]

The interfaces to digital fabrication are still built on a user model where the toolmaker has been separated from the tool-user. Therefore, usability of those interfaces is based on the assumption that the people who are going to use the machines are not

able to or interested in modifying them—n00bs. This user model has been applied to both kit machines and commercial machines. A machine kit provides convenience of a pre-specified bill of materials at the cost of versatility. Commercially successful machines now provide *user experience*, but at the expense of modifiability. These machine types are predicated upon the notion that as a user of a machine you are not expected to reconfigure it.

In a famous Xerox PARC video now known as "When User Hits Machine", anthropologist Lucy Suchman recorded the almost-slapstick performance of two prominent researchers trying to use the two-sided paper functionality of a photocopier [77]. The complexity of Xerox photocopiers was meant to be a selling point for the machines. But the machines Suchman et al. analysed were of the first batch that were marketed not to a dedicated user (i.e. a technician) but to a casual user. Customers reported that the machines were unusably complicated. While the marketing of Xerox at the time promised the ease of a big green button, the reality was that to use the machine, the user inevitably needed to learn more about it.

Suchman points out that the interface is a prescriptive representation—a *plan* of use. These interfaces, instructions, or recipes might be made in one place, such as the design laboratories of Xerox. But they anticipate use in a different context—a photocopier in an office of casual users. How that *situated action* of use unfolds needs to inform the design of meaningful interactive interfaces as much as their assumed user models. One anticipates use as the other enacts, and through iterations of planning and observing we can design powerful and meaningful human-machine interfaces.

Contemporary interactive interface work for digital fabrication machines is still comparatively superficial—we are at the stage of big green buttons. Usability work for digital fabrication machines right now means developing more and more convincing *plans*. For example, the user/machine interaction is very well-scripted and prescribed for printing 'demo prints' on 3D printers. Printing 3D models that have been opti-

mised for the printer leads the machine user through an ideal experience. But the difficulty of optimising the orientation, temperature settings, material support selections, and so on for a new 3D object can make the initial demo part print experience seem like a false promise. The language to describe personal fabrication sounds a lot like the marketing language Xerox also used in the 1980s to describe the office of the future. But the user inevitably needs to learn more about the 3D printer.

Even if all settings could be optimised automatically, prescriptive use limitations of digital fabrication machines remain. Even if ease of use, speed, and cost are taken to their limits for digital fabrication, machine users are still being cast into the separate roles of part designer, drafter, toolpath planner, and machine operator. The separation between these roles made sense historically; a machine operator needed different expertise and tools than a drafter. These limitations are no longer relevant, yet the tools used in each role are still mostly separate. An example is the distinction between CAD software and CAM software.

Translating from tool to tool can seem like a bad party-game of telephone where progressively more design intent is lost [15; 48]. These issues run deeper than the user interface—they extend to the very structure of information passed from tool to tool. CAD and CAM tools still rely on file formats, such as g-code or .stl, that have not changed much since their inception in the 50s and 70s, respectively.

G-code is only a format for sending position coordinates to a milling machine, along with a few more commands like spindle speed, coolant flow, etc. [73]. Any acceleration, deceleration, or control flags need to be implemented at the g-code interpreter level, fragmenting control into subsystems that are hard for end-users to modify. The interpreter is the computer (or microcontroller) connected to motor controllers; the user computer streams g-codes to the interpreter which are then executed. The user computer does not keep track of where the interpreter is in execution, which means that the user interface does not always know exactly where the machine is. As a

result, it is very difficult for the end-user to incorporate more forms of control into the system. For example, if a user had issues with material discolouring in a 3D printer due to excess heat, and wanted to increase the flow rate of the extruder when scorching can be visually measured, that modification would have to be implemented in the g-code interpreter firmware. Originally, this separation was necessary because the bandwidth between the user computer and the interpreter was very limited. This is no longer the case.

For 3D printers, designs can be created with any CAD software but typically will be exported as STereoLithography (.stl) files and 'sliced' by a machine-specific CAM tool to create tool paths. On 3D design sharing websites such as Stratasys' *Thingiverse*, the most commonly shared 3D print files are therefore STL files. STL files give a boundary representation of an object by specifying a list of triangles with their surface normals. Typically this means that the original design file which may contain curves for specifying volumes needs to be approximated with triangular facets. A high resolution approximation of complex curved surfaces might result in millions of triangles and gigabytes of data even if the original design file was much simpler. Unlike the original CAD source files, much of the design intent is lost with STLs (for example, a sphere saved as an STL will have no attribute such as radius). Many 3D designs are shared online with permissive licenses for modification and reproduction [53], yet the file formats don't carry with them the original design intent [48].

G-code and STL files are only two examples of the limitations of file formats and software workflows of digital fabrication. Some efforts have been made to create successor to the STL file format called Additive Manufacturing File format (AMF or 3MF [49]), although it has yet to be widely adopted and still has many of the same issues as STL. Functional representation CAD software such as Antimony [40] uses math strings to describe objects, which is often both a more efficient and precise method than boundary representations. However, to create machine instructions,

the design files still need to be exported as STLs. Robotic arm users are starting to develop alternate open control methods such as Kuka Parametric Robot Control [10], but they are still architecture specific. In short, while there are many individual laudable efforts for improving parts of the digital fabrication software pipeline, the translation from one system to another is rarely routine.

I will not dwell much more on the implementation details of existing software or machine tools in the remainder of this document. However, my research is in direct response to the limitations I have encountered with the file formats, software tools, and digital fabrication machines I've dealt with. While I have enumerated separate roles—designer, drafter, toolpath planner, machinist—I do not believe they are relevant any longer. I believe we get new capacity for action as we develop new configurations of humans, tools, and machines. Here I draw from Suchman's analysis of reconfiguration between humans and machines [77] for understanding where to develop that capacity, and from theories of object-oriented programming and end-to-end principles in infrastructure design presented by Sutherland, Goldberg, Saltzer, et al. [81; 30; 71] to understand how to. To make progress, I believe we need more than an iterative or evolutionary refinement of existing practices. It is not enough to just add some XML markup to STLs. We need reconfigurable and extensible technology.

1.3 Personal Computers/Personal Fabricators

The trajectory of personal fabrication is often compared to the trajectory of personal computing. The first digital computer, ENIAC (on the left in Figure 1-4), was developed in 1946 as part of a wartime effort to calculate artillery firing tables. By the 1970s the technology was redirected to personal computing, making computers available to casual users at fraction of the cost of ENIAC (and at a fraction of the footprint). In 1973 the Xerox Alto Personal Computer released many features that

Figure 1-4: Left: The ENIAC, one of the first digital computers, developed in 1946. Right: the Xerox Alto, one of the first personal computers, released in 1973.

are now standard: WYSIWYG editors, graphical interfaces, and pointers (see Figure 1-4). The technology that was only available to governments fighting wars some twenty-five years prior had been adopted, reduced to practice, and improved with usability features.

There are many commonalities between personal computing and personal digital fabrication. The first personal computers were originally available as kits, just like the first desktop 3D printers. The price of a personal computer dropped over time despite its speed going up, as with 3D printers. However, the analogy does not hold up quite as well for other details. A calculation done on a mainframe computer will produce the same answer as the same calculation done on a personal computer—on a personal computer it may just take much longer. But the parts produced by desktop 3D printers cannot be made to the same tolerances as the parts that were produced on the first CNC mills. Nor are they made with materials of similar quality. Digital fabrication machines shrank to fit on our desktops and budgets but also sacrificed precision and accuracy in the process. PCs might be universal computing devices, but no fabrication machine is a universal fabrication device.

Personal computers provided a platform for users to develop software that would run on those personal computers. The personal computer itself was a tool that aided in

the improvement of the personal computer. This allowed a participatory and fast innovation process to take place in personal computing. The continued development of personal fabrication tools seems to have instead given rise to more commercial entities who are not interested in developing or sharing standards [53]. In comparison to the modularity and standards that were crucial to the development of the PC,[4] digital fabrication toolchains are rather arbitrary. Their interoperability often seems to depend on who their parent company is—the Autodesk/HSMworks/Eagle alliance? Or the Solidworks/Catia/Altium camp?—instead of any over-arching standard.

So while digital computing may have been repurposed into personal computing in a few decades, I argue that we have a long way to go for personal fabrication. I have participated in helping spread access to digital fabrication through Fab Labs and certainly believe there is value in continuing to do so. However, I don't believe that just developing cheaper and user-friendlier machines that adhere to the original CNC user model (including machines such as the *MTM Snap*) will lead to widespread access to the precision and accuracy of industrial digital fabrication. The current range of applications is only a narrow slice of what is possible. Each user's application varies slightly. Each problem space needs a specific approach. I argue that for all of these problems, people need the tools to build their own solutions.

1.4 Universality

In current machine building practice there is a focus on *universality* of machines (such as the *Universal Desktop Fabricators* in [85; 47]), a one-machine-fits-all solution. This is the model we are familiar with from science fiction: all rooms are outfitted with the same replicator. Replicators are easy to use, yet very powerful. Replicators can

[4]The use of 'upgrade' as a intransitive verb started in 1950, whereas 'update' had its first recorded use in 1944—both during the early days of computers.

make simple things like tea, or complex things like clarinets or engine parts, all with the same user interface. The replicator makes assumptions about what you want by inferring exactly what you mean when you say "Tea, Earl Grey, hot". However, if we take an existing tool such as a sewing machine as an example, we can see a wide variety of machines that differ slightly depending on the intended user. Industrial sewing machines can sew very quickly, but might offer only a few stitch options. Home sewing machines might be more tailored to quilting or tailoring. Children's sewing machines are smaller and outfitted with specific safety features. Sewing machines are a well established technology that have identified many different types of users. The machines are made for those users. Is personal fabrication going to be any different? Will an aerospace engineer need the same capabilities in their fabrication equipment as a circuit designer?

There are too many niches of possible users of automation and digital fabrication to be able to cater to them all with a universal tool. There are problems in principle with the notion of universality for tools. Meanwhile, variations of 3D printers are entering the market at a dizzying pace, but they offer very little in terms of after-market customisability or extensibility. Automation tools for custom processes need domain expertise to configure and use, are expensive, and don't work that well. These are not problems in principle, but problems in practice. So how can we build the practice of rapid prototyping of rapid prototyping machines?

1.5 A Playground for Building Machines

There is something fundamentally wrong with the infrastructure we are using to build machines. Instead of waiting for the ultimate dream team to develop the perfect machine that is appropriate for all, why are we not creating the infrastructure for everyone to configure personal automation themselves? In researching universal

fabrication machines, we make the rather strong assumption that one machine could *be* capable of fabricating all products. While it is possible to pair both the fabrication of large objects and small features in a single tool, that makes the resulting machine high in cost. Using a less precise large format machine or a highly precise but small format machine might be preferable for producing the majority of products. The obverse is perhaps more concerning: if the tools we have are incapable of producing the good we want, should we then be expected make concessions on the goods we produce? Why are we focussed on rapid prototyping of *things*, and not on rapid prototyping of rapid prototyping machines?

It is easy to overlook the importance of accessibility while working at an institution like MIT. My peers and I are already empowered to make machines. We have easy access to expert knowledge and fancy tools. But I'm not arguing that making machines is impossible. I'm arguing that making machines that make is *not easy enough*.

Access to precision manufacturing is available at large scale but not at low volume. To still make precision manufacturing available at scale, we accept the compromises of mass-manufactured products. Those compromises range from functional, to social, to environmental, to cultural issues. Being able to have an automobile only in black might not be a big compromise. But the compromises we make on labour conditions or environmental impact are.

To manufacture goods at low volume without high cost or loss of complexity, we need access to the precision and reconfigurability that digital fabrication provides. We need an infrastructure that provides this access to a broad base. As personal computers provided a platform for wildly diverse computing to happen at scale, we need personal fabrication to provide a platform for wildly diverse fabrication to happen at scale.

In this dissertation, I propose a conceptual framework called *object-oriented hardware* for machine design. Drawing from infrastructure implementation lessons learned in

network architecture and computer programming, I introduce a separation between machine infrastructure and machine applications. This paradigm crucially involves considering a specific machine as an *application* (with capacity to produce a specific good) rather than a (general purpose) tool or infrastructure. End-to-end principles guided the development of internet infrastructure and applications; I argue that end-to-end principles should also be applied to machine design. In an object-oriented hardware paradigm, machine building infrastructure is built up of modules that can be configured to make machine instantiations, separating machine application development from machine infrastructure development. In this dissertation I present specific technical implementations of machine infrastructure within the object-oriented hardware framework, including distributed networked controls, a framework for writing software for machine applications, and mechanical modules for building up machines. I consider all these implementations as objects that have interconnectivity across domains. This means that software objects are peers of mechanical objects in an object-oriented hardware paradigm. This enables easy (re)configuration of machine assemblages. I finally make these implementations widely available and observe their benefits as people use them to build machines using a Cardboard Machine Kit.

I hope that this work will help enable broad base participation in the development of digital fabrication technology. Thank you for taking the time to read.

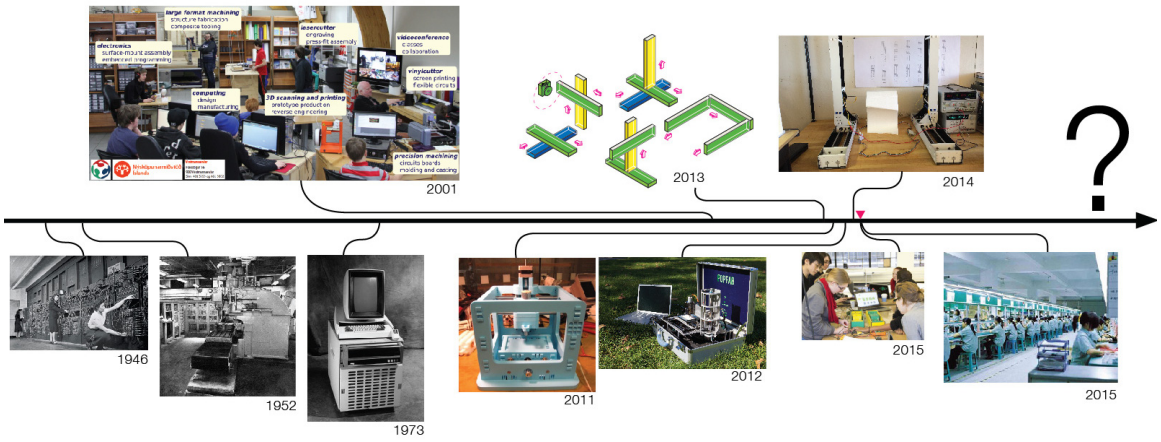


Figure 1-5: A timeline from the first digital to the first personal computers, and from the first time a milling machine was connected to a computer, to a Fab Lab, to Fab Labs making machines such as the *MTM Snap* to rapid prototyping of rapid prototyping machines.

Chapter 2

Background and Related Work

The work presented in the next chapters draws from fields including advanced manufacturing, robotics, material science, and computer science. It also is influenced by commercial development of machines for makers and by commercial development of client-side browser-based computation. This chapter lists some of these fields' contributions to digital fabrication as well as their influences on the work I present.

2.1 Automation and Manufacturing

Automation enables precision and accuracy in manufacturing, which in turn enable speed and throughput. The first computer numerical controlled (CNC) machines were developed to fabricate precise parts reliably [59]. CNC tools were not originally intended to be used for mass production. Industries around 'advanced manufacturing techniques' (AMTs), such as computer aided design (CAD) or computer aided manufacturing (CAM), were also initially more concerned with precision than throughput [41]. However, paired with infrastructural improvements these techniques quickly led

to mass-manufacturing capabilities of small-toleranced goods [9]. Flexible manufacturing systems (FMS) were built to carry materials to and from machine tools as well as to aggregate their instructions [38]. A contemporary example in the wild is Apple Inc.'s use of CNC milling machines to produce the enclosures of their computer and phone products [72]. A single product from the Apple line sells on the order of millions of units per year, so the reprogrammability that CAM enables is hardly used. Instead, the products use the precision of the manufacturing process to emphasize connotations of luxury and expense in their designs.

The market focus on production throughput has come somewhat at the expense of focus on startup costs. CAD/CAM systems were widely deployed in manufacturing plants, where initial investment in both capital and system expertise are quickly amortised over the operating expenses. When manufacturing products in low volume, those startup costs can be very high in comparison to the operating costs of a short run. As a result, goods that are produced in low volume with high precision are often prohibitively expensive. Here we see, for example, that highly precise and complex goods such as cell phones or automobiles are manufactured in volume. Custom electronics or electromechanical systems are often reserved for high-cost markets such as the military or medical devices.

Despite the limitations of existing advanced manufacturing, we draw heavily from these practices when developing the software, electronics, and hardware that I describe in the following chapters. We evaluate the bottlenecks with existing manufacturing software input and file formats [92] and use that information to inform our system design. We use the communication protocols that are standard to machine control and flexible manufacturing systems such as Profinet as the benchmark for minimum levels of bandwidth and robustness [64]. We use the extensive research in mechanical systems, especially with respect to machine stiffness and vibrations, to inform the designs of the machines I present [74].

2.2 Architecture

In the field of architecture, thousands of unique parts are the norm rather than the exception [15]. Buildings like the MIT Stata Center (designed by Frank Gehry, fabricated by A. Zahner Co.) use complex curves and precision-cut sheet metal as building blocks. Parametric design tools have completely revolutionised how architectural designs are produced [54]. Digital fabrication machines like laser cutters or 3D printer were used extensively in architecture studios. Design tools that were originally developed for the aerospace industry were appropriated by architects for the modelling of complex surfaces and structures [7]. Robotic arms that were originally designed for automobile manufacturing are being used for architectural construction [31; 32]. 3D printing is being employed at architectural scale [8], and even autonomous flying robots are being tested for construction [89].

For many of these tools architecture is appropriating technology originally designed with other applications in mind. But architecture in turn also improves or redesigns these tools. Robotic arms are notoriously tedious to program, with proprietary path planning and simulation packages. Architecture practicioners have developed open alternatives for machine control that are now also being used in manufacturing [11; 10]. Architects develop software tools for their workflows, such as Rhino with its dataflow programming language Grasshopper (and its rich user-created ecosystem of add-ons), or by combining structural engineering software with design software [58].

We are heavily influenced by the model, prototype, test, and critique architecture workflow in the development of our machines. We draw both from the architectural practice of using technologies for unintended purposes and reiterating on those tools to fit the task at hand. The futurist narrative of how architecture will relate to the built environment is also of inspiration [84]. We especially take advantage of parametric design tools, fabrication methods, and design critique.

2.3 Robotics

The field of robotics deals with many of the same system integration problems that digital fabrication machines have; it needs to coordinate motion and work through mechanisms, control systems, and software interfaces. Robotics on top of that also deals with issues of portability, mobility, and weight limitations. For machine tools, we learn from how these issues have already been tackled in robotics. Industrial robots often work tirelessly as welders, painters, or cutters in factories and manufacturing plants. However, here we consider specifically the influence of academic robotics.

Modularity is a clear benefit in robotics. Systems can be extended by adding extra robots and repairs can be carried out offline as replacement parts are employed. This introduces versatility, robustness, and potentially also lower cost of manufacture. The interconnect used in modular robotics is therefore of great influence in our work [96; 95].

Unlike in machine tools, robots commonly use parallel manipulators despite the increased complexity of their control [34]. The advanced control techniques and the corresponding communication protocols widely used in robotics therefore also influence our methods. Closed loop control is standard in robotics, yet still not widely used in automation systems. We take advantage of the work that has been done in robotics to make closed loop control more accessible.

Robotics researchers also consider new modes of human-computer interaction for machine tools. We are indebted to novel work on interaction such as the hand-held smart carving tool *FreeD* [94], or the smart router that uses computer vision for position [68].

Digital materials or programmable matter are an extension of modularity across all length scales. Robotic sand, reconfigurable blocks, or folding robotics chains all fall

under this category [12; 28; 36; 93]. A future in which all matter is programmed instead of assembled is a paradigm that also influences our design choices for structuring manufacturing infrastructure.

In *A Method for Building Self-Folding Machines*, Felton et al. describe a method for active folding of robotic structures [23; 24]. By taking advantage of the precision of CNC scoring and cutting as well as cleverly designing error-correcting joints, folding can be a very fast and precise construction method. Especially with our work on cardboard construction kits for precision machine design, we draw inspiration from this body of work [60; 90].

2.4 Self-Assembly

Robotic programmable matter or digital materials harness systems of self assembly (or codes describing materials) that have long existed in biology or chemistry. If all molecules can be made out of a limited number of types of atoms, or if living tissue can be made out of a finite number of amino acids and building molecules, then why can we not design the construction of our built environment in the same way? Self-assembly in chemistry and biology is therefore also of influence in our work [88; 87]. Specifically how codes can describe materials directly rather than describing the instructions to the machine building the object.

Assembling digital materials for the production of electronics or composite materials has been explored [42; 13]. How to get nano-scale robots to assemble these structures is also an ongoing research investigation [18]. Finally, taking advantage of how materials may change over time is the subject of '4D printing' [83]. While our work does not directly relate to self-assembly or digital materials, we believe that we are part of the lineage going from codes giving machine instruction to codes describing materials.

2.5 Materials

Material science has recently had great impact on digital fabrication. Instead of etching or milling circuit boards, functional inks can be used to write PCBs directly [70]. Functional inks have had great influence on additive manufacturing especially with researchers developing processes for 3D printing tissue scaffolding, electronics, antennas, lithium ion batteries, etc. [44; 91; 80; 1]. These kinds of processes have also put pressure on developing new file formats and data representation models for 3D printing, for example to accommodate multi-colour or different levels of gloss [20]. These materials for additive manufacturing are not limited to current 2D deposition methods such as inkjet printing or fused deposition modelling. Because of these materials, there is also pressure to develop new additive manufacturing workflows [33]. Voxel8 is a new company that recently released beta-versions of their printers that can make 3D circuitry [5]. These new materials and the workflows and end effectors has influenced our machine development work.

2.6 The Maker Movement

The general population has typically been of little interest to the researchers from the preceding sections. However, since the turn of the 21st century there has been a marked increase in attention to *making*. The 'Maker Movement' engages the general public in discourse on technology, craft, and the politics of DIY [2]. Maker Faires, as a kind of technology show-and-tell, are held throughout the world demonstrating clever technological solutions or educational kits relating to science, techonology, engineering, and math. Manifestos such as the 'Right to Repair' or the 'Maker's Bill of Rights' or 'The Maker Movement Manifesto' [35] are somewhat reminiscent of the hacker ethos of the 90s in that they demand technological self-sufficiency and the

Figure 2-1: These are MIT architecture students using a 6-axis robot arm to cut complex surfaces into the foam block on the right. To create the toolpaths, they first designed the surfaces and Rhino, then used RhinoCAM to create instructions for the robot arm. The cutting end is an electric hot knife. To regulate the temperature of the hot knife, the student standing on the left is observing the cutting and flipping the switch on a surge protector on and off. This is an extremely intelligent controller for an extremely simple control task. However, despite these students being proficient at technologies such as CAD, CAM, and robot control, the system integration is tedious enough that the easiest way for them to get their system running was to incorporate an imprecise yet very expensive controller–a human.

related right to open up boxes and look inside. It is common for this movement to be heralded as a revolution (the third industrial revolution, a manufacturing revolution).

Whether or not there is a revolution ongoing, there is certainly an economic opportunity that has formed. Manufacturing 'maker' wares as educational toys has flourished. Crowd-funding sites such as Kickstarter or Crowdsupply have enabled entrepreneurs to get funding for their technology products through pre-orders, for some several millions of dollars. Some maker oriented startups that came from crowd-funding platforms have enjoyed multi-million dollar exits. Most importantly to the work we present here, the maker movement has been unfalteringly of the opinion that (digital) fabrication belongs in the home. Makers are the early adopters of personal fabrication, and now co-create much of the work in the field of digital fabrication. Maker engagement has led to the momentum in personal and digital fabrication that has been critical to this research.

2.7 Free and Open-Source Hardware

The Open-Source Hardware Association provides a definition of what qualifies as Open-Source Hardware. Unlike Open-Source Software, the definition is slightly more complex, as copying pieces of hardware involves more work than just ctrl-c, ctrl-v. You need to reproduce manufacturing processes, source the same parts, and assemble in the same way.

From `http://www.oshwa.org/definition/` [3]: *Open Source Hardware (OSHW) is a term for tangible artifacts—machines, devices, or other physical things— whose design has been released to the public in such a way that anyone can make, modify, distribute, and use those things.* The definition of OSHW includes documentation of the work such that it is reproducible, stating the scope of the work, including any

software that is required to run the work, and stating the the limitations on sharing, redistributing, and modifying the work.

I believe that free and open-source hardware is imperative for creating more democratic technologies. Goods and products need to be verifiable, modifiable, and reproducable if we are to trust and use them. For this to be a reality we need standards for publishing and sharing such as those outlined by the Open Hardware Association.

I argue that sharing free and open-source hardware depends on the infrastructure of digital fabrication—where codes can describe materials. Physical objects are not the part that is freely shared, their designs are. Digital fabrication instructions can describe an object completely. To enable the distributed production of goods, we need accessible digital fabrication technology. So it is in service of veriable, modifiable, and reproducible goods that I am working on digital fabrication infrastructure. I am indebted to the thinking and organisation of the free and open-source hardware movement.

2.8 Browser-based Computing

While interpreted JavaScript has historically been slow in comparison to compiled languages such as C, modern implementations of JavaScript use just-in-time compilation techniques to close this gap. V8 for example is the JavaScript execution engine that was written by Google for Chrome and open-sourced in 2008. V8 was written in C++ and compiles JavaScript into machine code, giving in-browser JavaScript performance that is comparable to compiled C. Firefox has their own JavaScript execution engine that performs comparability. In developing the work described in Chapter 4, we rely heavily on this new capacity of JavaScript, and are indebted to the Google and Firefox engineers who have fixed the bugs we found while running our software.

2.9 The Machines that Make Project

The MIT Center for Bits and Atoms has cultivated an ad-hoc group of machine building practioners loosely grouped under the name the Machines that Make project. The project was originally aimed at sustaining the projects that would come out of the class *MAS.865, How to make something that that makes (almost) anything* which has been taught intermittently since 2004, most recently in 2015 by the author. The class focusses on machine bulding fundamentals but with a strong inclination towards low-cost high-precision machines for personal fabrication. Documentation of class projects is collected at `http://mtm.cba.mit.edu`. Past students include Manu Prakash, who has commercialised origami microscope Foldscope [16]; Max Lobovsky, who founded the desktop 3D printer company Formlabs [66]; Jonathan Ward, whose work led to the desktop mill Othermill; Ilan Moyer, who founded the handheld smart router company Shaper [68; 57]; James Coleman, who works in digital fabrication R&D at Zahner [15]; and many more individuals whose research has inspired and shaped the work presented here.

Chapter 3

Networked Controls

If computing had gone from mainframes to laptops, then what was the laptop for personal fabrication? *PopFab* is a a pop-up multi-head fabrication tool that fit in a briefcase Ilan Moyer and I developed in 2012 (see Figure 3-1). We sought to make a general-purpose digital fabrication platform for different processes, including both additive and subtractive manufacturing. For mechanical positioning of the different end effectors, we used a ball/slot kinematic mount [75]. This allowed us to attach the 3D printing head, the knife head, and the milling head precisely each time with only a single thumbscrew for applying preload. However for the control system our approach was to use one control brain for all the functions of the machine. This was fine for the motor control of the XYZ motion, as this was the same for each machine configuration. We were using a mechanism similar to an H-bot for the XY platform called *CoreXY* [55], and a direct drive lead screw based drive train for the Z-axis. But for the different heads we needed different control electronics—for spindle control, heater control, etc. To save space and simplify the design, we used the same electrical connection any high-current head attachments, e.g. an extruder heater would be controlled by the same connection as a spindle motor. This was simpler when developing the hardware, but after developing the control board it was difficult to introduce new functionality

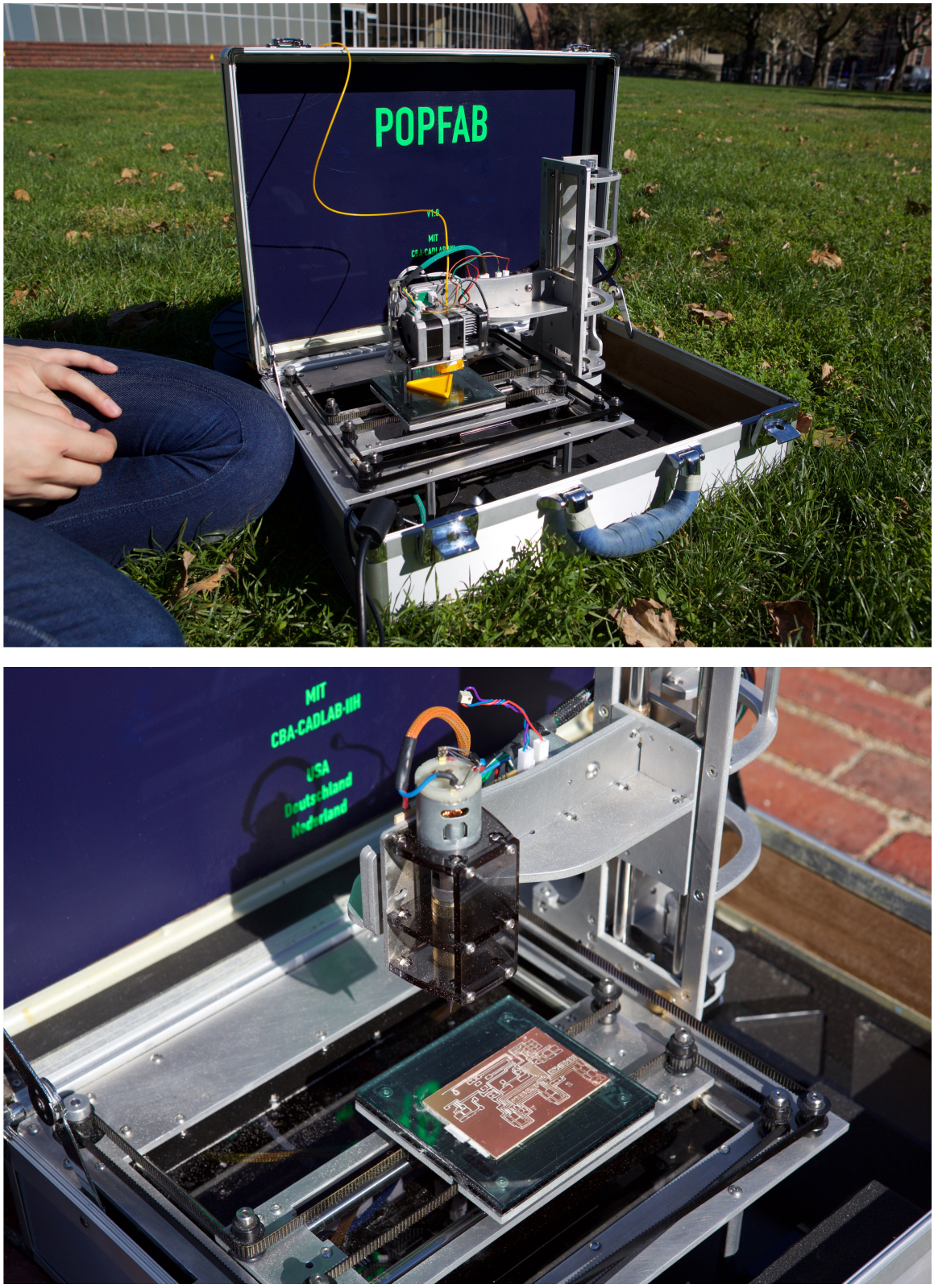


Figure 3-1: The PopFab, a portable pop-up fabrication machine here shown 3D printing and milling a circuit board. It has heads for milling, cutting, and imaging.

to the machine's control network. To be able to control *PopFab* in different machine modes we wrote many kinds of file format conversion software to be able to work on top of the hacked together system architecture. Our electrical control system was inextensible, and our software platform was quite fragile, despite our mechanical platform being suitable for attaching any alternate end effector.

Is it possible to create distributed control systems for digital fabrication machines? Specifically, is it possible to use a control network that is extensible to operate digital fabrication equipment? Instead of having to redesign a control circuit board when we modify a machine, can we instead add modules to a distributed control network?

In processing plants or manufacturing assembly lines there are so many operations that need to be managed that central control is not possible. Distributed control is implemented in large plants (e.g. petrochemical, pharmaceutical, paper, etc.) by having an operator interface which controls a series of programmable logic controllers (PLCs) which in turn use a fieldbus to do real-time control of the motors, sensors, etc. that are being used [50]. Timing critical elements are managed by fieldbusses, whereas less critical elements are connected through e.g. ethernet. PLC programming requires knowledge of the particular subsystems that are being used and their respective control languages. These could be of many types, including CAN bus, Foundation Fieldbus, Profibus, Modbus, EtherCAT, etc. (8 types are described in IEC's 61158), all of which promise interoperability, interchangability, and interconnectivity. The standards deliver on that promise to varying degrees with various bandwidth and power limitations, which is an indication for why there are so many standards. For large-scale plant processing, the amount of time it takes to design, implement, and calibrate all these systems is a fraction of the running cost, making the up-front investment in communication infrastructure worthwhile. For low-volume manufacturing or other types of small-scale automation, implementing distributed control through these technologies quickly becomes a large fraction of the running cost. Therefore,

thusfar distributed control has been employed only at large scale and high cost. To be able to make distributed controls available at a very small scale and low cost, we (like many others before us) have implented several standards of our own for machine communication.

3.1 APA: Asynchronous Packet Automata

Historically deterministic protocols have been preferred for real-time network communication applications. The timescales at which network nodes could process and pass along packets was historically comparable to the timescales in which a machine operation could take place. Then missing a packet would mean missing an entire instruction cycle. However, now the timescale at which data processing and networking takes place is several orders of magnitude off from machine instruction requirements (e.g. regular 1000BASE-T ethernet runs at 1 Gbit/s, compared to RS-232's 115200 kbit/s), and the benefits of a distributed asynchronous network for machine control become attainable.

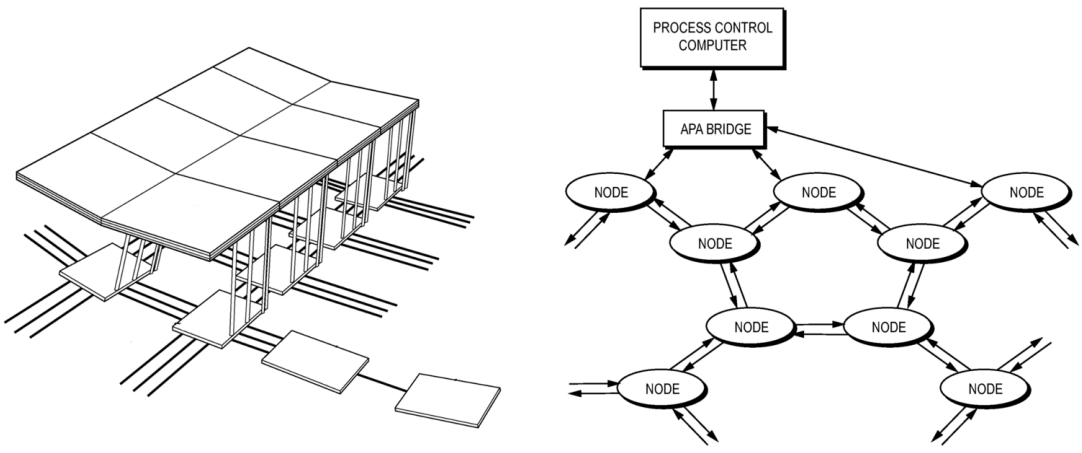


Figure 3-2: An APA network for tiled heaters. Image from *Method and Apparatus for Online Calorimetry* [25].

Composite airplanes such as the Airbus A380 or the Boeing 787 are ramping up in production. To make large composite parts, the aerospace industry has to make even larger autoclaves. The autoclave size increases linearly, its cost increases exponentially. Working with the airplane fuselage and wing manufacturing company Spirit Aerosystems, we came up with the idea to replace autoclaves with smart moulds that would use networked heating tiles to cure the epoxy (see Figure 3-2 [25].

If we make the tiles 10 cm long and use them on a wing that is about 30m long, we can expect to use at least 1000 tiles for a single smart mould. Most fieldbus standards support up to a few hundred nodes, and would require us to set addresses for each node [50]. To avoid needing to specify addresses or determine routing beforehand we used both *network coordinates*, meaning that the location of the APA nodes was the same as their address, and *source routing*, meaning that packets specify the route the packet takes through the APA network.

The form of an APA packet is `{121^|payload}`. The `{}` is the packet framing. The `121` is the address to the node where each number is a port to exit on along the route. The `^` is where in the route we are, such that `12^1` is midroute and `^121` has arrived at its destination. The address can be arbitrarily long, and several addresses might address the same node. The `|` delimits the routing from the payload. The payload could be of arbitrary length, and if it contains special characters they are escaped with \.

The hardware handshaking for transmitting an APA packet involves raising a line to indicate there is a packet to send. The recipient node acknowledges by raising the second line. If the recipient node does not acknowledge, the packet does not transmit. This *backpressure flow control* makes sure that no packets are lost due to bottlenecks in the network. The timing of the data transmissioon can set to be the same for all nodes, or the handshaking can be used to determine the data rate. There is no central clock for the network communication; it is asynchronous. Node to node

communication can execute at any time.

The topology of a network can be discovered by having nodes send out flood packets. Because the naming is the same as the routing, longer addresses will mean routes with more hops. This provides a good heuristic for selecting the shortest route for the packets. However, if there are triangle inequality violations, these will still form an issue for APA.

The original implementation for APA was designed to work at thermal timescales. Even if a packet roundtrip was more than a tenth of a second, it would still be much faster than the thermal response of the epoxy/fibre system. But the benefits of the APA communication protocol were applicable also to other systems, including digital fabrication machines. The first networked control system I tried for a small mill was made with repurposed APA nodes outfitted with motor controller daughter boards. However, the timescales of even a small mill were higher than the speed at which we ran APA. Reimplementing APA with faster processors would have likely dealt with some but not all of the issues. Therefore we created several interim solutions for distributed control using busses instead.

3.2 Simple Machine Bus

Unlike asynchronous communication busses communicate with all nodes on the network at once. If there are any delays in the bus network, it is due to the physical layer (although wires would have to be very long for this to be a factor in our setup). The simplest bus protocol we implemented used a shared software serial bus implemented on an 8-bit AVR microcontroller (the ATTINY44) for receiving packets. We used this bus for controlling the *MTM Snap* milling machine that is depicted in Figure 1-3. The receive and transmit lines (rx/tx) were bussed through using RJ9 connectors (see Figure 3-3). Each bus node connected to its own daughterboard for stepper motor

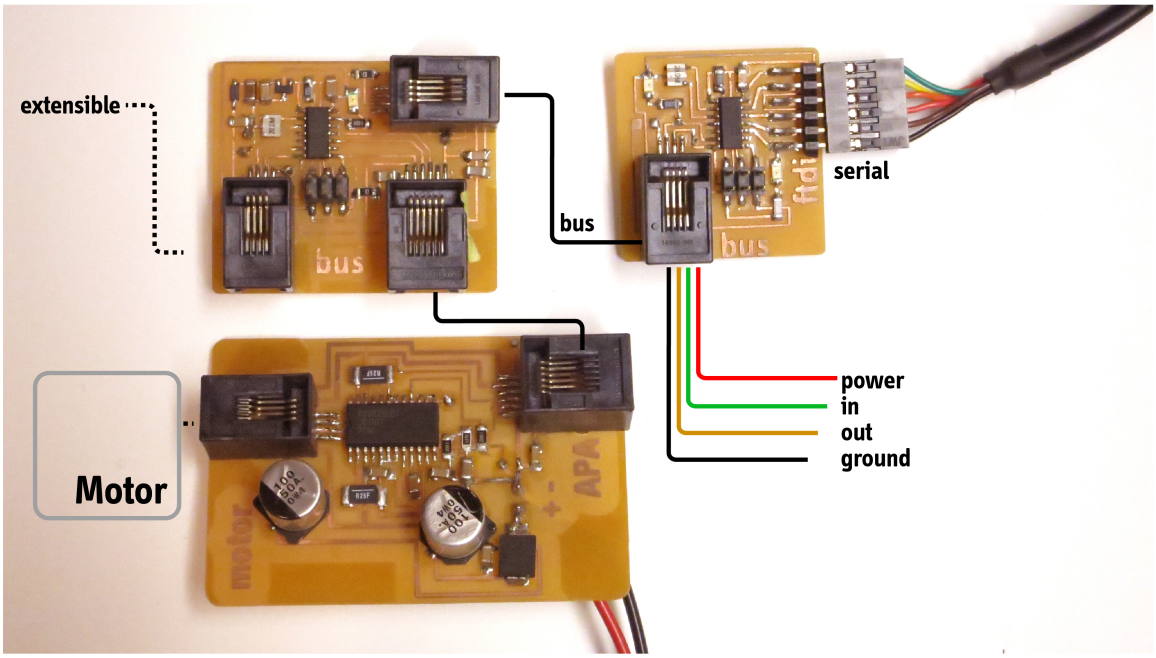


Figure 3-3: A simple 2-wire bus network, here shown with a stepper motor daughter board.

control, which in turn was connected to the *MTM Snap*'s stepper motors. Packets were sent on the line in the form of

$x\ hi\ lo\ Y\ hi\ lo\ Z\ hi\ lo\ T\ hi\ lo\ G$

where x, X, y, Y, z, or Z are IDs for the bus nodes. Each node is programmed with its ID, where for example x denotes a move in the positive direction along the x-axis, and X denotes a move in the negative direction. Likewise for the other axis identifiers, which might be y, Y, z, Z, a, A, b, B etc. After each identifier, a high and low character are sent, which add up to the number of of steps to be taken. The T identifier is special, as it denotes the number of milliseconds during with that number of steps should be executed. Because each move needs to execute for all axes in the same amount of time, the T value is the same for all nodes connected to the bus. The g character is not an identifier, but a *go* command. When it is processed by the nodes on the network, they all execute the previous command that has been addressed to

them. In the case of a packet x 00 01 Y 00 FF Z 00 00 T 01 FF G, the move will be $x + 1$ step, $y - 255$ steps, and z 0 steps in 511 ms. The nodes would all first receive both their number of steps as well as the time they had to execute those steps. These values would already be saved by the time the nodes received the g go command.

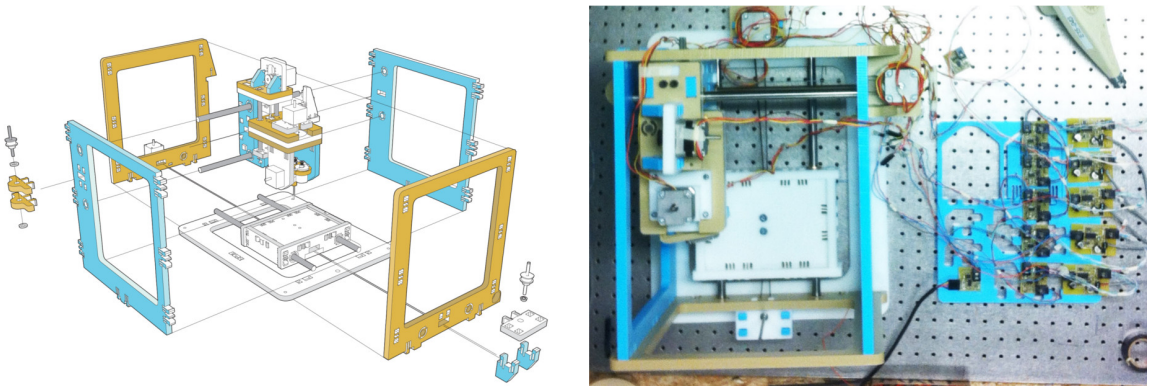


Figure 3-4: The simple bus network controlling a 5-axis timing-belt driven version of the *MTM Snap*, made by James Coleman.

This bus implementation was fast to set up, low cost, and could be extended on the fly, all attributes we were aiming for. Furthermore, the control boards could be milled by the milling machine they were used to control, which was a nice recursion. However, this implementation was also plagued by low bandwidth, rounding errors, in practice one-way communication, and a large number of boards for a relatively simple application (e.g. 3 stepper motors and a spindle motor would use 4 bus boards, 4 daughter boards, and a bridge board to run).

3.3 Fabnet

Although we wanted an asynchronous implementation of a distributed network, we also needed control systems for the machines we were developing. The 5-axis machine shown in Figure 3-4 could execute simple moves, but was hardly a robust platform for

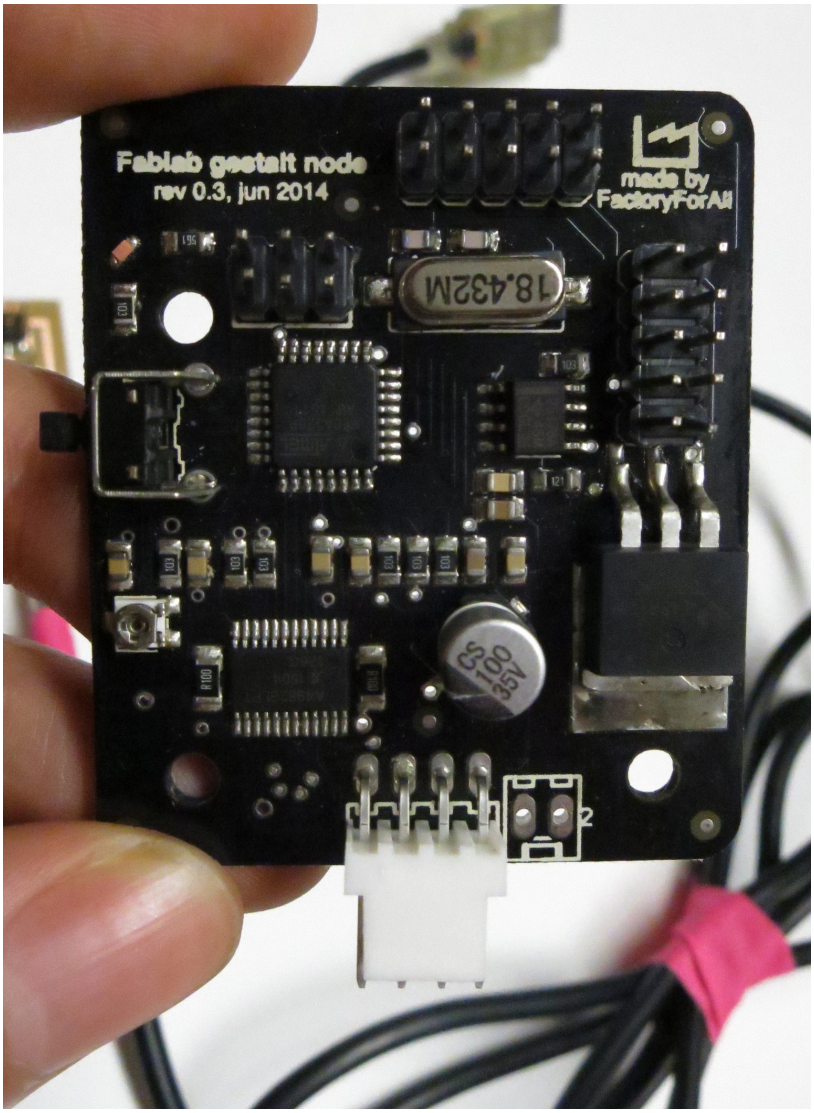


Figure 3-5: A gestalt node, including an ATMEGA328 microcontroller, an RS-485 differential bus transceiver, a stepper motor driver for up to 2A of current, and button for identifying the nodes on the network. Thanks to FactoryForAll, this particular board version features all pink LEDs. The schematics are available in Appendix A, or in [62].

5-axis precision milling. The same time we were hitting the limitations of the simple bus network, Ilan Moyer was developing a control library for automated tools called *pyGestalt* [56]. His library specified the use of a communication network he called *Fabnet*, which extends the RS-485 standard for multipoint networks with sync, error, and stop lines [56]. To be able to take advantage of the library, Ilan and I developed a *fabnet* compatible node for stepper motors which is depicted in Figure 3-5, as well as several other end-effector specific nodes such as the extruder node shown in 3-6.

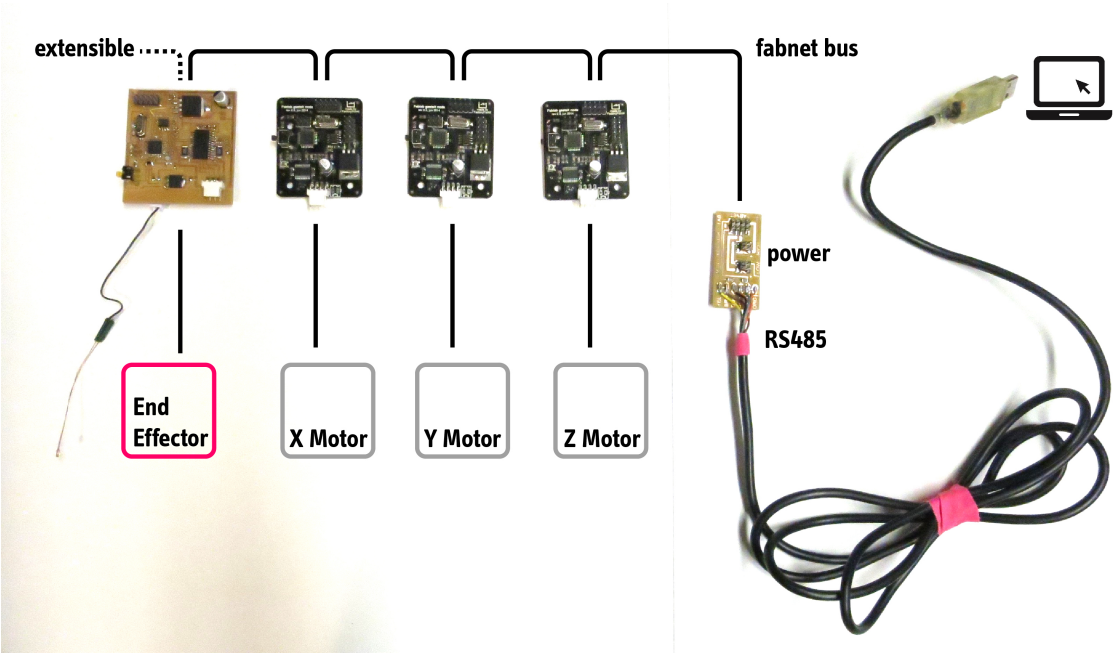


Figure 3-6: A network of gestalt nodes. Here are 3 stepper motor gestalt nodes, and one extruder node with temperature sensing, extrusion motor control, and heater wire.

The gestalt node receives commands, can send back statuses or data, and can controller stepper motors. Unlike APA it has synchronisation between nodes. It runs firmware that is complementary to the pyGestalt library. This firmware implements a soft synchronisation, where nodes are set up with initialising packets and a synchronisation is sent multicast to all nodes. This is similar to the simple bus network, but more robust because it is using a multipoint protocol. Note that any kind of node can

be connected to the fabnet network, not just stepper motor controllers. I have also developed gestalt nodes for sensors such as temperature, and other actuators such as DC motors or heater wire.

The gestalt nodes can theoretically be connected with any network topology, but practically the stepper gestalt nodes have been designed with only two fabnet connectors. Practically, this means the network topology is that of a bus. An example configuration of gestalt nodes can be seen in Figure 3-6 using a 3-axis 3D printer as an example machine. There are 3 stepper motor gestalt nodes, and 1 gestalt node designed for a fused deposition modelling extruder head.

The extensiblilty of the gestalt network makes it much more flexible than conventional control boards. It becomes trivial to add fourth or fifth axes to a digital fabrication machine, something which in conventional systems requires a substantial retrofit of controls.

In the next chapter on software and *virtual machines*, I will go into more detail about how to control a network of nodes.

Chapter 4

Software Control and Virtual Machines

There are many different types of software and file formats used in a digital fabrication pipeline. In a typical workflow, first a designer's intent is codified in a digital model. Then the model is translated into machine toolpaths. Finally the toolpaths are translated into motor commands by the "computer" of Computer Numerical Control. While CNC might sound like there is one computer involved in this workflow, it's typically a minimum of four, including the computer that provides the user interface to the CNC machines and the computer which issues commands to the motors.

Computer-Aided Design and Computer-Aided Manufacturing (CAD/CAM) have been applications of computers since well before computers had interactive displays. CAD work done on minicomputers was output with pen plotters in batch mode. Bezier surfaces were developed at Renault well before they could be displayed on a refreshing screen [65]. The work that Ivan Sutherland did on interactive CAD in 1963 in his dissertation *Sketchpad: a Man-Machine Graphical Communication System*[1] was

[1]Man-Machine or Man-Computer has unfortunately been the term of choice for many early

well before its time in commercial applicability. An interactive display at the time had an estimated cost of well over 100k USD, which would be about 800k USD in 2016. Interactive CAD gained much more traction after the first personal computers started appearing and progress was made on shape modelling with the invention of boundary representations and b-splines [67].

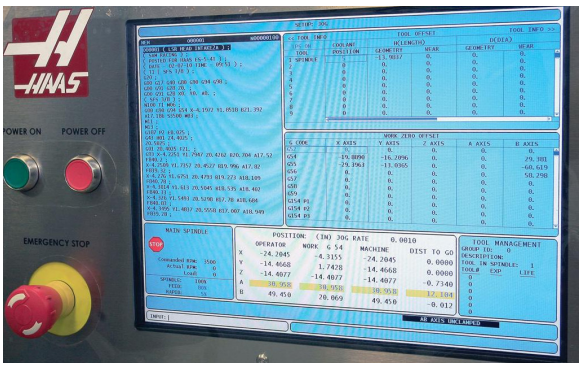


Figure 4-1: A current user interface for a 5-axis CNC milling machine.

CAM packages also became commercially viable in the 1980s with groups like Mastercam or Delcam commercialising academic efforts, and groups like Dassault Systems spinning out of industry spaces. Mastercam remains the largest CAM software company currently operating.

The computer interface to a digital fabrication machine receives the least attention in research, development, or user experience design. See Figure 4-1 for a typical example. These interfaces are commonly developed by the companies who sell the digital fabrication machines, whose core competency is hardly software usability. Due to these kinds of interfaces and methods of interacting with machines, the current job of a machinist is much more closely aligned with that of a computer engineer than

researchers in human-machine systems: I. Sutherland's *Man-Machine Graphical Communication System* [81]; J.C.R. Licklider's *Man-computer symbiosis* [45]; J. Martin's *Design of man-computer dialogues* [52]; N. Hogan's *Controlling impedance at the man/machine interface* [37] etc. After L. Suchman published the widely influential *Plans and situated actions: The problem of human-machine communication* [78] in 1987, the term human-machine started gaining traction, although many works are still published that exclude half of humanity in their terminology.

that of a traditional machinist.

This history of software and digital fabrication has led us to the software packages we translate between today (see Figure 4-2). However, the historical limitations that shaped the architecture of these systems do not still apply. In comparison to mechanical systems or electronics, software is easy to modify and rewrite. However, this rarely happens in a digital fabrication setting. The issues with interchangability, interoperability, and interconnectivity we saw in Chapter 3 on control networks are also in play here. Specific software workflows are not typically robust and break with small changes. Therefore users often work around the workflow's restrictions instead of modifying them.

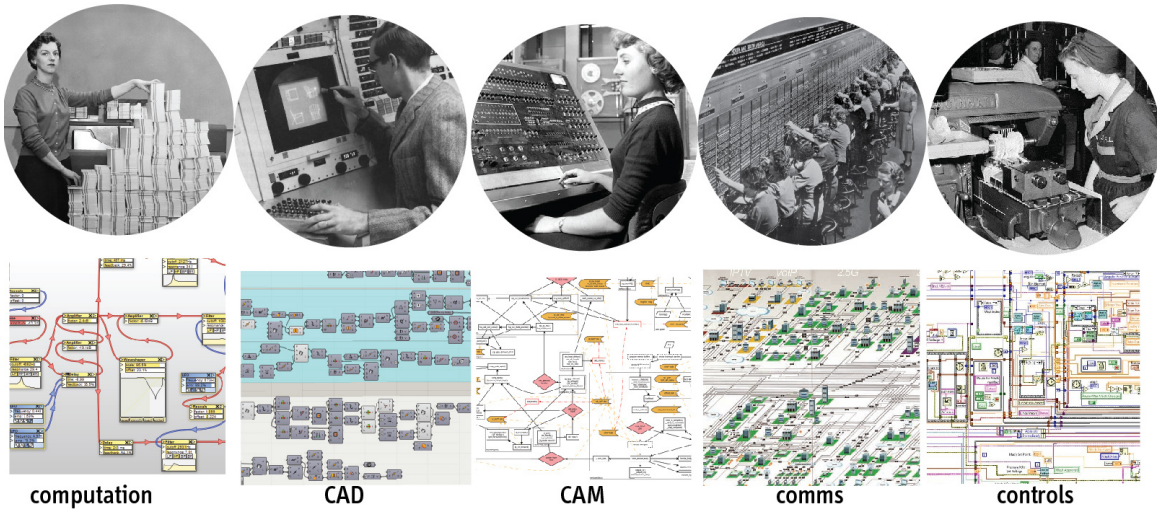


Figure 4-2: The historical separation of roles into computer, drafter, toolpath planner, machine interface, and machine control no longer apply. Although we have dataflow programming languages to describe each of these steps in the digital fabrication pipeline, their interfaces remain poorly connected.

How can we restructure the digital fabrication software pipeline for better control? How can we expose functionality to the end-user without requiring domain expertise? What are the modes of control we require?

4.1 Virtual machines

Using distributed networked controls such as described in Chapter 3 implicitly introduces three layers of separation in machine control. The first is the network of nodes that makes up the machine's control system. The last is the application layer that communicates with the machine. An intermediate layer is the description of how the distinct nodes form a machine instantiation. We call this layer the **Virtual Machine**. The Virtual Machine can help translate application-specific commands, such as "move to these coordinates", into commands specific to the network, such as "A-node needs to move its motor this much B-node needs to move its motors this much." If we consider how machine control layers correspond to the Open Systems Interconnect model, the virtual machine corresponds most closely to the transport layer (see Table 4.1). The virtual machine handles transmission and handshaking details, but also determines which node needs to receive what command.

OSI Layer	**Machine Layer**
7. Application	e.g. jog interface
6. Presentation	
5. Session	
4. Transport	Virtual Machine
3. Network	e.g. APA
2. Data link	
1. Physical	e.g. physical node

Table 4.1: Comparing the layers of machine implementaion to the OSI model for communication

The application layer might produce a series of coordinates for the machine to move to, such as is the case with g-code. With g-code the interpretation of which motor needs to move all happens on the g-code intepreter control board. However, if we have control networks, which motor needs to move needs to be described a layer above the network. This is the task of a virtual machine. The transport layer figures out

which node is addressed and when.

For example, if we have an XY stage that moves with two separate drive trains, the virtual machine will assign X motion packets to the X node and Y motion packets to the Y node. If we have an XY stage that moves with some form of parallel kinematics (where both motors are used in both X and Y positioning), then the appropriate components of the moves will be calculated by the virtual machine and assigned to the A node and B node. Conveniently this means that if we create an application, such as a program that generates coordinates to send to a machine, we can still apply that to several different machines as long as we swap out the virtual machine controller. This layering encourages better reusablility in application development.

The PyGestalt library provides a structure for describing virtual machines and their network interfaces in Python [56]. In the code below, I set up the parallel kinematics of an H-bot configuration together with a direct drive Z-axis.

```
def initKinematics(self):
        # drive components of h-bot.
        # Inputs are A/B stepper motors,
        # outputs are X/Y in machine coordinates.
            self.aMotor = elements.elementChain.forward(
                [elements.microstep.forward(4),
                elements.stepper.forward(1.8),
                elements.pulley.forward(2.03),
                elements.invert.forward(False)])

            self.bMotor = elements.elementChain.forward(
                [elements.microstep.forward(4),
                elements.stepper.forward(1.8),
                elements.pulley.forward(2.03),
                elements.invert.forward(False)])

        self.zAxis = elements.elementChain.forward(
                [elements.microstep.forward(4),
                elements.stepper.forward(1.8),
                elements.leadscrew.forward(8),
```

```
                elements.invert.forward(True)])

        xyKinematics = kinematics.hbot()
        zKinematics = kinematics.direct(1)
        compoundKinematics = kinematics.compound(
                                [xyKinematics, zKinematics])
        self.stageKinematics = compoundKinematics
```

In the kinematics definition of this virtual machine, I describe which node is connected to what hardware (in this setup, two pulleys and one leadscrew with 8mm of travel per rotation). Other definitions will determine the controller interface to the computer (a serial connection), and to the respective nodes (through a fabnet instantiation).

The virtual machine I defined can now be imported into other software programs and moves. Say my VM is called `virtualMachine`. An example program below would move the machine along the vertices of a 1.5cm cube:

```
import virtualMachine

if __name__ == '__main__':
        # The persistence file retains which node has been
        # assigned which ID
        myMachine = virtualMachine(persistenceFile = "mm.vmp")

        # This is for how fast we will move in mm/s
        myMachine.abzNode.setVelocityRequest(6)

        # Some random moves to test with
        moves = [[15,0,0],[15,15,0],[15,15,15],
                [0,15,15],[0,0,15],[0,0,0]]

        # Move!
        for move in moves:
                myMachine.move(move)
```

Making the list of moves can happen in the same way that g-code is currently produced from a digital design. However, it is just as easy to generate moves that respond to

information in real time, such as sensor data collected from the machine, or user feedback generated during runtime. Generating g-code and then running it is exactly how we have been running CNC mills for the past 60 years. However, this method of work was established well before we had access to the computational resources and sensor inputs we have now. Programming virtual machines to be able to respond to actions other than 'move', 'spindle on' is something that is now well within the scope of this work.

4.2 Application interfaces

In 4.1, the example we give for a simple application is a jog interface for a machine. A user presses arrow keys, and the machine moves accordingly. This is perhaps the simplest application we can conceive. How can we make it easier to generate machine control applications? How can we allow the user to incorporate things like sensor data or other feedback?

4.2.1 *Mods*

Mods is a dataflow programming environment where modules can be connected into a program. *Mods* is written in JavaScript and uses the Node.js runtime environment for machine interfacing. Much of the *mods* development has been done by my thesis advisor, Neil Gershenfeld. Each module performs a function and can output it to another module. For example, a single module might threshold an image. Each module contains its own appearance code as well as the code for the function it executes. If that function is compute-intensive, the module will spawn a worker to

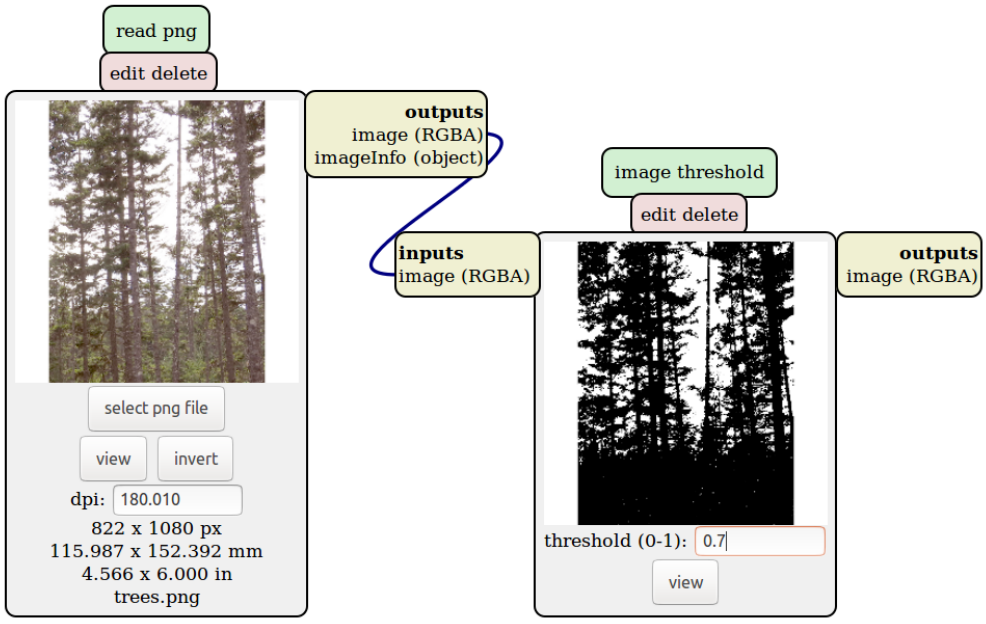


Figure 4-3: Two modules in the mods environment. The first reads in .png images. The second takes an image and applies a threshold function to it.

perform the computation without crashing the webpage. For example, this is the worker that performs image thresholds:

```
function worker() {
   self.addEventListener('message',function(evt) {
      var h = evt.data.height
      var w = evt.data.width
      var t = evt.data.threshold
      var buf = new Uint8ClampedArray(evt.data.buffer)
      // for each colour channel
      var r,g,b,a,i
      for (var row = 0; row < h; ++row) {
         for (var col = 0; col < w; ++col) {
            r = buf[(h-1-row)*w*4+col*4+0]
            g = buf[(h-1-row)*w*4+col*4+1]
            b = buf[(h-1-row)*w*4+col*4+2]
            a = buf[(h-1-row)*w*4+col*4+3]
            i = (r+g+b)/(3*255)
            if (a == 0)
```

```
            val = 255
         else if (i > t)
            var val = 255
         else
            var val = 0
         buf[(h-1-row)*w*4+col*4+0] = val
         buf[(h-1-row)*w*4+col*4+1] = val
         buf[(h-1-row)*w*4+col*4+2] = val
         buf[(h-1-row)*w*4+col*4+3] = 255
         }
      }
   self.postMessage({buffer:buf.buffer},[buf.buffer])
   })
}
```

That worker is called by the threshold module, which is depicted in Figure 4-7. To be able to vectorize an image so that its edges can be used as toolpaths, we can connect a series of modules together. Based on the binary image generated by the threshold function, we can calculate the distance map (which labels each pixel with the distance to the nearest boundary pixel). Using the distance map, we can easily offset the image to account for the diameter of the tool we are working with. Distance map and Offset are both existing modules in the *mods* environment.

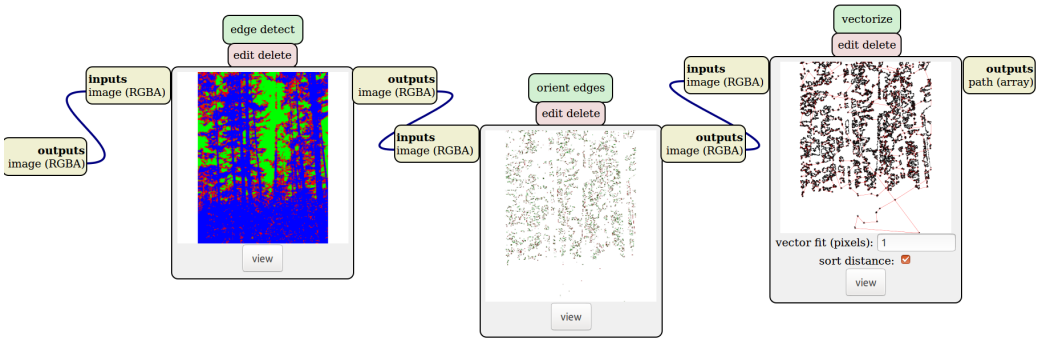


Figure 4-4: *Mods* edge detection of a binary image, edge orientation of vertices, and vectorization and vector sorting for toolpath generation.

After generating a binary image, we can detect the edges of our now-offset image. This is done with the Edge detect module, depicted on the left in Figure 4-4. The edges can then be sorted for direction (north south, east west), before they are vectorized by the Vectorize module (all shown in Figure 4-4).

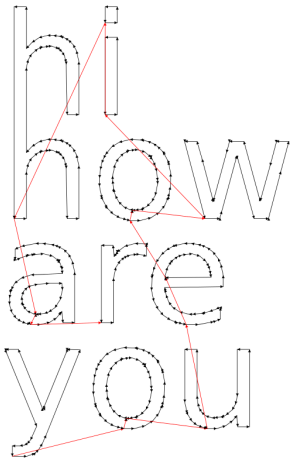


Figure 4-5: A closeup view of the *mods* vectorize output on a different image.

In the *mods* environment, besides connecting groups of modules into program, users can also edit modules themselves. They can also write new modules and add them to their *mods* programs. The modules are then executed immediately; they do not need to be uploaded to a server. The functions of the modules all execute in-browser. For example, the calculations that need to be done for distance transforms execute within the browser environment. Any of the *mods* programs can run offline. For convenience, the *mods* environment is served from `http://mods.cba.mit.edu` and includes many of the modules we have built.

To be able to talk to a machine, the *mods* enviroment does need to talk to a server that has access to that machine. To do this, we use Node.js and WebSockets to run a local server that listens for commands from *mods*. The code for running such a server

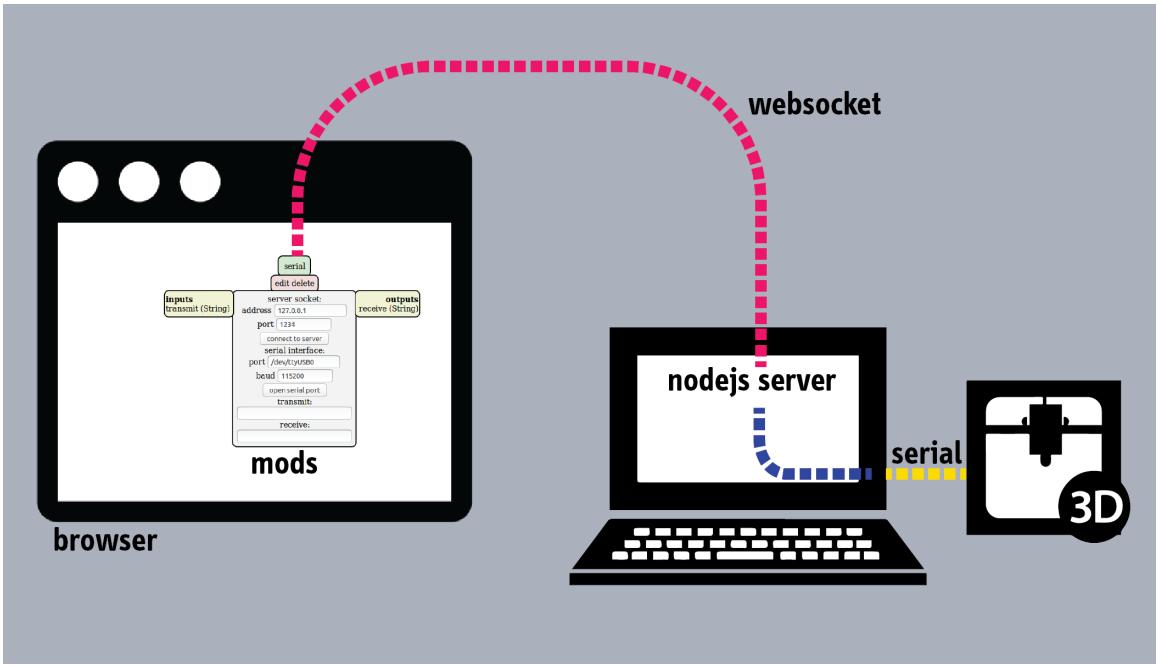


Figure 4-6: *Mods* communicating with a physical machine through a websocket.

is as follows:

```
// fabnetserver.js

var server_port = '1234'
var client_address = '127.0.0.1'

console.log("listening for connections from
                "+client_address+" on "+server_port)

var server = {}
var WebSocketServer = require('ws').Server
wss = new WebSocketServer({port:server_port})

function worker(ws,arg) {
   var child_process = require('child_process')
   console.log("python fabnet_plotter.py '"+arg+"'")
   child_process.exec("python fabnet_plotter.py '"+arg+
                      "'",function(err,stdout,stderr) {
        ws.send(stdout)
        })
```

```
    }
server.worker = worker;

wss.on('connection',function(ws) {
   if (ws._socket.remoteAddress != client_address) {
      console.log("error: client address doesn't match")
      return}
   ws.on('message',function(msg) {
      server.worker(ws,msg);
      })
   })
```

This particular server receives a list of coordinates as a string from a client module in *mods*, and uses that list as an argument to pass to a Python program with a PyGestalt Virtual Machine. It listens to 127.0.0.1:1234, which is localhost. It is possible to run the fabserver on a different computer than the computer that is connected to the machine. However this code is not very secure and could easily be reprogrammed to run exploit code. Other servers use the JavaScript serial library to talk directly to fabnet, however we do not have a kinematics library such as PyGestalt implemented in JavaScript yet.

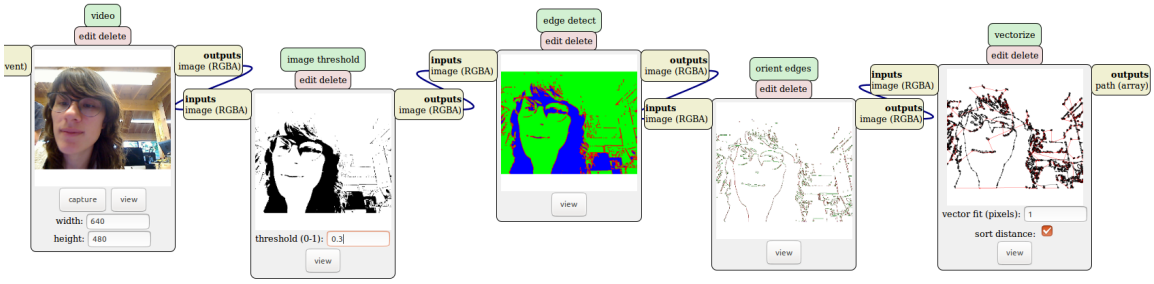


Figure 4-7: *Mods* image capture from a webcam to a cutting toolpath.

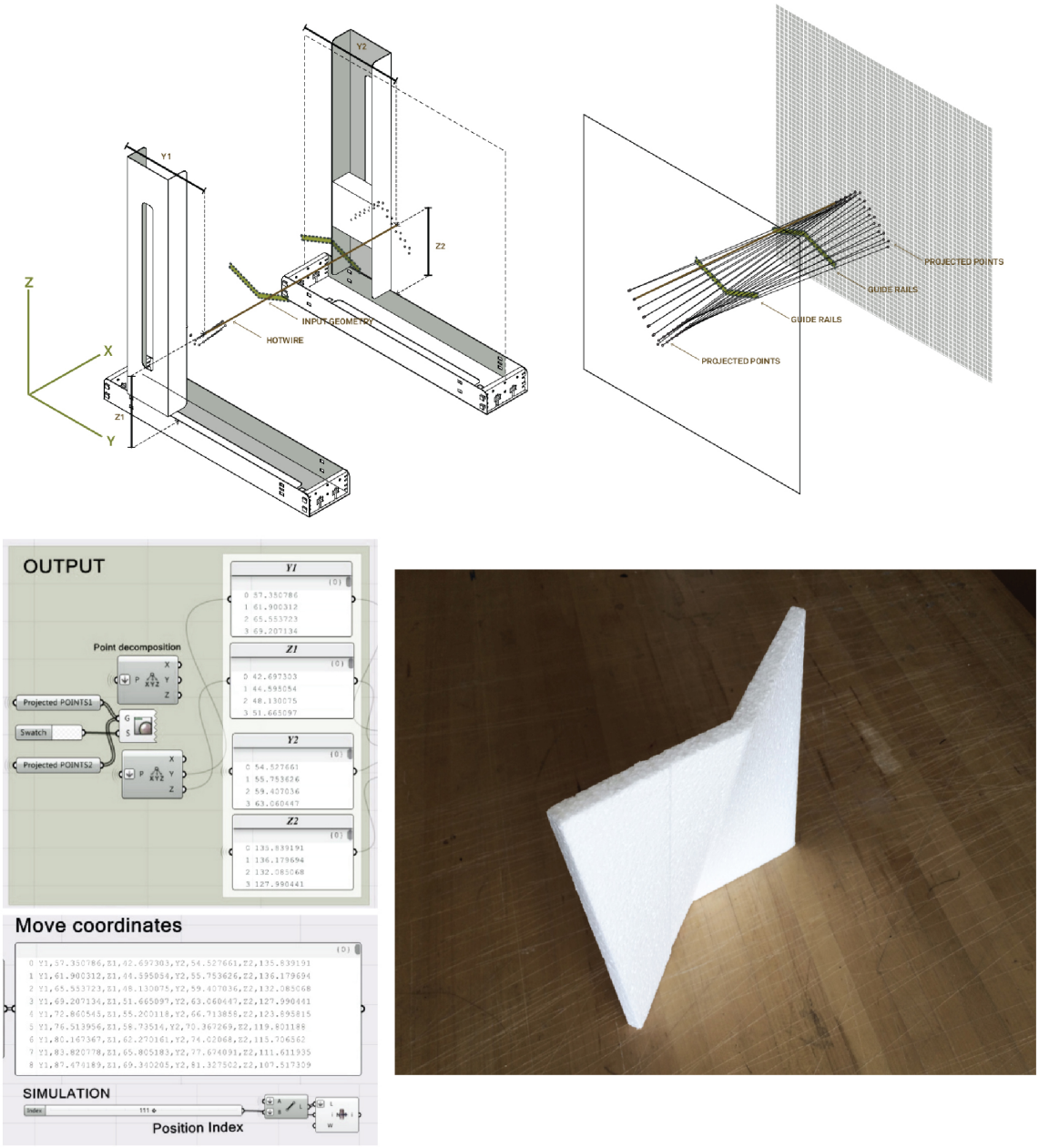


Figure 4-8: Toolpath planning for a four degree of freedom hot wire cutter. The rails are specified on the surface of the material, such that their projections onto the plane of the end effector become the toolpath. The toolpath is calculated in Rhino with Grasshopper and exported as a csv file to be imported in to a virtual machine.

4.2.2 Interfacing with existing CAD/CAM software

There are many existing applications for generating machine actions–e.g. many different CAD/CAM packages. It would be remiss to assume that these cannot also be used in conjunction with networked controls. The virtual machines that we defined in the previous section can be called with commands from any application.

For example, a 4-axis hot wire cutter can cut foam pieces into complex ruled surfaces. To generate the toolpaths for these kinds of parts, the desired curve to be cut needs to be specified, as well as where in the material that curve is to be cut. If we make the assumption that the specified curve should lay on the surface of the material provided, we can extract the machine wire connection position required to make those cuts from the projection of the desired curve on the material plane onto the machine's wire connection plane.

These geometric operations (curve sampling, projections) are well-established methods in CAD software. Re-implementing these kind of methods in e.g. *mods* does not improve the output of our machine. Therefore, to make toolpaths for a hot wire cutter running on a PyGestalt network, we used the application Rhino and its data flow scripting language Grasshopper. We exported the toolpaths as comma separated value files and used them with a python script that imported the hot wire cutting virtual machine. These details are shown in Figure 4-8.

Much of the work that goes into digital fabrication is in the form of translating between applications through specific kinds of imports, file formats, or modification scripts. To be an expert at dealing with these kinds of issues one not only needs to understand in depth how to run a machine tool, but also how deal with software and

computers. These are often invisible skills, an experience-learned knowledge. This section detailed how we interfaced with Rhino, which is rather extensible and flexible as far as software tools go. However, in the future, our hope is that this elusive and often invisible labour of connecting software tools is routinised and eventually automated. By layering the components of the software stack more explicitly, it becomes easier to modify each piece independently.

4.3 Spawning control

How different nodes are connected is not only an attribute that is set by the software designer, but in many cases also a result of how the machine is wired and connected. It is conceivable to extract much of the information needed to describe a virtual machine from the node network interconnect on the physical machine. This way, a connected machine can self-report its configuration to applications. Since the machine description is contained in a virtual machine, once the application receives a virtual machine it can immediately use it to execute actions.

A community contributor has developed a graphical user interface for auto-generation of Python virtual machines; it walks the machine developer through a set of questions about the machine and how it is connected to be able to do this. More details are covered in Chapter 7. That is one of the first steps in the direction of spawned controls, but more work on this remains to be completed.

Chapter 5

Modular Machines

Digital fabrication machines are often characterised by how many degrees of freedom they have. A 3-axis mill is less complex than a 5-axis mill but cannot produce parts with undercuts or overhangs. Why can we not extend the capabilities of mechanical machines? Networked distributed controls make it possible to extend the electronics of a machine tool with new capabilities on the fly. Why are similar extension interfaces not built into the mechanics of machine tools?

The wire EDM in our lab has four degrees of freedom, meaning that the wire feeder on the top can be controlled in X and Y and the wire receptacle on the bottom can independently be moved in X and Y. This enables us to make shapes such as a square cut on the bottom of a piece of metal stock with a circle cut into the top. With more degrees of freedom, a machine tool is able to make objects with more complex geometries.

To add capability to our wire EDM, two researchers from the Center for Bits and Atoms, Will Langford and Sam Calisch, decided to try to add a parasitic 5th degree

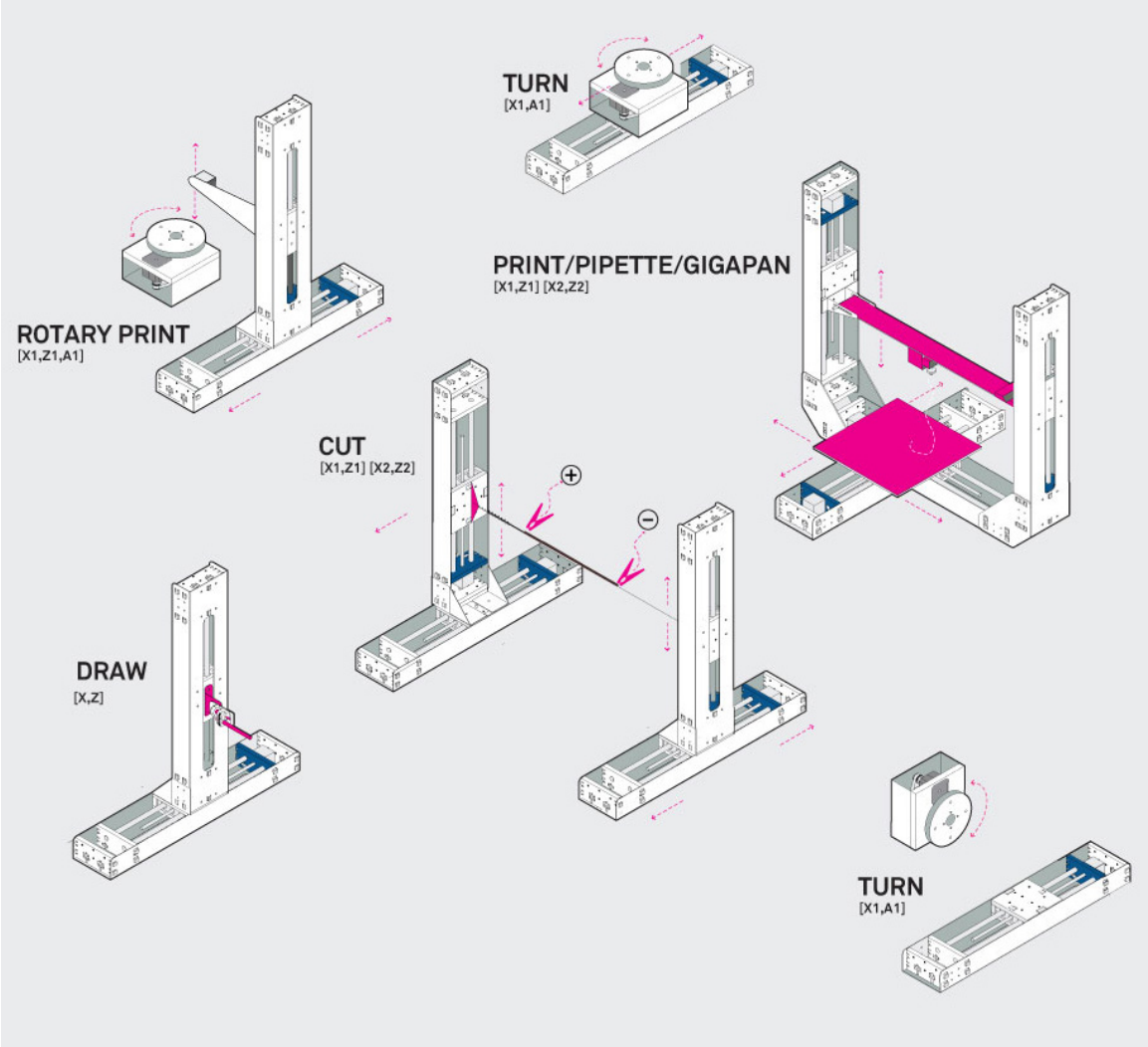


Figure 5-1: Using two types of modular mechanical components for machines, we can quickly assemble many different instantiations of machines. These are all machine instantiations that have been tested over the course of this project. Although you can quickly build up an XY stage with independent linear modules such as is done in the DRAW machine above, it can be beneficial to use parallel kinematics instead. We have therefore also often used a *CoreXY* stage as a machine module that provides XY motion with much less moving mass [55].

of freedom to the wire EDM. This would allow them to cut helical shapes (useful for making couplings for example). Making the rotary degree of freedom was easy—they built a rotating chuck that could hold a piece of cylindrical stock. But when it came time to tie it into the rest of the machine's control system, they were presented with difficulties. Simply put, wire EDMs do not move at a fixed speed, but change their velocity based on the resistance they encounter in the sheet stock. To be able to move at the same speed as the rest of the control system, the control system needs to report out where it is at that point. Will and Sam ultimately used a webcam with optical character recognition focused on the numerical readout on the screen of the machine's position to be able to move the 5th axis in time with the rest of the system. This was not a particularly robust or easy solution and ended up not getting used beyond the initial cut tests, although they were quite successful.

If the control system could be made extensible by introducting distributed networked control, as described in Chapter 3, then extending machines mechanically such as Will and Sam tried to do will become a much more common effort. Let's assume that interconnectivity for networked controls will happen. How can we make it easy to extend machines mechanically? How can we build up a machine tool out of individual degrees of freedom?

To start, what if we decomposed all machine tools into linear or rotary degrees of freedom, and built up machines with those building blocks? Using linear motion and rotary motion we can reproduce almost all machine tool motion requirements (see Figure 5-2). We can move both the material that is being worked on as well as the end effector doing the work. By using resuable building blocks we could rapidly assemble many different instantiations of machines. This would enable quick testing of different machine configurations.

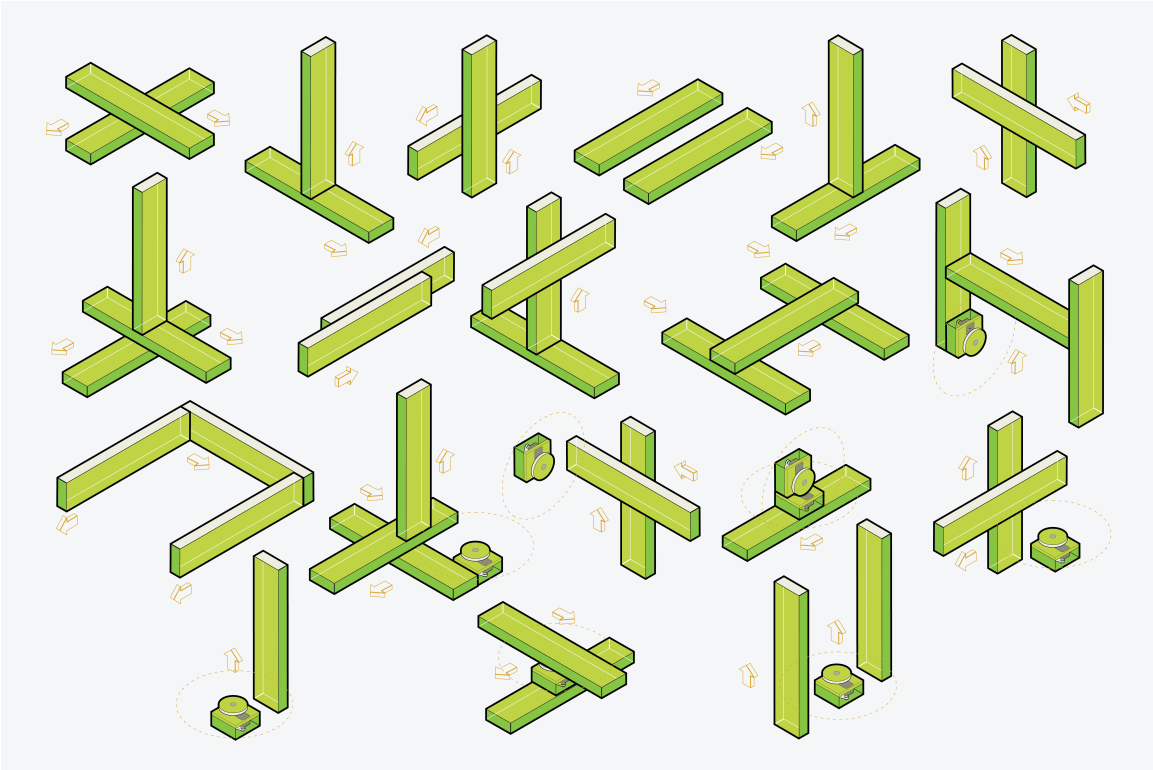


Figure 5-2: Configurations of linear and rotary modular machine parts that can make up machine instantiations. Two linear stages might make up an XY stage, or an XZ stage. Adding a rotary stage can introduce a polar coordinate system. Any assembly of machine parts can form a particular machine instantiation, be it with 1, 2, 3,...n degrees of freedom.

There are clear mechanical drawbacks to using modular stages that couple and decouple as machine tools. Such an assembly of parts will never be as stiff as a custom-built monolithic machine. Stiffness is required in machine tools to withstand cutting force with minimal deflection and vibration at the tool tip. Connections between parts might be particularly susceptible to vibration or introduce backlash. Using single-degree-of freedom modules implies the use of serial kinematics, meaning that stages are stacked on top of each other. This means that the X axis needs to carry the weight of the Y axis on top of whatever machining forces the machine might encounter. This has implications for the rate at which the machine can accelerate and also how slop in the system might compound.

These kinds of concerns are all valid. Precision and accuracy of a machine tool are important evaluation criteria. Parallel kinematics are often a more appropriate choice for high-speed motion requirements. However, we are concerned with a domain of machine building that is barely populated by *any machines at all*. How can we have the precision and complexity that automation affords in more domains than the digital fabrication tools (milling machines, laser cutters, 3D printers) we know now? What are ways in which we can encourage rapid prototyping of rapid prototyping machines? Some criteria to evaluate with include:

- Quick start: How long does it take to go from nothing to the first testable prototype?
- Accessibility: Can inexpensive and/or reusable parts be used? Is machine design domain expertise required?
- Diversity: How easy is it to incorporate different end effectors, sensors, or actuators? Can a wide range of processes be automated?

- Modification: Can the system easily be iterated upon? Do new iterations require a new machine to be built from the ground up?

These criteria—ease of start, accessibility, diversity, and ease of modification—make up a set of requirements for a machine building infrastructure that enables rapid prototyping of rapid prototyping machines. Ideally machine modules could incorporate the interoperability, interchangablility, and interconnectivity of distributed control networks. Legacy machines could be modules within an encompassing system (such as the wire EDM together with its parasitic 5th axis).

The drawbacks of perfectly stiff, low vibration, high accuracy machines lie not in the tools themselves, but in how applicable they are to the problem at hand. The need for machines that can mill aluminium is well met by the digital fabrication machine marketplace. Milling machines are very applicable to that problem. But what about a machine that needs to remove a petridish from a heating plate when its contents turn from blue to pink? Or a machine that needs to place stickers at the top right corner of a particular kind of package? Or a machine that makes helical cuts in soft materials? Or a machine to decorate cakes? These kinds of processes have not had access to the precision of CNC control. The cost of automation for these applications is traditionally far too high to justify.[1] Therefore, we are aiming not to replace well-established CNC tools such as mills for metal with modular parts, but to broaden the scope of machine tools to include anyone who is in need of repeatability and precision in their applications.

By making machines out of modular parts, we introduce more versatility through

[1]Many of these applications would be improved with the precision of automation, but the accuracy that is achievable by hand is often considered acceptable given the large cost increase of automation. One particularly salient example is that of the myriad biology lab workers who need to return to the bench at irregular hours to check on experiments or perform simple modifications.

reconfiguration, more robustness by making broken parts easy to swap out, and lower cost by reducing the design cost of each individual machine.

A similar argument was made during the introduction of packet switching networks, which served as the foundation of later protocols like TCP/IP. Some people argued that packet switching was a less optimal way to transmit data than dedicating entire connections between nodes to individual communications, as in the telephone networks of the time [69]. Later, TCP/IP was developed to create an "effective technique for multiplexed utilization of existing interconnected networks" [14]. TCP/IP, built on top of packet switching networks, enabled distributed heterogenous networking at enormous scale. A fundamental technology change gave rise to the proliferation of many different networks across the globe. Can we introduce a similar change in machine design?

We have developed several iterations of motion stage designs for modular machine building.[2] Here we will evaluate some of the versions developed for their versatility, robustness, stiffness, and cost.

5.1 Linear Modules

The first version of the stages had 20" of linear travel. They were made with 1/16th inch water jet cut and folded aluminium 6061 sheet stock (using a box brake) with milled HDPE parts. The motors were NEMA 17 stepper motors with 4-start teflon coated lead screws with a pitch of 0.05 inch. The guide shafts were 3/8" hardened steel

[2]The first iteration of the modular reconfigurable stages was built as a rapid prototyping exercise carried out in collaboration with researchers at MISiS University in Moscow during the Fab Lab 1.5 workshop held in October 2013.

Figure 5-3: Two linear and one rotary stage, made of 1/16th aluminium and 1/2 inch HDPE.

with nylon bearings. The stages have a hole pattern on the sides, bottom, and motion platform so they can be coupled to other stages in several standard configurations. These include with the second stage's back in the middle, or the second stage vertical on either side of the second stage (see e.g. Figure 5-3). Coupling pieces (for spacing) for attaching the stages to each other were made with HDPE as well. To maximise the holding surface area, we made custom fasteners, including rectangular nuts for stronger t-slot construction.

The stages were quick to couple and decouple and moved well. We connected each stage to a PyGestalt node as described in the previous chapter. However, we encountered issues with the stiffness of the folded aluminium. In several configurations, especially with a stage cantilevered off of another, there was play in the system that resulted in loss of accuracy.

To mitigate this issue, we built a second version of the stages with a wider motion platform and thicker sheet metal. We used a hydraulic brake on water jet cut 1/8" 3031 aluminium. We used the same motors, guide shafts, and fasteners. The moving platforms and ends of the stages were again made of milled HDPE. We updated the hole pattern so we could use three points of contact in any configuration of coupling. The resulting stages were very stiff and their couplings were more robust, but at the cost of extra weight. Instead of cantilevering stages, we also made idler wheels for supporting the cantilevered edges. We also made assemblies with redundant stages, e.g. two stages for X motion supporting a Y stage as a bridge.

To keep weight down but improve the stiffness, we built a series of stages with 60cm of travel out of a composite material of laminated polycarbonate and aluminium. We used NEMA 17 stepper motors with integrated 4-start 2mm pitch lead screws and

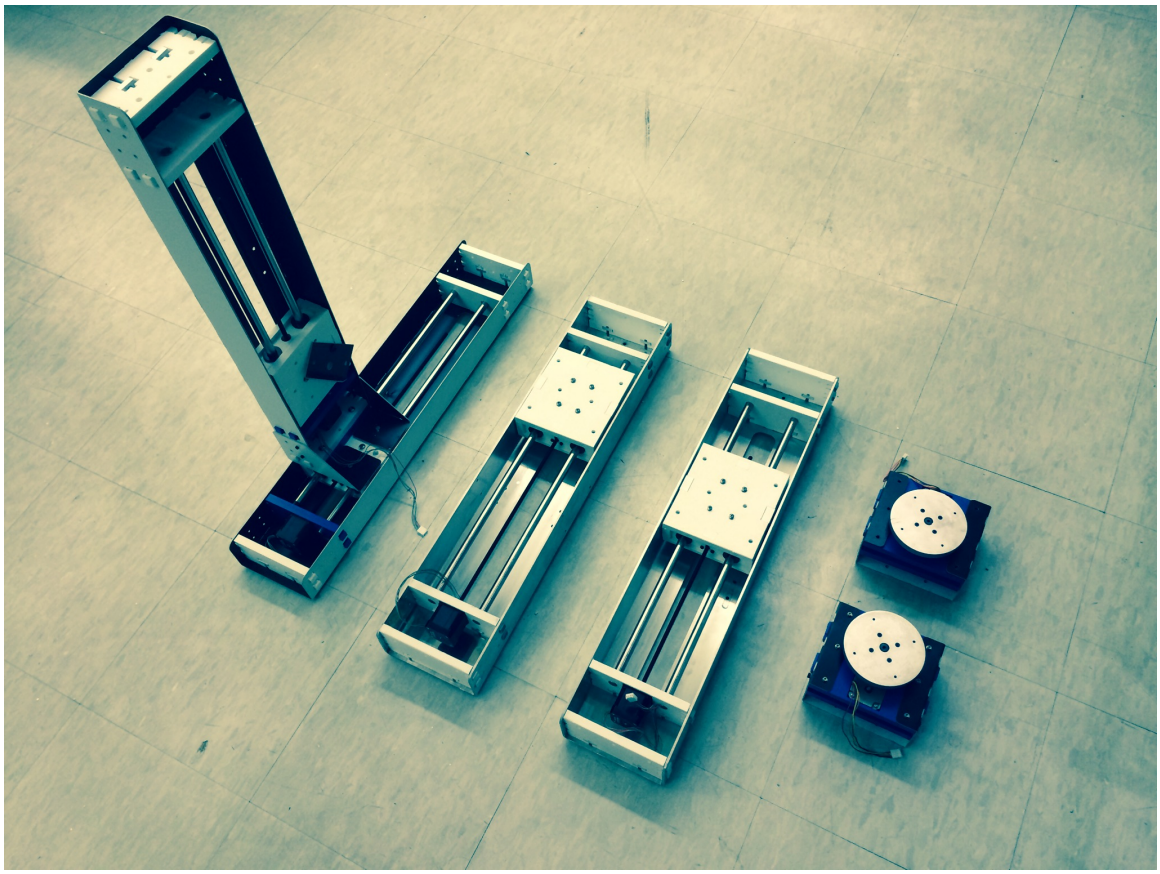


Figure 5-4: Two versions of linear stages, one with 1/16th" aluminium (left, XZ assembly), and another with 1/8th" (middle two). On the right, two rotary stages made with 1/6th aluminium and 1/2" milled HDPE.

3/8" hardened steel guide shafts and nylon bushings. To fold these stages, we scored the material using a v-groove end-mill, creating a crease that could be folded without the mechanical advantage of a brake. These stages were entirely cut on a CNC mill without a water jet cutter. However, the size and vacuum bed level tolerance we needed meant that we used a mill that was about as expensive as a water jet. However, this did remove the requirement of a hydraulic press brake.

5.2 Rotary Modules

Linear stages were the first modular mechanical parts we developed. However, it was immediately apparent that we would need rotary modules as well. (Robotic) arms rarely use any linearly translating elements at all; they more commonly use a stack of rotating elements. Stacking rotary elements does mean that you might end up with a precariously cantilevered end effector. Calculating the motion trajectories for end effectors is also more complex, as there is typically more than one way an end effector can be placed in the same position (imagine the 'elbow' could be bent to the left or to the right to keep the 'hand' in the same position). The range and geometry of moves that you can execute is therefore also very rich.

To make a rotary stage, we used MXL timing belts connected to a NEMA 17 stepper motor with a drive pulley to rotate an aluminium platform. We used a 3D printed MXL pulley to attach to the platform. For the stage body we used the same 1/8" 6061 aluminium and milled HDPE [3]. We used steel taper bearings as a contact surface.

[3]The HDPE we use is the brand Starboard. We have found that it is significantly stiffer than other types of HDPE we have used, and is consistent and reliable in its tolerances and material properties.

The rotary module platforms are tapped to accommodate the attachment of other stages or end effectors. They are shown in Figure 5-3 and Figure 5-4.

5.3 End Effectors

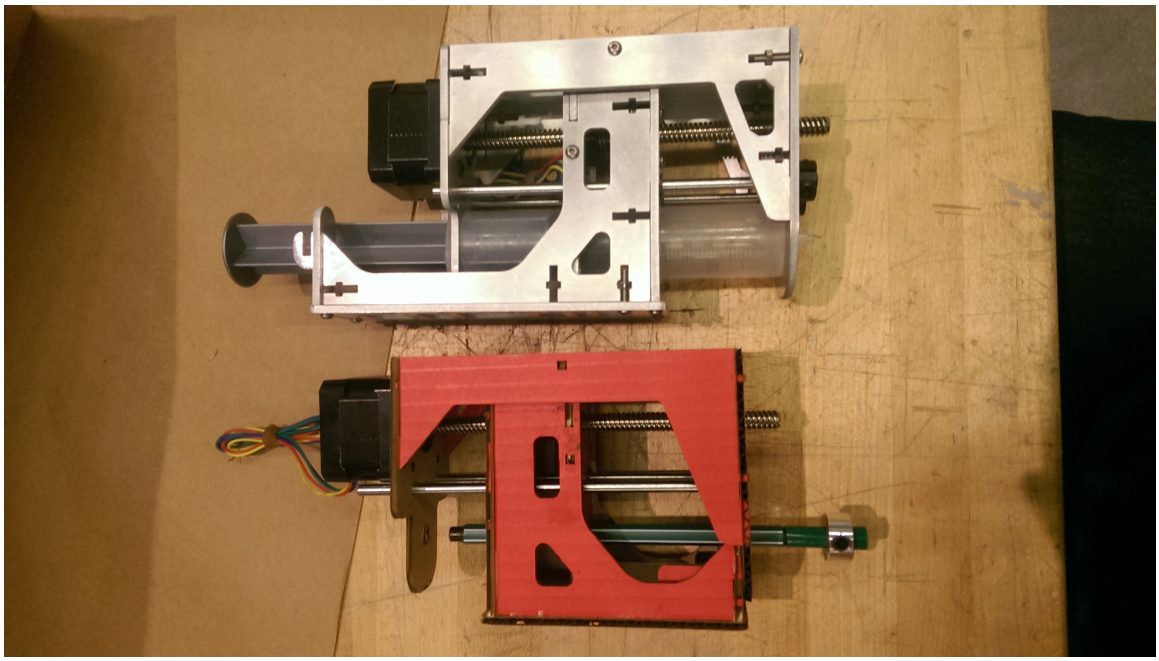


Figure 5-5: A syringe pump end effector made out of aluminium next to a spring-loaded pen holder made with cardboard. Both use the same motors, one for pressing the syringe plunger, the other for pen up and pen down.

The end effector is the part of the machine that will be interacting with the material that is being machined. Common end effectors include lasers for laser cutters and stereolithography 3D printers, fused deposition extruders for 3D printers, spindles for mills, knives for cutters, syringes and pumps for liquid handlers, et cetera. Each of these end effectors has cutting forces associated with its use (e.g. high for milling steel, low for moving a lens around to focus a laser) as well as weight, power, and data requirements.

We have developed several end effectors for using with machine the modules (see Figure 5-5), including nichrome wire with flexible attachment, syringes, extruders, spindles, and (oscillating) knives. To mount these end effectors on our machines, they include the same hole mounting patterns as the stages. Their controls can be connected to the fabnet network that the motion stages use. Diversity of machines comes in part from diversity of end effectors, so their standardisation is limited to how the end effectors connect mechanically to the motion modules and electronically to their control networks. More detail on what kinds of end effectors we have encountered will be given in Chapter 7.

5.4 Rapid Prototyping of Rapid Prototyping Machines

The modules shown here, along with the distributed controls described in Chapter 3 and the software described in Chapter 4, make it easy to quickly assemble automatic processes. I have configured modules into many different assemblies, each its own machine.

Figures 5-6 and 5-7 show two different hot wire cutters, one with four degrees of freedom and another with a thirty-six inch cutting span. The latter machine was quickly assembled one afternoon when a fellow student realised he needed an airfoil with a reference geometry to test in the wind tunnel during his allotted time slot, which happened to be later that day. We made the machine, made the part, and got the data. The reference geometry airfoil is shown on the left, and the experimental wing shown on the right in Figure 5-7.

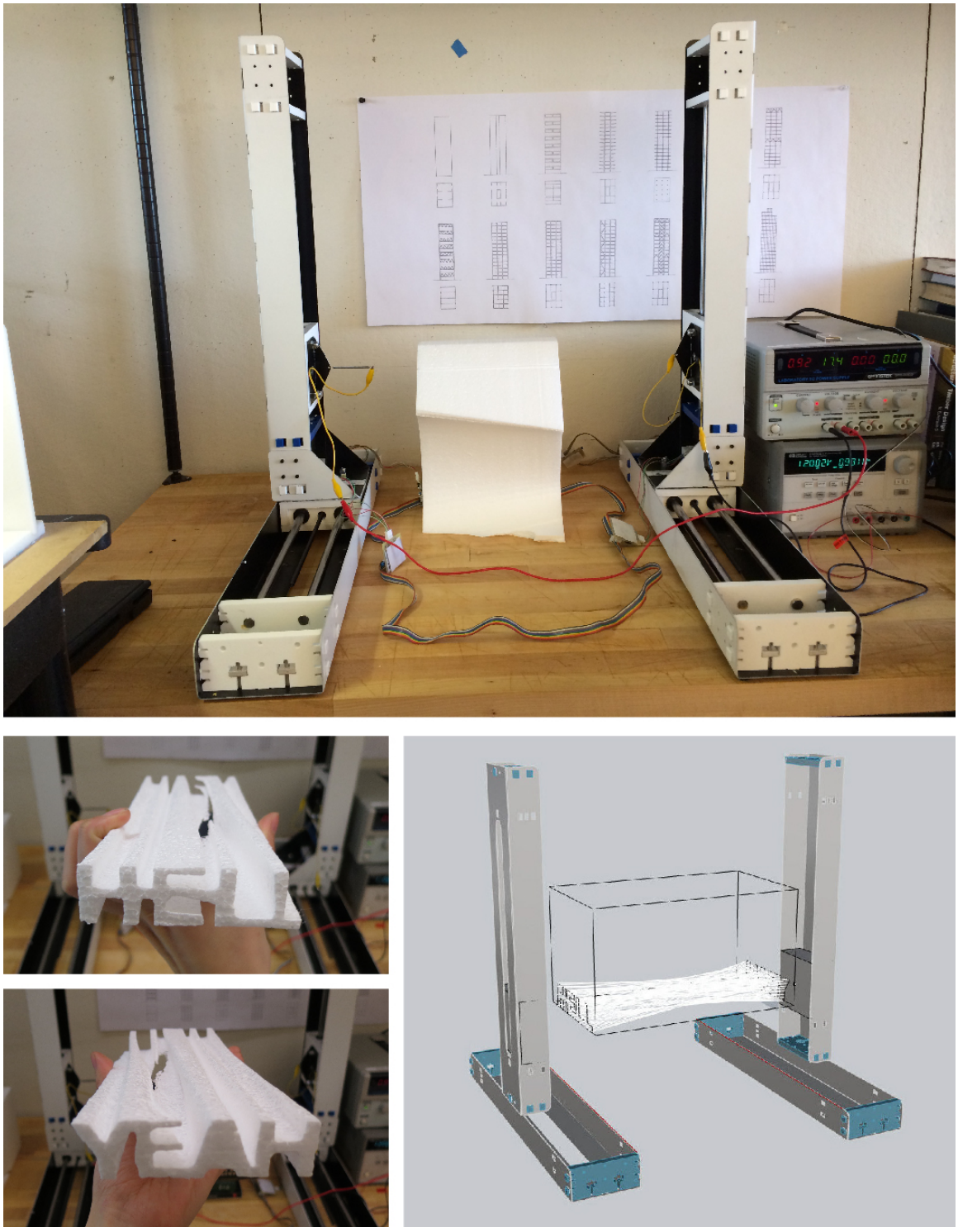


Figure 5-6: This is a 4-axis hot wire cutter named *Slashbot* that uses 4 linear stages and nichrome wire mounted on springs to cut foam. The machine's control network runs on PyGestalt, and we used Rhino/Grasshopper as a CAD/CAM package for generating actions.

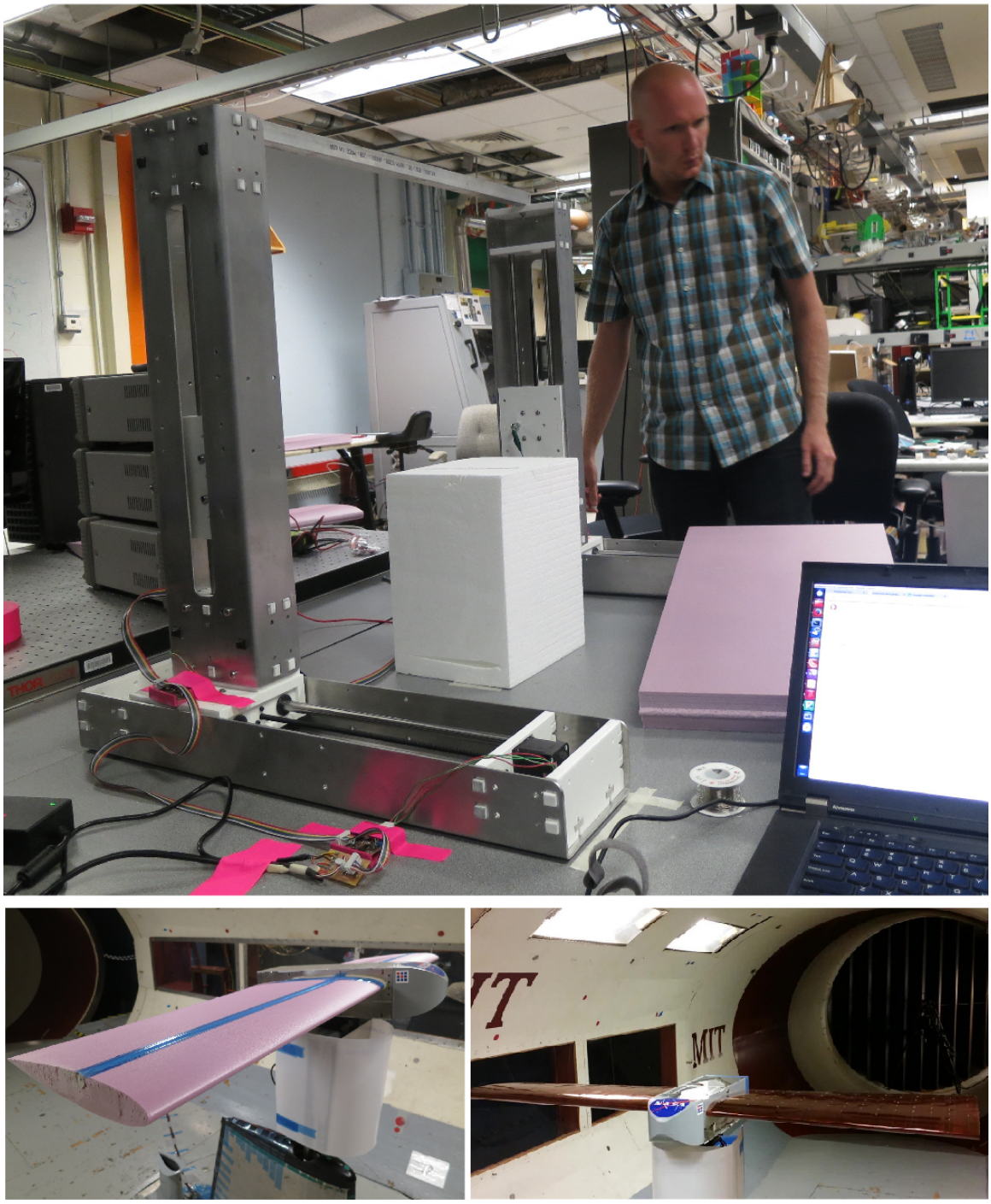


Figure 5-7: A hot-wire cutter with a thirty-six inch span. This hot wire cutter used a later iteration of the mechanical stages connected with aluminium cross bars. This means that it only had two degrees of freedom; we did this so that we could apply much more tension on the nichrome wire so that it would cut parts with a thirty-six inch span without the drag in the middle causing geometric imperfection.

Figure 5-8: This is a machine that uses a rotary stage, a high-resolution camera, PyGestalt stepper nodes, and the *mods* workflow scripting environment to automate image capture. This imports a *gphoto2* module as well as a PyGestalt VM into *mods*.

One of our collaborators was interested in meticulously recording the colour information of glossy objects. Unfortunately, specular reflections make it hard to get the colour of an object. Figure 5-8 shows the setup we created with a rotary module and a *mods* workflow for automating image capture using the camera control library gphoto2.

These are only a few examples of the machines we were able to rapidly configure using the infrastructure presented thus far. We will go into more detail on machines others have created with these tools in Chapter 7.

5.5 Tool-users versus tool-makers

Breaking machines into modules that can be assembled might seem like a semantic distinction from current common practice in machine design and production. However, I argue that it is a strong redistribution of power into the hands of the tool-user. If the wire EDM that Will and Sam attempted to modify was built out of modular machine parts, it would have been trivial for them to add a fifth degree of freedom and thereby expand the number of parts they could build. The company that makes the wire EDM has a model of the machine that includes an additional rotary degree of freedom, but that machine is almost twice the cost. Furthermore, and perhaps more tellingly, it is export controlled by international arms traffic regulations. A 5-axis wire EDM is considered a type of weapon, and access to that level of fabrication capability is worth of government control.

It is currently difficult to build complex machines such as a 5-axis wire EDM. Perhaps someone will make an argument that access to precision fabrication should be

limited to those we can trust to not build weapons. Some such arguments were made after the first working guns were made on a 3D printer[4] [43]. I would provide three counterarguments: that people who want to make precision parts will find a way to; buying weapons is still very easy (at least in the US); and limiting a country's access to precision manufacturing is going to damage national security much more in the long run. I believe that it is the technology paired with the person and their intent that we must be concerned with when creating legislation. Not having the infrastructure for widespread access to precision and complexity in manufacturing is ultimately a dangerous state of affairs.

[4]Many more of the arguments on the 3D printed gun debate centered around whether or not CAD files could be considered protected speech, and a government takedown notice for the design of the 3D printed gun *Liberator* was in violation of the First Amendment. In response, bills were introduced which would incriminate the act of printing a gun instead. Unfortunately, these kinds of regulations are all but unenforceable.

Chapter 6

Object-Oriented Hardware

Our approach to networked controls, software interfaces, and mechanical machine components all have modularity in common. The layers of machine building shown in Table 6.1 are covered by the example implementations described by the previous three chapters. An approach could be to continue tightening up these implementations and to try to make them general and universal.

Machine Layer	**Example implementations**
End effector	spindle, knife, laser, extruder
Mechanical system	rack and pinion, guide shafts, rails
Sensors, actuators	motors, hydraulics, encoders, end stops
Control system	pygestalt nodes, g-code interpreter
Applications, interfaces	CAM software, machine controlller

Table 6.1: Interconnected systems needed for machine design.

In developing this research, I have been deeply influenced by thinking that has gone into network design, including the Open Systems Interconnect Model (referenced in Chapter 4 for the need to put *virtual machines* into their own layer). However,

the OSI model was eventually somewhat of a failure, with the layers 5 (session), 6 (presentation), and 7 (application) being only theoretically distinguished between. (These three layers are, for example, lumped together the Internet protocol suite (TCP/IP) as application [22].)

While a clean separation between layers would make it much easier to develop layer-specific implementations and swap them out with improvements and modifications, in reality there are many things that can only superficially be developed independently. For example, the actuators of a machine need to be specified such that they can handle the loads that the end effectors and mechanical system impose. Each of the layers described in Table 6.1 is independently a very well established field, but the system integration is where most machine developement turns sour.

In thinking about the rapid-turn production of application-appropriate machines, with a careful eye cast specifically on system integration, I have come to the conclusion that incremental improvements on the techniques we use for machine design are insufficient. The pre-specified layers above impede our ability to provide infrastructure for unanticipated applications. Therefore, in this chapter I describe attributes for a machine design paradigm called *object-oriented hardware*, with which I aim to mitigate some of these issues.

6.1 Embodied Objects

In Sutherland's dissertation on *Sketchpad*, he uses the terminology 'master', 'object' and 'instance' to refer to different components of graphical design. This kind of object-oriented thinking was further developed by Alan Kay (who was a colleague

of Sutherland's at the University of Utah) and colleagues into the *object-oriented programming* paradigm that the language Smalltalk was based on [30].

A smalltalk object:

- has *state*, which contains references to other objects (in smalltalk, booleans or integers are also objects)
- can receive messages, both from itself and other objects
- can send messages in the course of processing a received message.

These attributes were generalised in later programming languages into *objects* having *data* stored in object attributes, and *code*, stored in functions.

A mechanical machine module paired with a networked control node, such as those described in Chapter 5, has state in the form of position, methods in the form of moves that can be executed, and a physical embodiment that can be connected to other modules. A *mods* module, such as described in Chapter 4, has state in the form of fields, methods in the form of functions, and an embodiment that can be connected to other modules in the *mods* message passing framework. Mechanical modules can be connected to other mechanical modules. *Mods* modules can be connected to other *mods* modules. But more importantly, a *mods* module can also be connected to a mechanical module.

All of these machine objects are peers, despite what machine building application layer they belong to. Instead of thinking about machines as a system integration project between different distinct layers of implementation, I propose we think of machines as a collection of such objects.

Object-oriented hardware is an extension of the object-oriented paradigm back into physical space[1]. Instead of thinking about developing particular machines, we can think about developing *machine instantiations*, where each instantiation is denoted by the objects it uses.

6.2 End-to-End Machine Design

The widely influential paper on internet architecture *End-to-end arguments in system design* provided a set of arguments for implementing functionality at the endpoints of a communications system [71]. The primary example given was of assuring accurate and reliable transfer of information through a network. If any network subsystem attempted to do an intermediate data verification, the data could be corrupted after the subsystem check and still arrive corrrupted at its destination. The paper argues specifically for not implementing any functionality within the network if not strictly necessary, arguing that this will:

- reduce the complexity of the core network, making upgrading the network later easier
- make the network more general, in anticipation of unexpected applications which might run on the network
- increase the reliability of the network, as applications do not have to rely on application-specific implementations in the network to function correctly.

[1]The metaphor deployed by Object-Oriented Programming was that of objects in physical space interacting, so it seems now particularly fitting to unite the two realities

These design principles have served the internet well for the implementation of applications such as the World Wide Web and email (they are being called into question now that the internet is ridden with more demanding applications with service differentiation and a general untrustworthiness of infrastructure [6], but this aside). Are there lessons to be learned here for implementing machines instantiations as well? The internet is implemented on very heterogenous infrastructure, yet manages to provide end-user functionality in a reliable fashion [82]. How can we apply what was learned in network engineering to machine building?

I believe that up until now, machines themselves have largely been considered infrastructure, and the things that were made on those machines were considered applications. Instead, I believe that machines together with their output *are* the applications. The infrastructure consists of objects that can be employed to make machines. I propose to call this paradigm of machines as applications *end-to-end machine design*.

End-to-end machine design implements the functionality of the machine as much as possible in the end points of the machine network. If we analyse machine tools historically, the end points are often design intent, codified by a CAD file, and material forming, executed by a general purpose CNC tool. However, if the purpose of a machine is to, for example, repeatedly stamp the surface of metal sheets with the same design, the end points might be the stamp design and the feed rate of the metal sheets. Instead of creating a machine that implements as many applications as possible, end-to-end machine design research focusses on building object-oriented hardware as infrastructure and making machine instantiations as applications.

In the future, we could ideally produce an adequate machine instantiation from high level instructions. By using objects as in object-oriented hardware, it would become

easier for the machine to:

- reconfigure if process requirements change
- automatically discover errors and then automatically solve problems
- (and therefore) support new and unanticipated applications

To take this thought a step further, I'll sketch a future scenario for making machines that make:

Imagine a designer beginning to design a product's geometries, sub-assemblies, and materials. Each time a new feature is added to the design, a simulation of a machine that would output the product is shown in real time. As the design is developed, the machine evolves as well.

In the process, the simulation of the machine can show the designer details about the complexity of the machine required to output the design. These constraints of production (including rigid body simulation of the machine, modelling of the materials and processes, and many other things that need to be sped up from their current states to make this a reality) can thus be combined with end-user intent.

This scenario is still science fiction[2]. But exposing functionality to the end-user is something that is currently poorly done, and will continue to be poorly done no matter how many incremental improvements we make to our current machine design paradigm.

[2]But I would argue not the science fiction of "the perpetually impatient and somehow perpetually unworldly futurist, seeing his model going terminally wrong in the hands of the less clever, the less evolved", but a fiction that places technological capacity in the hands of the tool-users, and involves all in the further development of infrastructures and their narratives [27].

I argue that employing object-oriented hardware in end-to-end machine design will make it easier to rapidly create machine instantiations. To test this, I have reduced some of the infrastructures for machine objects to practice (networked controls, workflow composition software, and mechanical modules) and tested them with a broad base of participants. The results of this experiment are described in the next chapter.

Chapter 7

Cardboad Machine Construction Kits

The linear and rotary modules described in Chapter 5, together with the networked controls and software described before them, simplify assembling machines for automation. However, these technologies are not universally available. While their designs are freely available, they require access to manufacturing tools and manufacturing expertise. To test if object-oriented hardware simplifies rapidly prototyping rapid prototyping machines, we need to test their use with many more users. To be accessible to more users, we need to have a deployable design. Rather than mass-manufacturing the electronics and stages we described in the previous chapter, we sought to make the modules more accessible in other ways. We wanted an easy to make, easy to modify, easy to use, low cost version of the modules we had. To do this, we developed cardboard versions of the linear and rotary modules to be used with the PyGestalt stepper boards.

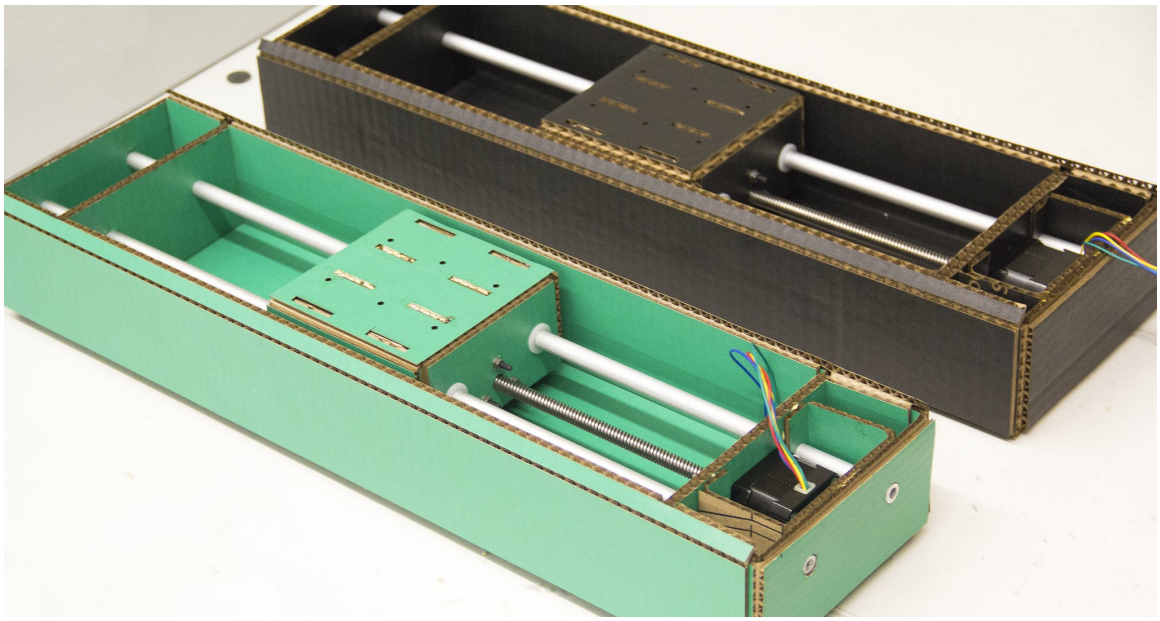


Figure 7-1: Linear stages made out of cardboard. They use nylon bushings, aluminium tube guide shafts, and stepper motors with integrated lead screws. They have 30cm of travel and three standard coupling configurations.

Cardboard is a very democratic material—everyone knows how to modify it. Tools for cardboard construction can be as simple as a pair of scissors and some glue. Making machines out of cardboard builds an affordance for modification into the machine itself. Furthermore, cardboard is globally available and globally inexpensive. Due to corrugated cardboard's ubiquitous use as a packaging material, it is readily available even in the most remote locales. Finally, cardboard is a strong and lightweight material. Its popular use as a packaging material is due to it being resistant to abrasion, impact, and crushing while not adding much weight to shipped goods.

To make sure machine builders also had access to the same control system and software we used, we wrote extensive documentation, example code, and tutorials. The development of all of these systems was already open-source and freely available online, but without adequate documentation, the source files would be effectively useless.

We learned from the difficulties people had reproducing the *MTM Snap*—which often involved a lack of details that might be obvious to people who already have domain expertise, but not to the intended audience of the design. To avoid this we documented the kit extensively during development, focusing not only on how we made it but also on why we made the design decisions we did.

To introduce the cardboard construction kits, we taught a series of machine building workshops[1]. Others groups subsequently taught the curriculum using the same lecture notes and kit materials. The workshops combined resulted in several hundred machines.

7.1 Cardboard Machine Modules

Laminating corrugated cardboard in layers significantly increases its stiffness. This is well exploited in the design of packaging and signage. The stiffness can be further improved by alternating the rotations of the corrugations.

The cardboard stages' bodies are made up out of 2-3 layers of corrugated cardboard, glued together with wood glue. For the stages we specifically used the coloured cardboard that is sold as tri-fold poster presentation displays for science fairs. This cardboard is painted, which slightly improves its susceptibility to water. One linear cardboard stage is made out of one tri-fold poster board, or approximately 12 square feet of cardboard.

[1]The workshops on machine building with cardboard construction kits were partially inspired by a workshop I ran with Ilan Moyer at the Fab10 conference in Barcelona. The Barcelona workshop was on prototyping machines using foamcore and a petting zoo of hardware parts. In three days, workshop participants were able to make almost-working machines, which I believe was especially enabled by the constant, enthusiastic, and penalty-free modification of the foam core structures. Subsequent workshops I taught with James Coleman provided a baseline working machine element for iterating upon, so that all participants could make at least something that worked.

The stage parts are cut using a laser cutter. They could also be cut with a blade, but using a laser cutter is speedy and avoids crushing the cardboard while cutting. To ensure the accuracy of the folds, we pre-scored all the folds in the laser cutter as well. The scoring cuts through the first layer of cardboard including part of the corrugations. This ensures the fold stays consistent through the length of the cardboard.

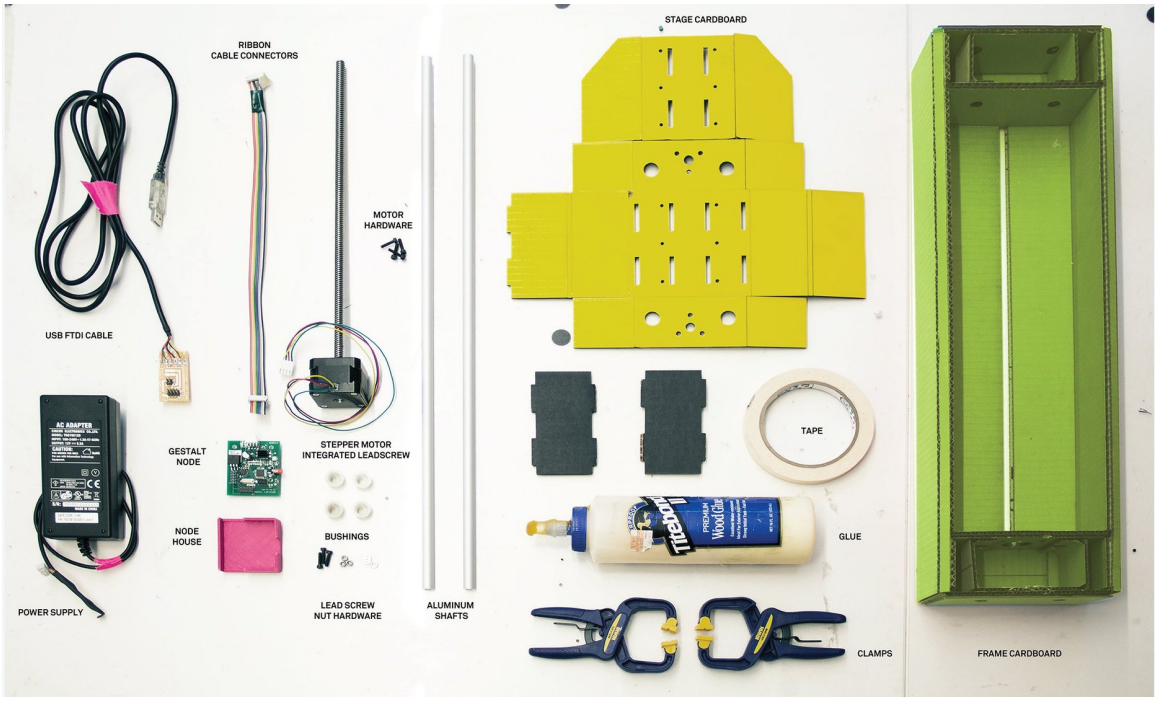


Figure 7-2: The parts for making the linear cardboard machine monomers, including bushings, guide shafts, stepper motor, control nodes, cabling, and power supply.

The cut files for the linear stages are parametrically designed in Rhino with Grasshopper. Users can vary the thickness of the cardboard they use, the width, and length of the stages, and export the cut files as DXFs. We also provided a reference cut file as a DXF for users who were using the tri-fold cardboard in the BOM, or who perhaps did not have access to Rhino software. The reference design is meant to be cut with a whole 36x24" sheet at once, but we also included dovetail cuts, breaking up the parts for users who have smaller laser cutters.

description	**quantity**	**total price**
Tri-fold Poster Cardboard	1	4.16
Nylon bearing, flanged, 3/8" ID	4	2.93
Aluminium tube, 3/8" OD	4'	9.26
Stepper motor nema17 30cm leadscrew, w/nut	1	27.45
M3 fasteners 12mm	3	.30
M3 fasteners 8mm	4	.40
M3 locknuts	3	.30
Gestalt stepper motor board	1	15.20
total		**60.00**

Table 7.1: Bill of materials for a cardboard linear stage module. A more detailed BOM with vendor information and part numbers is kept up-to-date at `mtm.cba.mit.edu`. This BOM includes the electronics for the stage. To run the stage (or a number of them), the user also needs a USB to Rs-485 cable and a power supply.

For the bill of materials, we limited the number of parts to eight line items (see Table 7.1). Having a short BOM makes sourcing the required material significantly easier. To achieve a short BOM, we used a stepper motor with an integrated leadscrew. These are available off the shelf, but with variations and unreliable stock levels. To accommodate the number of workshops we anticipated running, we placed a custom order for stepper motors through *AQS* Inc. in Dongguan, China. We ordered bipolar NEMA 17 1.8 degree stepper motors, 6VDC, 6ohms, 37g.cm2, 1A, 12mH, with a 4-start lead screw and 2mm pitch (TR8x8), with an ABS wear-compensating lead screw nut. More details on sourcing are given in appendix D. The lead time for the motors was about three weeks, and the cost per motor was under 28 dollars, and the minimum order quantity was 10 motors. For comparison, a wear-compensating lead screw nut alone from McMaster-Carr is 32 dollars, and an integrated lead screw motor we had quoted from an Connecticut based motor manufacturer was 140 dollars per motor without nuts, with a lead time of 12 weeks and a minimum order of 30 units. If we used an off-the-shelf lead screw and couplings to connect to the motor,

we'd need a helical coupling on the motor side, and to restrict axial motion on the lead screw, shaft couplings and thrust bearings on the idler side. In total there would a total of seven more line items, with an estimated cost of at least 100 USD.

Sourcing materials is a delicate optimisation task, balancing value, availablity, and design intent. Sourcing materials for open source hardware projects is compoundingly difficult because the projects need to be reproducable by many different kinds of users with access to different markets. Some guidelines for how to build a BOM for an open source hardware project we have are:

- Re-orderable (not end-of-life, vendor without membership requirements)
- Re-placeable (provide sufficient detail so that a replacement can be identified)
- Re-usable (take advantage of existing economies of scale)

You can get stepper motors for free by harvesting them out of discarded inkjet printers or other e-waste. But the design will not be reproducible. Certain hardware parts sold by clearing house vendors might be cheaper, but they will not be available in the long term. Using parts that are standards or used in mass-produced goods (such as Formlabs using a BlueRay laser for curing resin in their SLA printers) lowers the risk of the part you are using becoming irreplaceable.

Like the metal versions of the linear stages, the cardboard stages are intended to be coupled in a variety of standard positions. To make this even easier, we included tabs on the motion platforms and corresponding slots on the backs and sides of the stages. These allow the stages to be stacked like Legos for a first motion configuration without needing any glue or tape.

The cardboard we use can handle loads of 200 pounds per square foot. The part of the cardboard that the leadscrew is attached to is approximately a square inch, and thus we can expect the cardboard stages to move loads of around one pound without deflection and backlash. This way we can still achieve high precision in motion without using an expensive material.

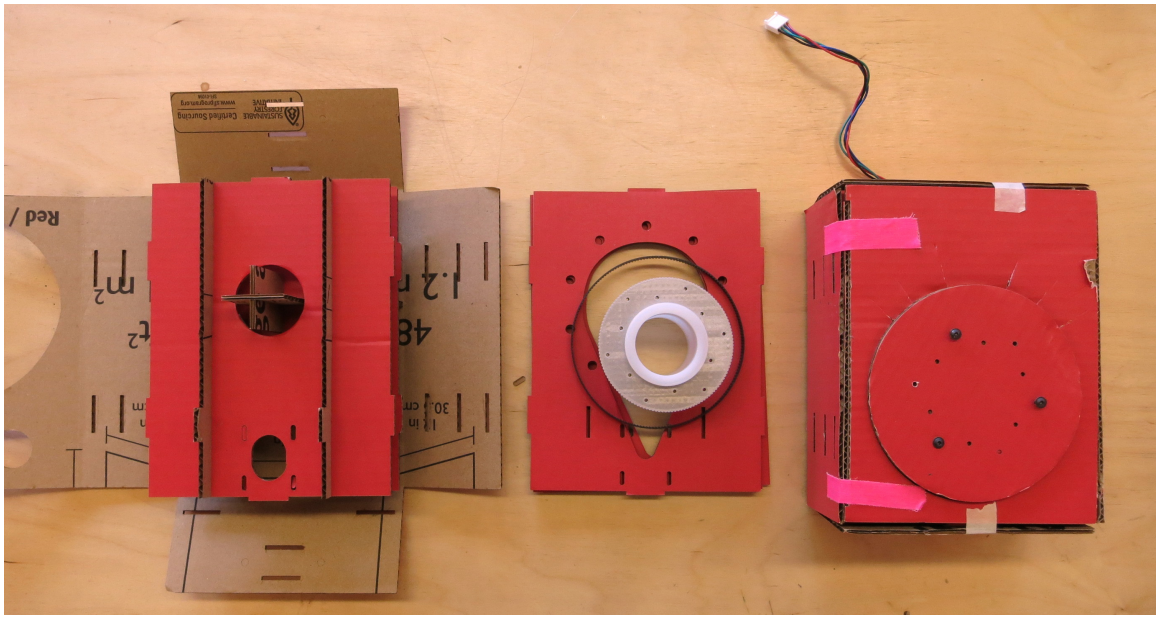


Figure 7-3: Assembly of the rotary stages, which use MXL timing belts, a nylon bushing as an axle, nylon ball bearings, and stepper motor.

The rotary stages use the same cardboard as the linear stages, but instead of a lead screw they use a timing belt and drive pulley as their drive train. The other parts are familiar—nylon bushing, NEMA 17 stepper motor. In the BOM for the rotary stage, we included a platform gear that needs to be 3D printed by the user. The print is simple and can be done on an inexpensive consumer 3D printer. A version was done in laser cut cardboard but was less robust and wore out quickly. This reference design for a rotary stage was added after many of the workshops described in the next sections were completed. Therefore, many of the machines developed in the wild

include custom rotary stage designs.

7.2 Assembly Instructions and Documentation

The documentation, assembly instructions, BOMs, design files, and source code are all linked to from `http://mtm.cba.mit.edu`. The first page provides an overview of the sub-modules and examples, with more detailed pages for each. The documentation includes step-by-step instructions for installing and running the software, the wiring of the control boards, the assembly of the cardboard stages, and full code for example projects to get started with.

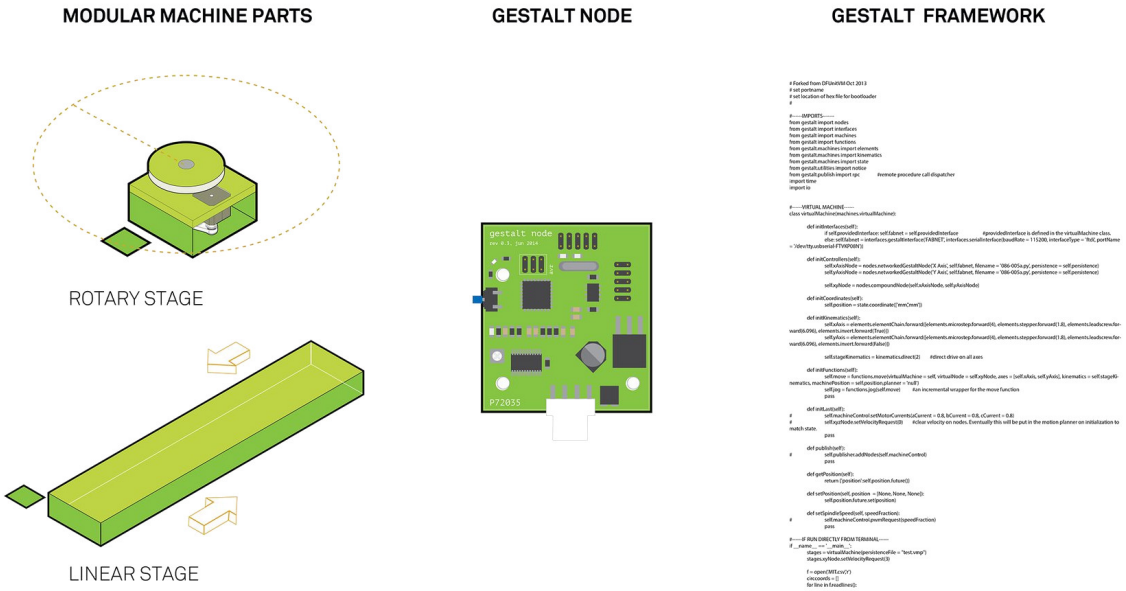


Figure 7-4: In the documentation, we tried to make system integration for the machines as clear as possible so participants could focus on their customisation instead of debugging communication.

Many of the source files are maintained on the website `github.com`, which also provides users with a venue to report bugs, fork designs, and participate in new iterations.

Because of this avenue of contact, we have been able to incorporate feedback on errors in the BOM, developed design iterations for people with smaller laser cutters, and clarify details.

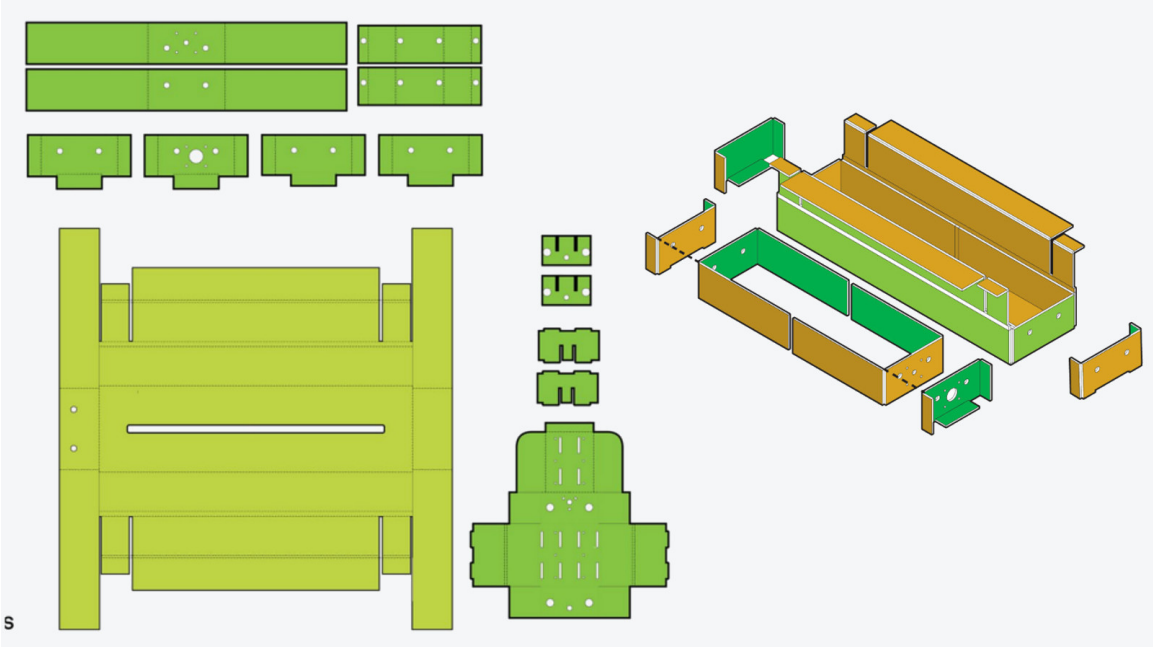


Figure 7-5: Assembly instructions we developed includes animated GIFs to explain how to assemble the parts.

System integration is one of the most important aspects of machine bulding. A working machine needs a viable software interface, a control system, and a mechanical structure. Without one of those parts, the whole machine is useless. In software engineering, it is easy to write a short test program and run it and then continue building on it. In machine design the approach has historically been for engineers to spend a long time perfecting a design, then begin prototyping after much of the design is complete. This takes a long time, does not leave much room for error, and makes mistakes much more costly. We instead encouraged machine builders to make the minimal viable machine they needed—including all subsystems—first. This

means iterations happen much faster, and potential issues with system integration are identified early on.

We also provide some documentation on how to build the next iteration of the minimum viable machines, including material iterations, different types of hardware, different types of controls, or different kinds of software interfaces. Because of the modularity of the system, any component can be swapped out without affecting the rest of the machine's functionality. In particular, we encourage reusing the drive train hardware (motors, bearings) and redesigning the machine frame either in form or material to better suit the application.

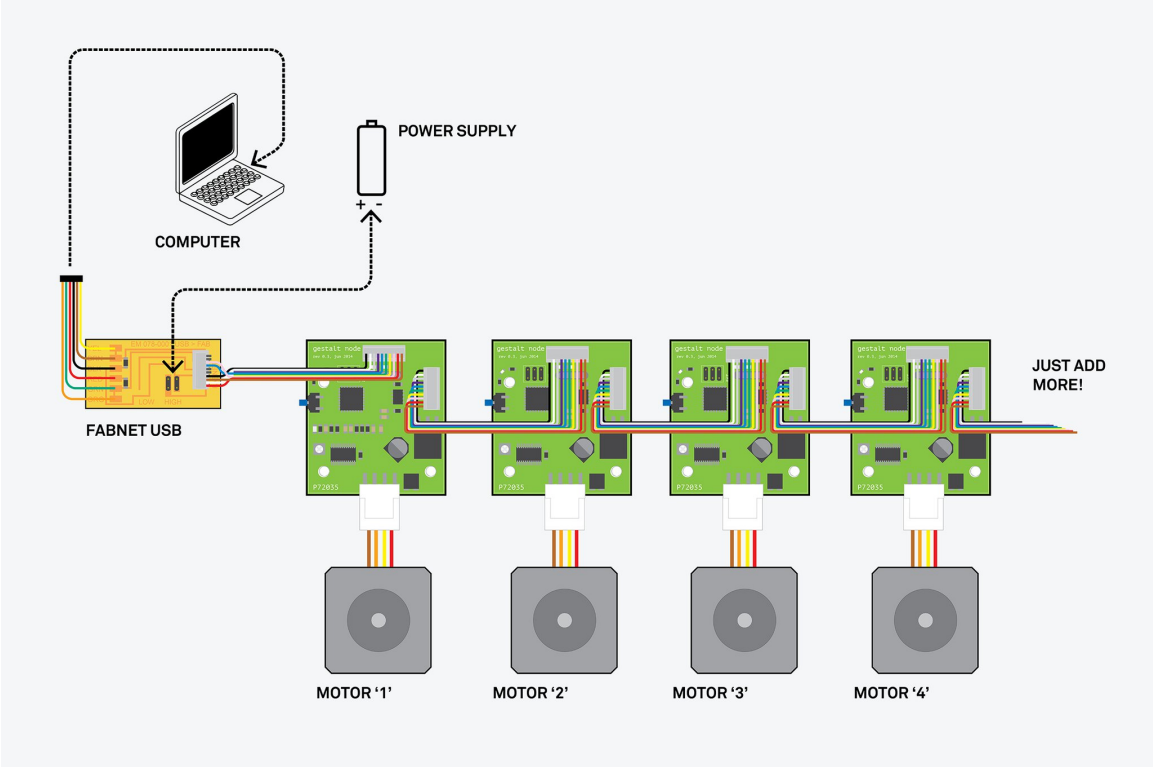


Figure 7-6: Schematic wiring for the PyGestalt nodes.

For the assembly instructions and wiring, we included animations and videos in the documentation. Some frames from an animated GIF explaining the linear stage con-

struction are depicted in Figure 7-5. Animations are much more interesting than step-by-step instructions for the participants, and are also less tedious for us to make.

Instead of only using photographic documentation, we used many schematic depictions of the hardware. Figure 7-6 shows schematic representations of the wiring. Here a schematic clearly points out details that otherwise might be overlooked, such as the orientation of the connectors.

7.3 Machine Building Workshops

Figure 7-7: MAS.863 *How to make (almost) anything* machine building workshops.

We taught a series of machine building workshops in person using the cardboard machine module designs. Two of the workshops were taught as part of the grad/undergrad

digital fabrication class *How to make (almost) anything*, with 60 students per workshop working in teams of 15. During those workshops, students came to a one-hour lecture, one-hour of lab demonstration, and had eight hours of lab time to work on their machines. Another workshop was taught during the computer graphics conference SIGGRAPH to 30 participants in one four-hour slot [61]. Finally, we taught a series of four one and a half day workshops at the MARMC naval base to five sailors at time. See also Table 7.2.

Workshop	Participants	Duration	Instruction time	Machines built
HTMAA 1	60	10hrs	3hrs	4
HTMAA 2	60	10hrs	3hrs	4
SIGGRAPH	30	3hrs	3hrs	5
MARMC	4x5	3hrs	3hrs	4x1

Table 7.2: Machine building workshops held with the cardboard construction kits.

The first part of the workshops consisted of a tour through machine design: tool heads, control systems, actuators and sensors, kinematics, mechanical systems, and applications/interfaces. We analysed examples of digital fabrication machines commonly found in Fab Labs and explained the machine design choices that were made. Why use a timing belt instead of a lead screw? What speed we can expect to mill at? What are the benefits of enclosures versus open machines?[2] How do machine control systems work? How do their software interfaces work? Where can we make improvements?

After a basic overview of machine design and machine building, we narrated the documentation we made for assembling the cardboard stages, wiring the PyGestalt nodes, and running the VMs. We distributed parts, including cardboard and guide shafts, but encouraged the participants to also look for other materials for their

[2]Hint: lasers and eyes are a bad combination.

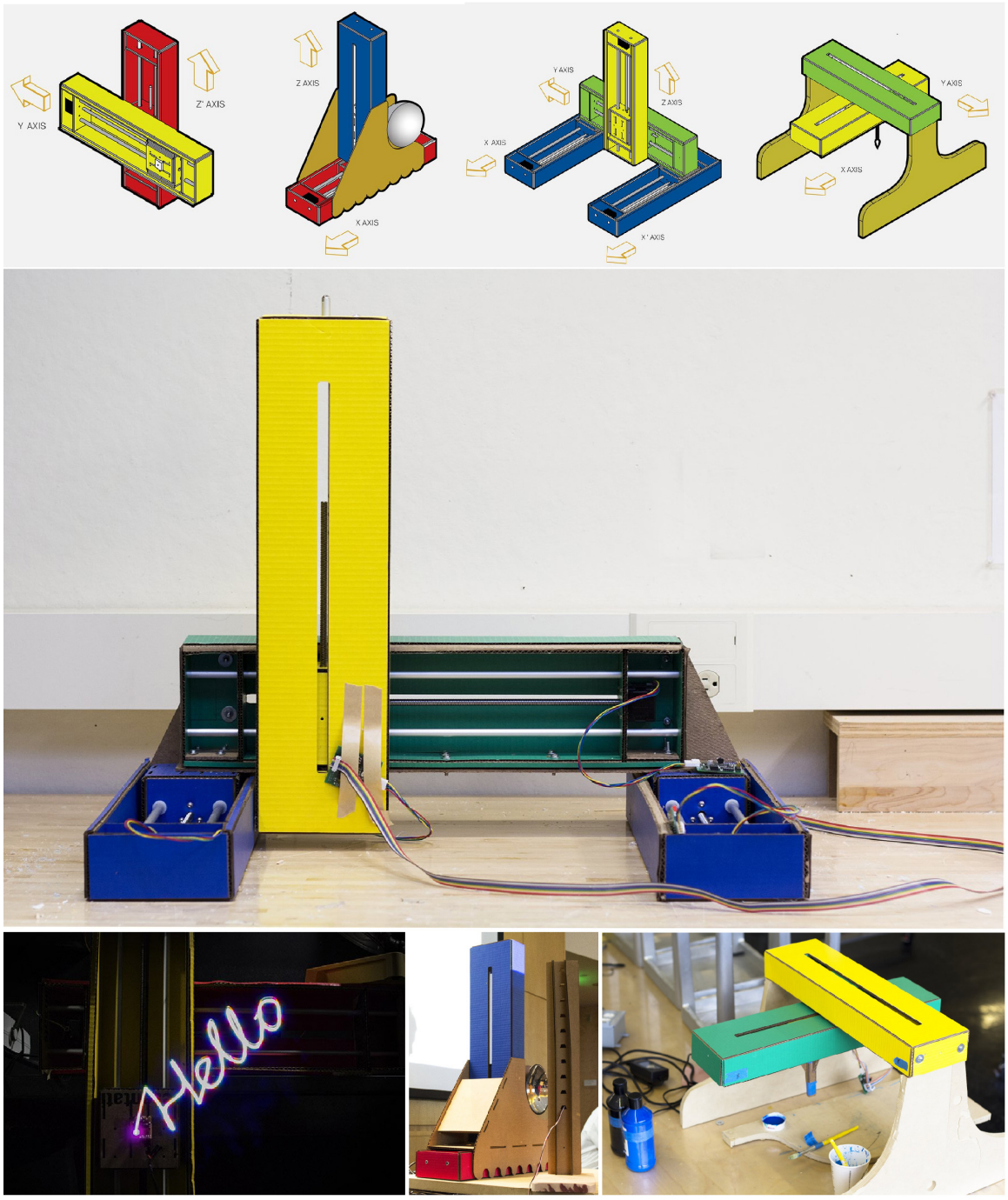


Figure 7-8: Machines built by HTMAA students: long exposure by the Harvard section; laser show by the Center for Bits and Atoms section; a mechanical turk chess-playing table by the architecture section; a painting machine by the International Design Center section.

machine building.

For the *How to make (almost) anything* workshops, the lab demonstration consisted of building four linear stages in teams. This was to ensure the toolchain was working on their own laptops without necessarily engaging in an machine design work. This included us being present through laser-cutting, gluing, and wiring. In the SIGGRAPH and MARMC workshops we also assisted during the follow-up; for HTMAA the participants had eight lab hours to design and complete a custom machine on their own. Because of the large number of participants in the HTMAA workshops, the teams were able to assign semi-experts to each of the different subcomponents and develop relatively complex machines. In the other workshops, there was more involvement from all participants throughout the machine building.

The machines for the first *How to make (almost) anything* workshop are shown in Figure 7-8. Because the first (minimally viable) machines were finished during the lab section, the remaining eight hours could be used to develop the machine-specific application. The architecture section built a chess table and developed a chess-playing app that generated the moves. They did a series of careful motion studies to prevent magnets from interfering with each other.

Due to a shortage of time and no access to a laser cutter during the SIGGRAPH workshop [61], we pre-cut almost all the cardboard parts for the machines ahead of time. The participants assembled the machines and used end effectors we already had. The participants in this workshop were most interested in developing different software workflows for toolpathing, and spent some time after the workshop writing scripts to send different kinds of commands to the machines.

The MARMC workshop participants had little to no prior experience in program-

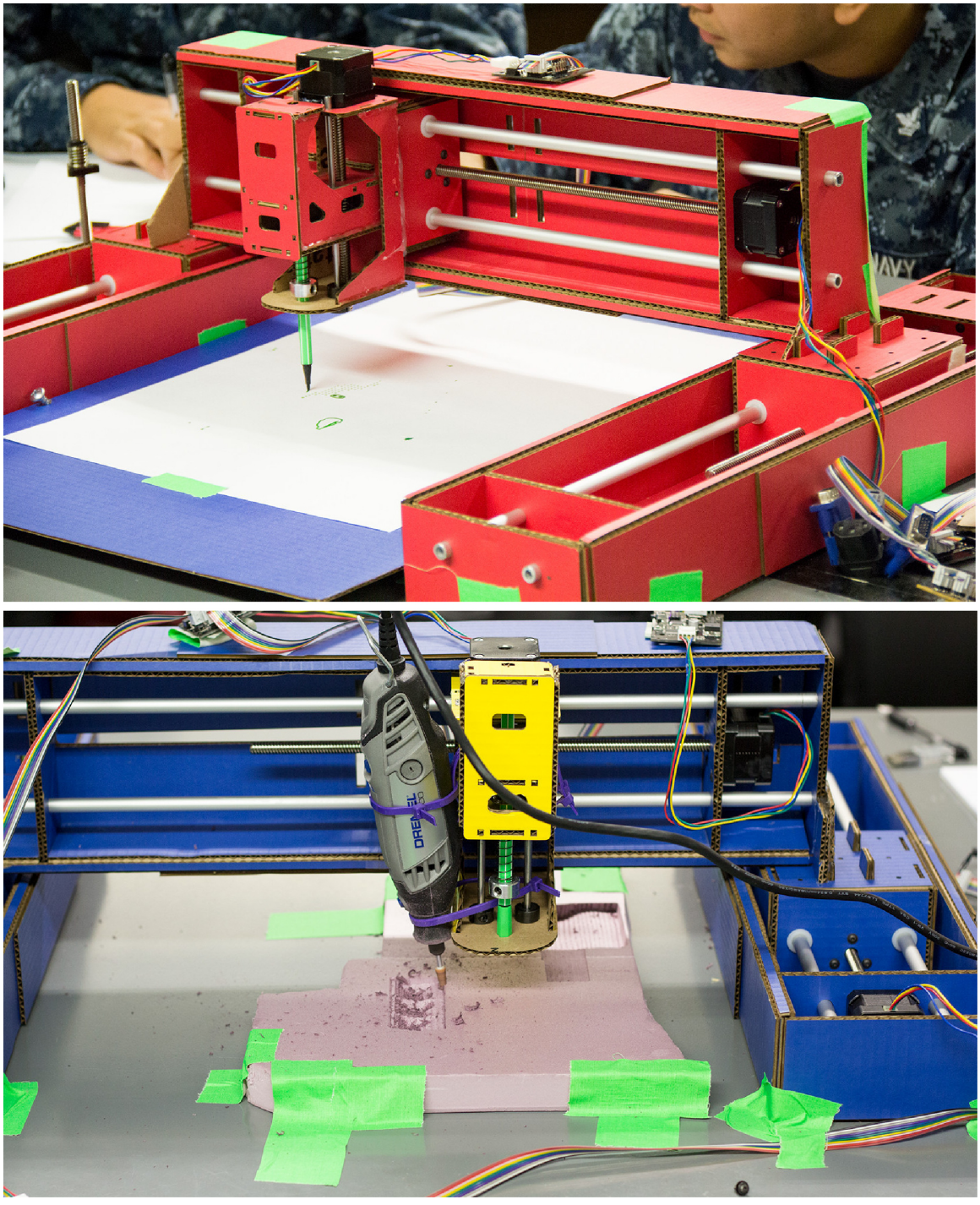


Figure 7-9: Above: a cardboard pen plotter built with 4 linear cardboard axes at MARMC shipyard. Below: A rapid iteration turning a plotting machine into a milling machine at MARMC shipyard.

ming, digital design, or digital fabrication. The workshop was taught in the newly installed fab lab that they had put on the base to provide enrichment and education for the sailors stationed there. During the workshop, the sailors all initially built the same machine—a plotter that used four linear axes (see Figure 7-9). After building the machine, they were challenged to improve it. Most groups provided structural improvement to the machines, but one group decided turning the plotter into a milling machine would be a marked improvement (see Figure 7-9).

At each of these workshops, the participants were able to successfully make a machine. Over the course of a few hours they designed machines, built mechanical components, built wiring, wrote software, and performed system integration. Because the network of machine objects was relatively simple, the participants could focus on developing their machine-specific applications. This led to a full mechanical turk robot in one workshop and to modifying a plotter to be a milling machine in another. I believe that without the focus on end-to-end principles for machine design, it would have been much more difficult for machine building novices to focus so much on the specialisation of their machines.

7.4 Modular Machines in the Wild

Besides the workshops we were available for in person, there were other sites testing the machine building infrastructure we developed. Most notably, these kits were used in the 2015 and 2016 Fab Academy cycles, with 220 students participating at 54 sites and 264 students participating at 75 sites respectively. Not all students or labs participated in the machine building exercise, but those who did built a total of 125 machines.

Not only Fab Academy students built machines. The designs for cardboard stages and stepper motor nodes were being downloaded and built by other people as well. Examples include a group of high school students, engineering professionals branching out into making, and an undergraduate class at an engineering university. However, I will focus my analysis on the machines developed in Fab Academy. The class archive `http://archive.fabacademy.org` (accessed June 2016) and the students' documentation of their projects are well suited for studying.

To get Fab Academy labs access to items from the bill of materials at low cost, the Fab Foundation compiled pre-orders from labs for PyGestalt stepper nodes, guide shafts and bushings, RS-485 cables, and motors with integrated leadscrews (see also the BOM in Table 7.1). Fifty labs ordered parts through Fab Foundation in 2015; 40 labs ordered parts in 2016. The kits were reused year to year, so the decrease in orders does not reflect a decrease in availability to participants. Of the remaining labs, some sourced the same parts themselves (including ordering the PyGestalt boards and stuffing them themselves, as in Fablab HRW), and some built machines with different hardware components altogether.

Table 7.3 catalogues the wide variety in machines built by workshop participants. Some familiar machine types were developed including milling machines, laser cutters, 3D scanners, lathes, and 3D printers. However, there were also many application-specific machines, including seed planting machines, cocktail mixing machines, petridish agitating machines, or machines made for gameplay.

The cardboard for machine building needed to be sourced and cut locally. This was straightforward for most labs and cardboard was broadly used. Limitations were encountered with small laser cutters and with widely varying cardboard thicknesses.

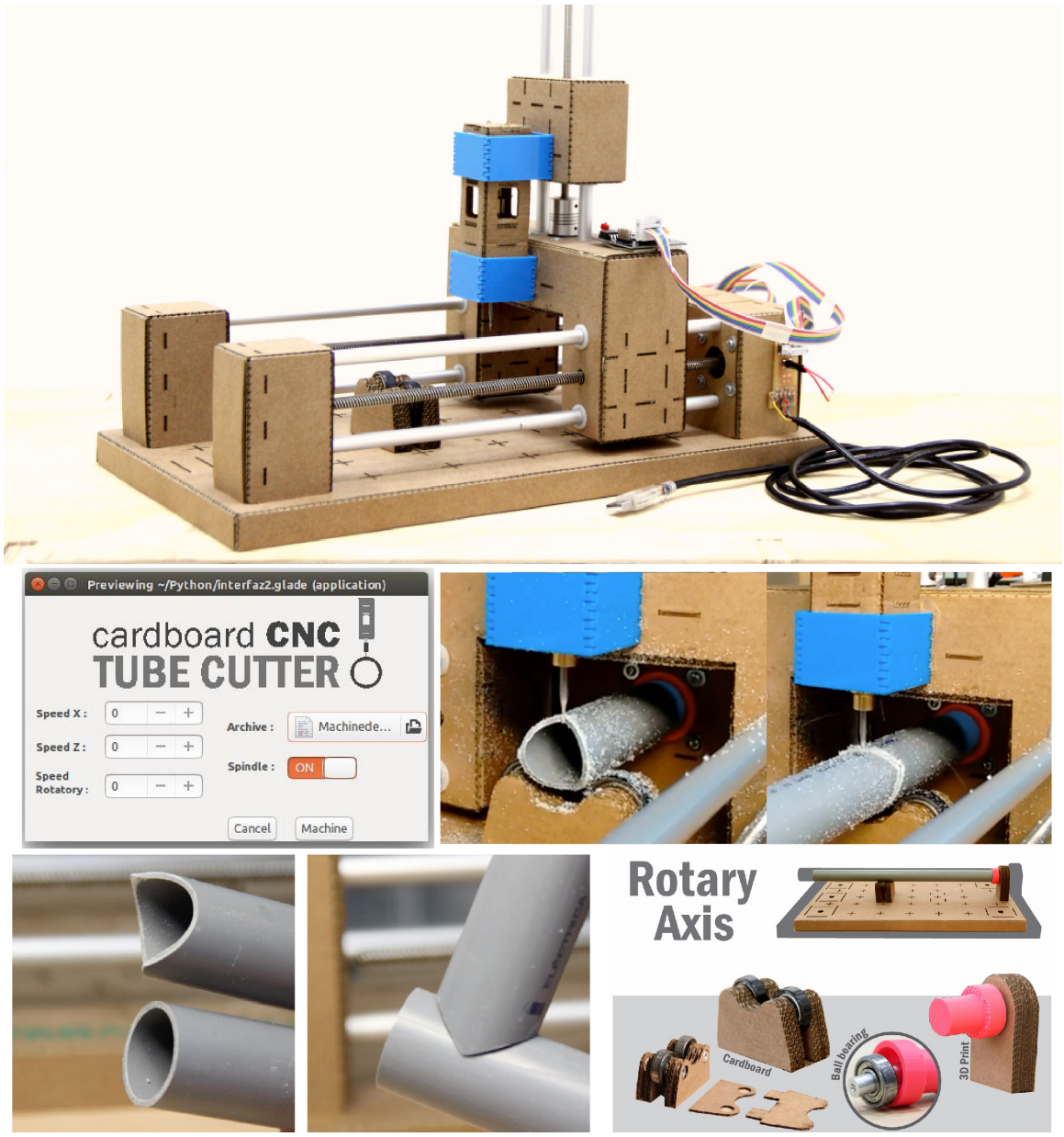


Figure 7-10: A cardboard CNC tube cutter designed by Fab Lab Tecsup students Fabio Ibarra, Gabriela Mojoli, Jesús Valencia, Roosvelth Cántaro, and Jorge Valcárcel. The machine uses a rotary axis to spin a tube, and a mill with two degrees of freedom to make precise cuts. Documentation available at `http://archive.fabacademy.org/archives/2016/fablabtecsup/students/machinedesign/index.html`, accessed June 2016.

Type of machine	**Number of machines observed**
Plotters/drawing/painting machines	25
Hot wire cutting machines	8
3D Scanners or Animation Machines	4
Laser cutters	4
Mills or lathes	12
3D printers	2
Biology lab equipment	7
Robotic arms and 5+ DoF machines	5
Music making machines	6
Arcade game machines	5
Food preparation	18
Agricultural or solar machines	5
Sand gardening machines	2
Interactive displays	12
Other machines	10
Total	125

Table 7.3: Machine types that were made as part of the Fab Academy machine building projects.

We believe it was partially due to these constraints that only 48 of the 125 machines were made predominantly out of cardboard. But using cardboard also has aesthetic considerations—more students than we anticipated objected to using such a 'maker' material, and wanted instead to go for something that seemed more like 'real engineering'. Some groups opted for a combination of 3D printed parts and acrylic sheet. This aesthetic visually indicates more engineering legitimacy, but I suspect that those machines would be comparable in stiffness to a laminated cardboard construction. A similar design decision was made by Makerbot Industries when they released the *Replicator 2* in 2012. They opted for aluminium body painted black instead of the earlier machine's plywood constructions. These examples further show how the tension between 'making' and 'engineering' plays out in many different contexts and situations.

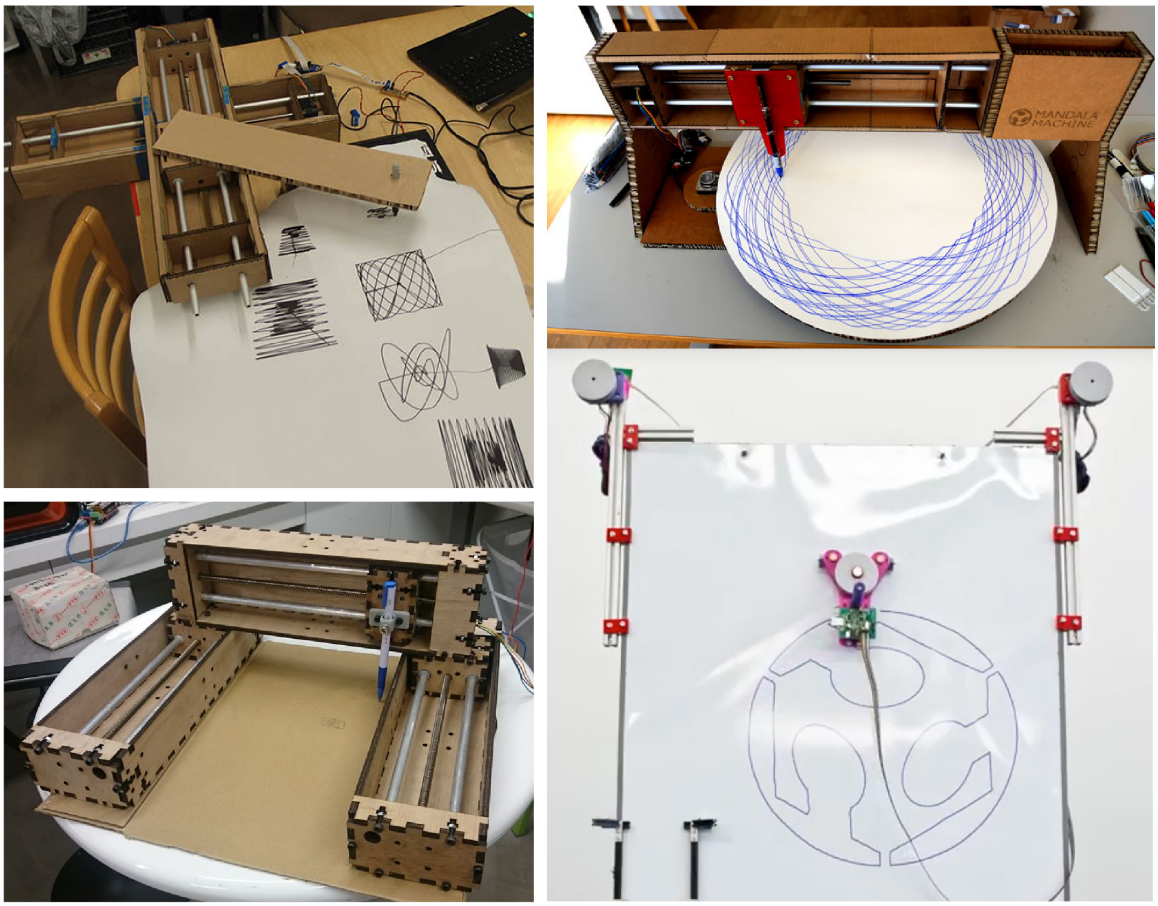


Figure 7-11: Plotters made in the wild using the machine building construction kit. The plotters on the left are by Fablab Montreal (top) and Fablab Tainan (bottom) and follow a fairly straightforward method for designing machine kinematics. The polar plotter 'Mandala Machine' and the tendon based parallel manipulator on the right by Fablab Reykjavik and Fablab HRW, respectively, use more complex kinematics. All four use the PyGestalt stepper boards for distributed controls (with Fablab HRW having sourced and stuffed the boards themselves). Twenty-five plotters or painting machines were made in total, making it the most popular machine category.

Figure 7-12: Modular hot wire cutting machines. Clockwise from top left, a four-DoF hot wire cutter (including rotating platform) by Fablab Puebla, a hot wire cutter by Fablab Waag Amsterdam, a three-DoF machine by Fablab Singapore Polytechnic, a hot wire cutter by Fablab Barcelona, results from a hot wire cutter at the Tecsup Fablab in Peru, and finally, a hot wire cutter by the Gregory School Fablab in Tuscon, AZ.

A cardboard milling machine for PVC tubing was developed by Fab Lab Tecsup (shown in Figure 7-10). It is an excellent example of a custom digital fabrication machine: it shows a functionality that is difficult to accomplish by hand (cutting complex geometries into tubes) without making an overly complex or expensive machine. It has a custom software interface (also shown in Figure 7-10) and controls a total of three degrees of freedom. As a tool, it provides functionality at an appropriate scale.

Many sites developed drawing machines or painting machines, such as the examples shown in Figure 7-11. A pen is a passive end effector; it requires no additional controls. Similarly, hot wire cutters can also be developed without additional end effector control and were a popular application. Figure 7-12 shows several examples. To make it easier to design a new type of control board in the future, we plan to release template designs for the nodes. we have already provided one design for a 3D printer extruder node, but unfortunately it uses a part that has since been discontinued by the manufacturer.

Unlike developing new control boards, using more of the existing stepper motor PyGestalt control boards appeared to be relatively simple for the participants. There was not a vast preference for working with machines with fewer degrees of freedom. In fact, four degrees of freedom was quite common, and several machines used five or more. The cost jump that exists in industrial machines in going from three to five degrees of freedom did not apply here. Participants added fourth and fifth degrees of freedom without much concern.

A popular general area of application of custom machines was food handling. Three separate labs made pancake making machines. Frosting extrusion was another celebrated application. Most of the frosting extrusion machines used syringes, but in one

of the original *How to make (almost) anything* workshops a group developed a "subtractive cake decoration machine," where the end effector was a silicone model of one of the participants tongues. Many of the food processing machines were developed by Fab Labs in Japan—two from Kitakagaya, and two from Hamamatsu.

The majority of the machines produced were somewhat difficult to classify. A few examples include a target practice machine, a delta-bot foot massaging machine, a bleach application machine, and a photo paper handling machine. Two separate labs developed Zen sand gardening machines (see Figure 7-14); one machine injected bubble wrap bubbles with paint (also in Figure 7-14). Many of these machines may seem somewhat frivolous. I would argue that if it becomes so easy to make a machine that you can apply the precision of computer control to frivolous applications, we are succeeding in our goal of making rapid prototyping of rapid prototyping machines a reality.

The informally organised community of machine builders made several valuable contributions back to the project. There was a great deal of additional documentation written by the participants, including step-by-step software installation guides, details for running the software on Windows, details for how to source parts in China, and videos with more detail on how to connect different parts mechanically. Additional documentation was also written for the PyGestalt source code, explaining the architecture of the library in one document.

Other contributions included circuit board design and software development. Bas Withagen developed a Fabnet RS-485 connector board that is easier to make cabling for (see Figure 7-16). Massimo Menichinelli contributed wxGestalt, a GUI for automatically generating PyGestalt virtual machines. In wxGestalt, the user selects the

Figure 7-13: Many machines dealt with some aspect of food preparation. For example, clockwise from top left: a ketchup printer from Fablab Kitakagaya; a pancake printer from CITC Fablab Alaska; a pizza cutter from Fablab Barcelona; a vegetable peeler from Fablab Sendai; a cookie froster from AS220; a chocolate printer from Fablab Trivandrum; a vegetable cutter from Fablab Hamamatsu.

Figure 7-14: Miscellaneous machines include machines for music playing and sand gardening. From top to bottom: the Pluribot bubble wrap cell injecting machine, by Fablab Torino; and detail of the syringe; Zen gardening machine by Fablab Opendot; another machine-controlled sand garden, this one by Lorain County Community College Fablab; a guitar-playing machine by Fablab Oulu; a windchime playing machine also by LCCC Fablab.

Figure 7-15: Cyclops: a machine that uses augmented reality markers for positioning. Cyclops was developed by Alejandro Escario-Mendez at Fablab Madrid. Additional documentation is available at `http://fabacademy.org/archives/2015/eu/students/escario_mendez.alejandro/cyclops.html` (accessed July 2016).

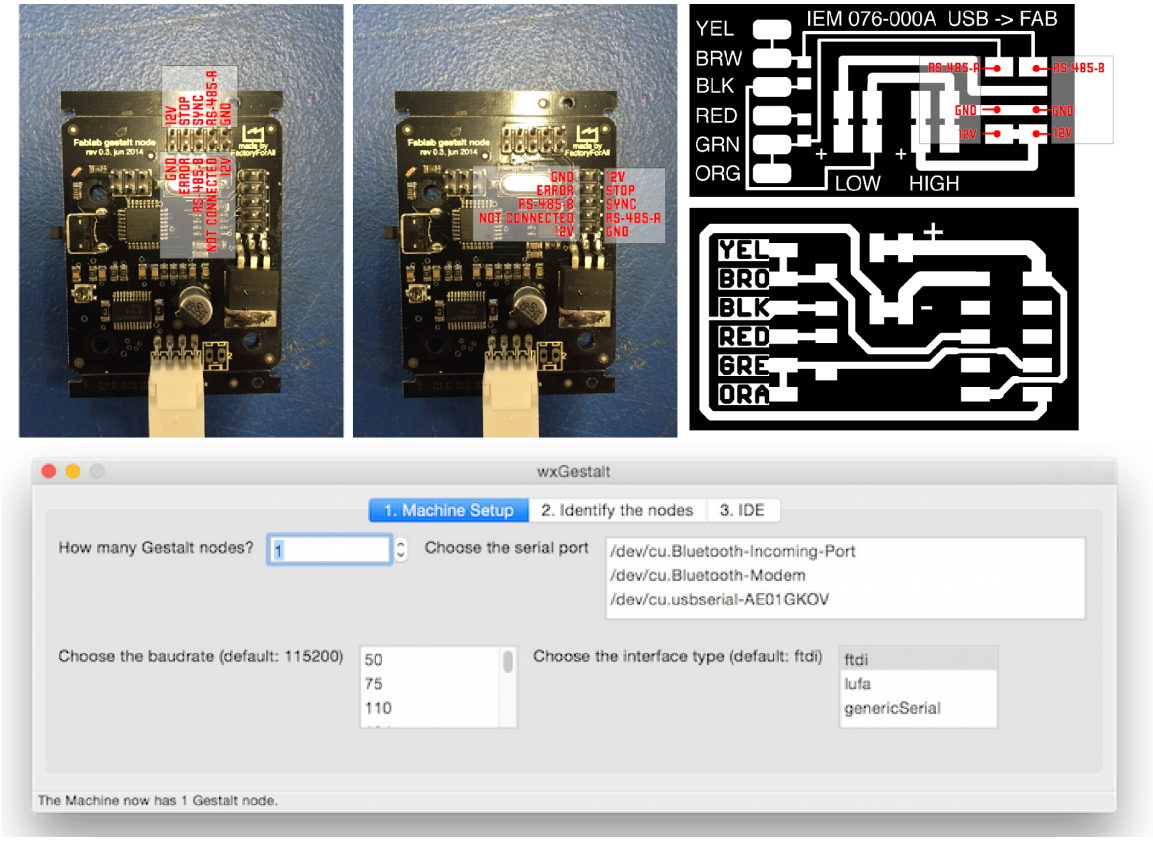


Figure 7-16: User contributions to the machine building infrastructures. On the top, additional documentation developed for PyGestalt boards, including an alternate design for the RS-485 fabnet interface board. Below, wxGestalt, a GUI for automatically generating VMs using the PyGestalt library.

number of nodes, the serial port, and details on the communication settings. The program guides the user through the process of assigning the nodes IDs (which involves pressing their physical buttons during a discovery mode). Once the VM is generated, wxGestalt also provides a template for creating a graphical interface to that machine (built in wx). These contributions build up an ecosystem of support for building machines.

The workshops that we taught have since been replicated by others. The Fab Foundation—with the TIES foundation—has adapted a version of the workshop we taught at MARMC for continued use. The workshops are being taught by instructors who have participated in the Fab Academy machine building sessions.

There are machine builders using these object-oriented hardware components who are contributing to machine building research. For example, Alejandro Escario-Mendez developed a machine called Cyclops, which uses augmented reality markers to determine machine position (See Figure 7-15). This allows for closed-loop control in positioning. Closed-loop control is typically considered an expensive addition, but Cyclops suggests that one could retrofit machines with closed loop control using a few stickers and a webcam. This kind of layered but interconnected development makes machine building more accessible and introduces new possibilities.

Overall, I was very impressed by the breadth, creativity, and complexity of the machines developed. I was struck by the ease with which people added degrees of freedom, position control, and other machine attributes that historically have been considered complicated. I was impressed by the contributions made by machine builders to the respective infrastructures they were using. I also learned more about what kinds of contributions are difficult for novices to make—designing new control boards, for

example. Having previously taught machine building workshops with kits such as the *Ultimaker* or *MTM Snap*, I was impressed by the success rate. There were only two labs who ordered kit parts and did not make a functioning machine with them.

Cardboard machines are not going to replace everything in manufacturing and fabrication. But the object-oriented hardware parts certainly made it easier to build more machines more quickly. None of the machines that was built was optimal. But this proliferation of machines shows that this approach is a step towards much more accessible rapid prototyping of rapid prototyping machines.

Chapter 8

Conclusions

At the start of this research trajectory, I thought I was designing machines (such as the *MTM Snap* or *Popfab*). I realise now that this is the wrong order. Instead of designing machines, we need to be able to design processes. Processes employ machines that are application specific. To make these machines, we need overarching infrastructure for machine design. To make this infrastructure and its standards relevant, it needs to be unprecedently accessible.

I presented both *object-oriented hardware* and *end-to-end machine design* as conceptual frameworks and design principles for process development. Object-oriented hardware considers a machine an assemblage of objects that make up a machine instantiation. Developing classes of these objects are contributions to machine building infrastructure. These objects include networked controllers, or mechanical motion components, or software modules. These objects are readily reuseable, reconfigurable, and extensile. End-to-end machine design principles argue for developing the application-specific components of the machines at the edges of the machine network,

instead of implementing application-specific functionality in the machine's subcomponents. This allows machine infrastructure (such as networked controls, software modules, or mechanical motion components) to be more general and usable in new, unanticipated applications. It makes the machine infrastructure (its objects) more reliable as they are tested and redeveloped. It reduces the complexity of the core functions (such as motion) shared by most machines.

To test these principles, I developed and deployed (hardware, software, and inbetween) objects for machine design. These were successfully used in a series of structured workshops and in the wild to develop many different kinds of application-specific machines. Novice machine builders built more than two hundred distinct machines, some of which I have catalogued in the previous chapter. The success of these tests underlines the potential of the frameworks I describe above.

The title of this thesis "*Making Machines That Make*" is routinely misheard as "*Making Machines That Make Machines*". There is an irritating futurist excitement about the notion of making self-replicating machines, and us as the masculine gods who create them. This work is distinctly not about that narrative. The design principles for machine infrastructures and the implementations I have presented here are a step towards changing how we can create products and goods, and who may create them. Instead of machines being a tool held only by those with privileged access to the means of production (and therefore what the machines make being things in service of those who already hold power) these machines are accesible and are extensions of the people who wield them.

These are tools for making machines that make *anything at all*.

Chapter 9

Reflection

> *Another flaw in the human character is that everybody wants to build and nobody wants to do maintenance.*
>
> Kurt Vonnegut, *Hocus Pocus*, 1997

There is a shop manager who works in one of MIT's fabrication facilities. Let's call her Justine[1]. Justine can program the CNC mills, fix the bandsaw, and shovels out the waterjet when it gets too full of garnet. Justine is comfortable in CAD/CAM and also writes scripts when she deems there is otherwise too much mouse clicking to be done. Justine has insight into mechanical design and material science and can provide recommendations based on experience. Justine knows where to buy end-mills, sheet metal, motors, and LEDs. Justine can rewire the robotic arms. Justine is down-to-earth and helpful. Justine makes an okay wage as 'support staff'.

People ask me all the time where they can hire someone like Justine. The problem

[1]Name has been changed to preserve privacy of the person in question–or more realistically so that fancy industry partners don't find out about our Justine and swoop in and steal her.

is that Justines don't exist. This Justine is a magical unicorn who for some reason is working a job that requires all of her skills but rewards her for only very few of them. Right now we are muddling through manfacturing the goods we need with the few Justines we have. Companies like Apple are a stronghold for Justines—they build fortresses of manufacturing expertise, with war chests of Justines protecting their competitive advantages.

If we continue as we have, I think this Justine shortage is going to become a very serious issue. Right now we throw out things that stop working. We upgrade and replace. But this practice assumes unlimited resources on a limited planet. As we increase our reliance on complex electromechanical products, we need to also expose them for reconfiguration by more than just a few experts. Our collective expertise in electromechanical products needs to grow.

One component of the work I presented here is developing skilled communities of pratice. Involving people in developing the technology they use does not only transform the technology—it transforms the people. This inclusion is not currently a common social practice; our technological infrastructures poorly support it. But having people develop the tools they use means more than those tools being better suited for them. It means the people themselves are different as well. It may be a key to having more Justines.

9.1 Interoperability Without Standardisation

The thing about standards is that none of them ever are that good for the application you have in mind. That is of course the point; standards are meant to improve

interoperability. But standards are often also overreaching, requiring too much of the products that implement them. They bog things down, making them expensive and slow.

Seeing the machine modules and controls presented here, you might think I defined the entire playing field for end-to-end machines, and am proposing a standard. This is not the case.

Going forward, I believe it is imperative to continue to develop systems that interoperate, but do not need absolute standards for communication. In the world of webservices, access points are often clearly defined by "application programming interfaces" or APIs. They denote the ways in which services can interoperate and make it easy to make mashups of different services. This is what I'd hope to see for machine design.

9.2 Object-Oriented Manufacturing

Learning from the abstractions of object-oriented programming, I developed technologies that implement some of the interconnect and interoperability you'd expect from an object-oriented system for machine design. I demonstrated the paradigm works well with these technologies as a method for rapidly constructing small scale automation systems. But what happens next?

Building machines involves a lot of legwork. To build machines you need to source material, find parts, check tolerances, and test functionality. That involves dealing with marketplaces, supply chains, and inventory. Even once you find the parts you need, you need to deal with shipping, lead times, and mark ups. Once you start

manufacturing and the system is set up, it is not necessarily clear why one machinist's parts come out better than another's. Process control and quality fade are complex manufacturing problems that have no simple solutions.

A main driving force of this work has been the ablity to manufacture in low volume without loss of precision or complexity. To do so, I argue you need access to custom digital fabrication tools. The automation of digital fabrication enables repeatability and accuracy; the customisation of those tools enables widespread applicability.

To actually produce goods in low volume, machines are not the only part that needs to be more accessible and robust. Making machines easier to make and to use could also further lower the margins on mass-manufacturing. Supply chains, quality control, and system integration also need to become accessible to low volume production. The overhead charges levied on small orders make low-volume production inaccessibly expensive.

The internet allows data packets to travel through very heterogenous networks to reach their destination. Each packet contains its addressing and details about its payload. But the same design file used on different machines will produce different parts. We lack an infrastructure as general as the internet for manufacturing.

Perhaps a next step to take is to consider something like Object-Oriented Manufacturing. What would it mean to make a formal language for supply chains? Or standard methods for quality control? What are elements that could be standardised further?

- Distributed manufacturing—many manufacturers can respond to the same order

- Product component layers with interconnectivity specifications—manufacturers can easily make parts that work together
- Security, quality control, and verifiability—enables a quick start

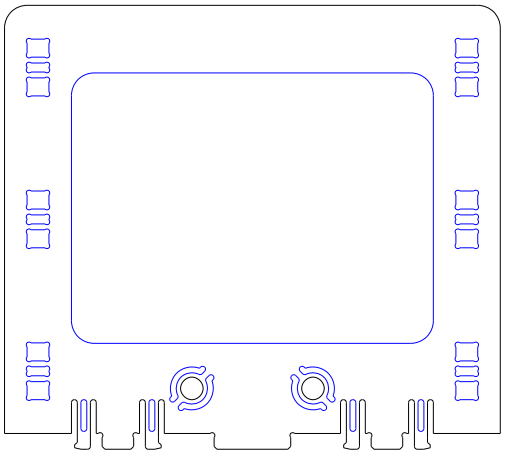
Manufacturing is unlike the Internet in that creating or destroying a packet has a cost. While we can learn from history of the digital, implementation difficulties abound in the physical. These provide a fertile foundation for future work.

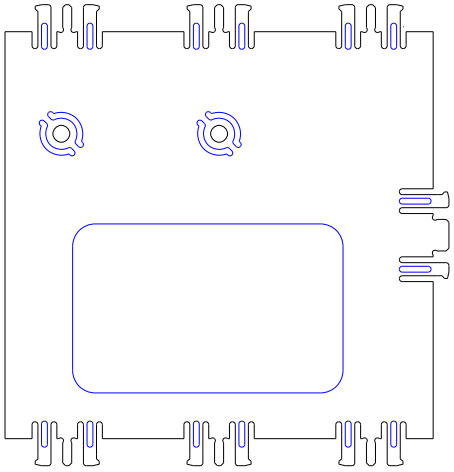
Appendix A

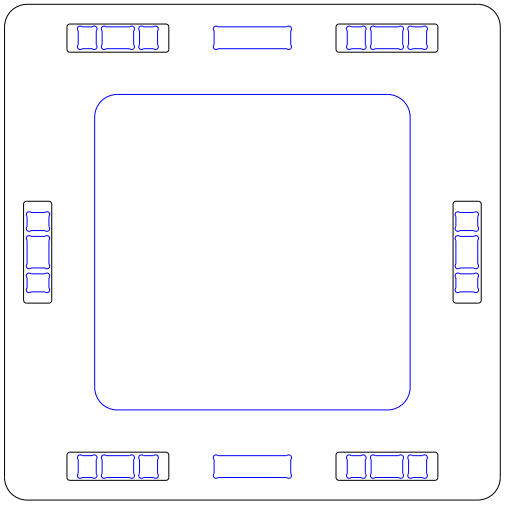
Design files

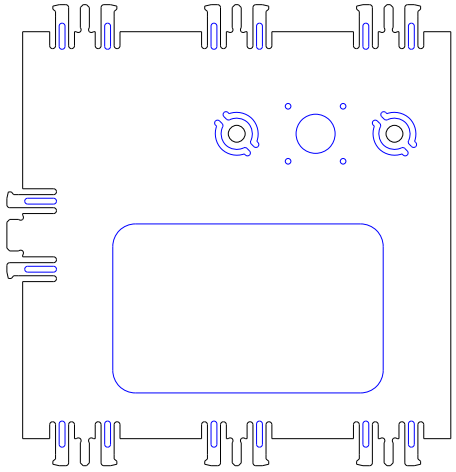
In the digital version of this dissertation, the design files below are provided as vector drawings that can be imported into CAD or CAM software. The board files are also vector graphics. These files are also available in their original formats at

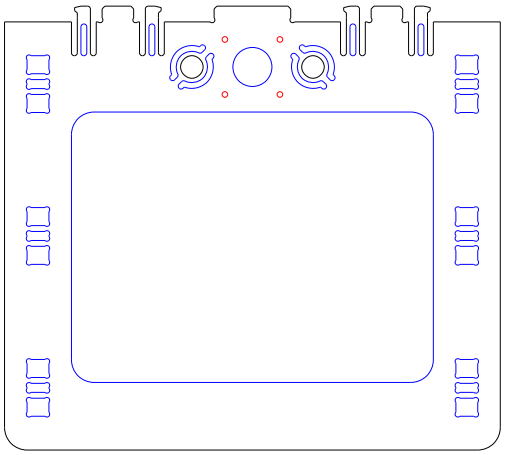
`http://mtm.cba.mit.edu`.

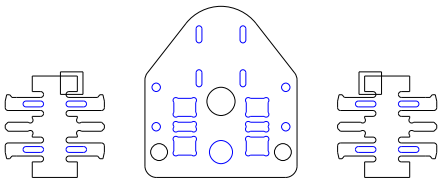


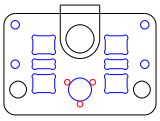


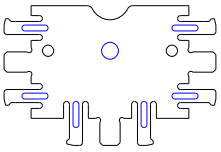


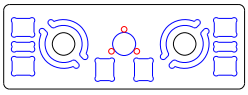


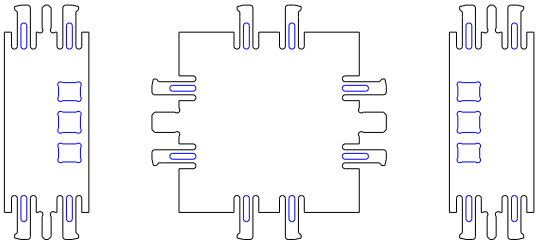


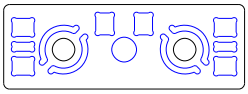


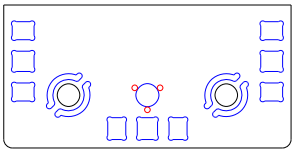


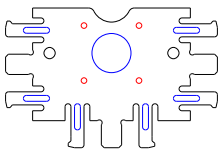


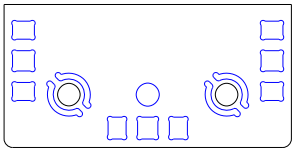


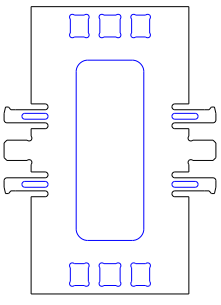


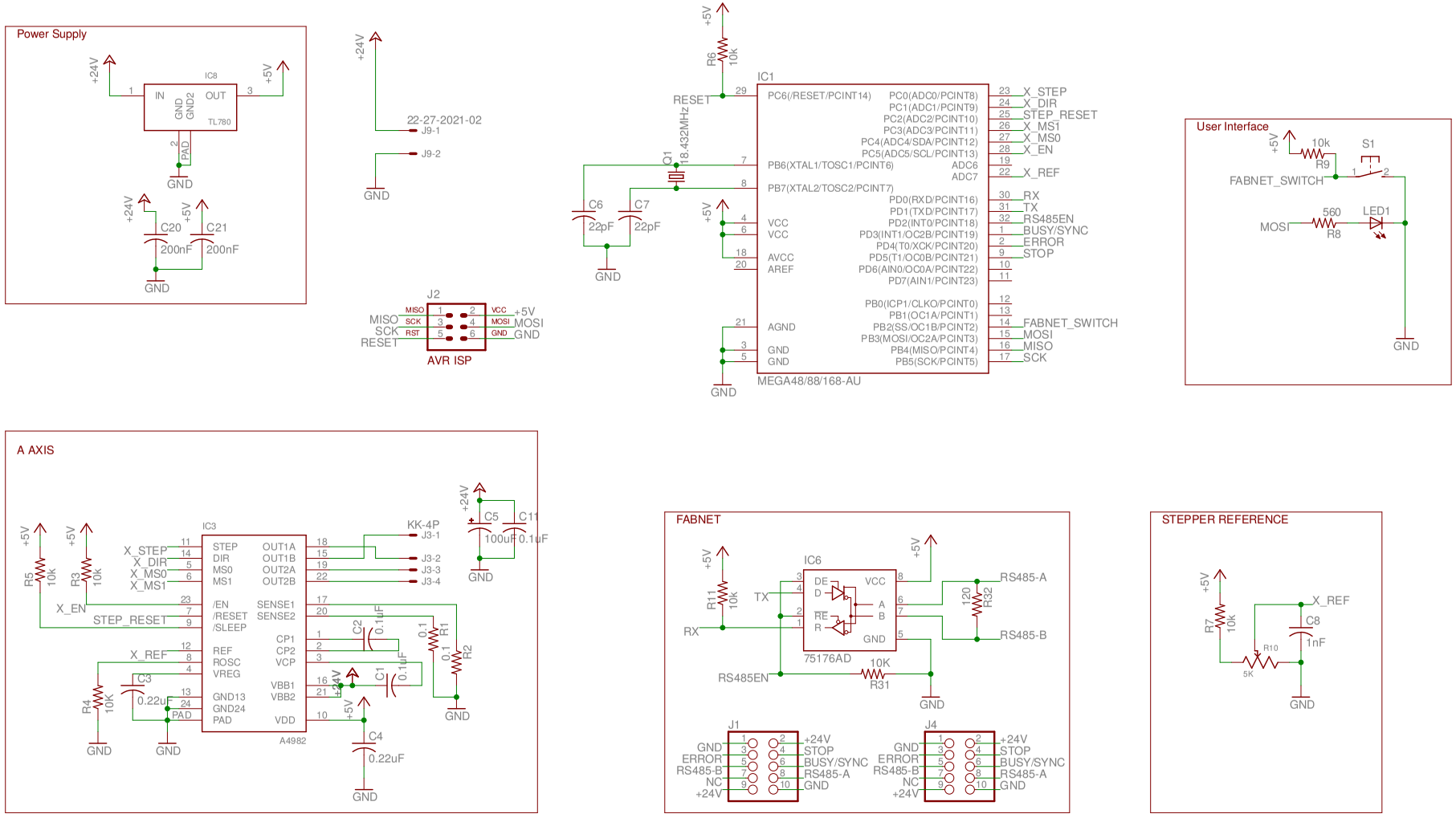


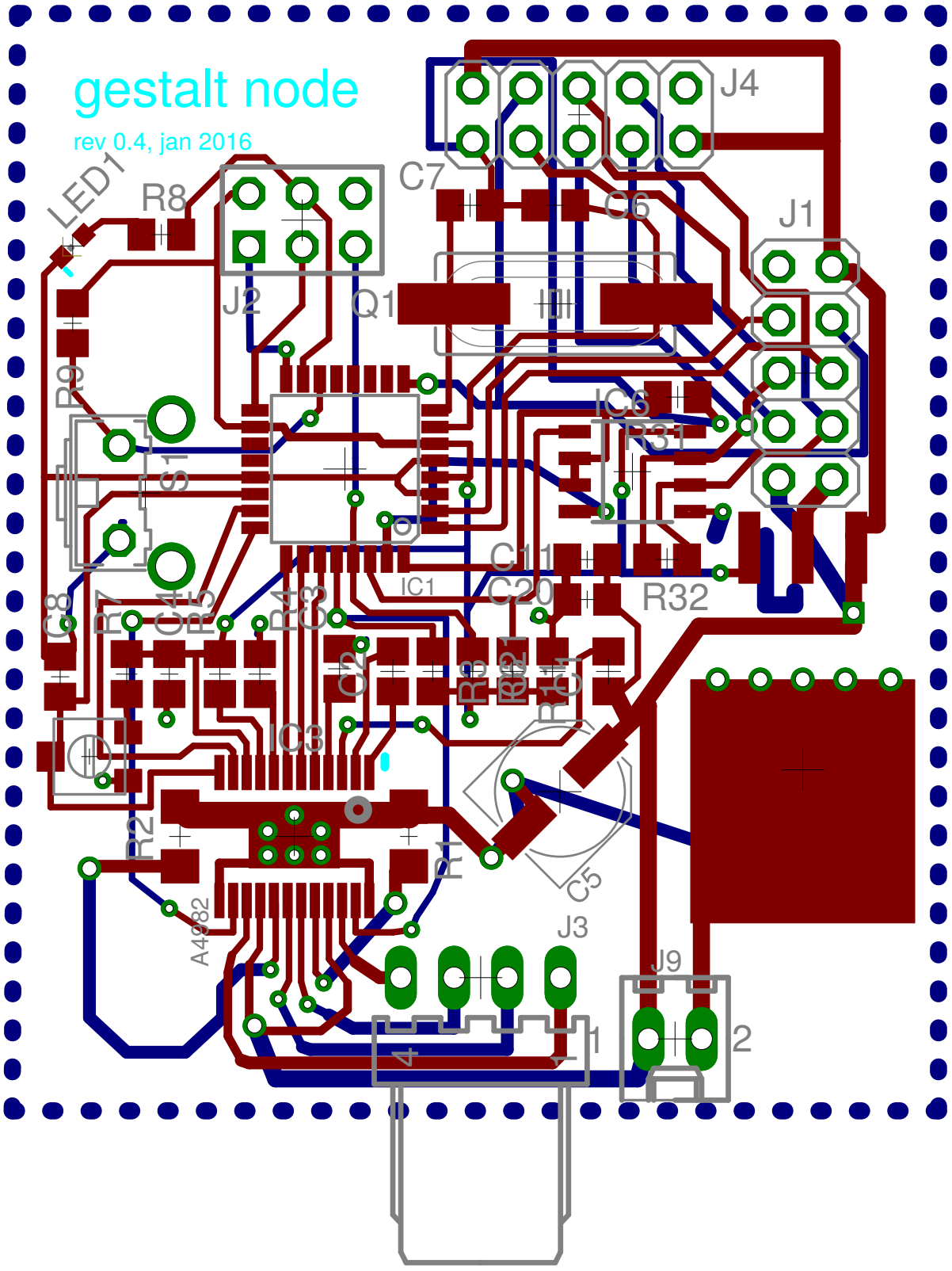


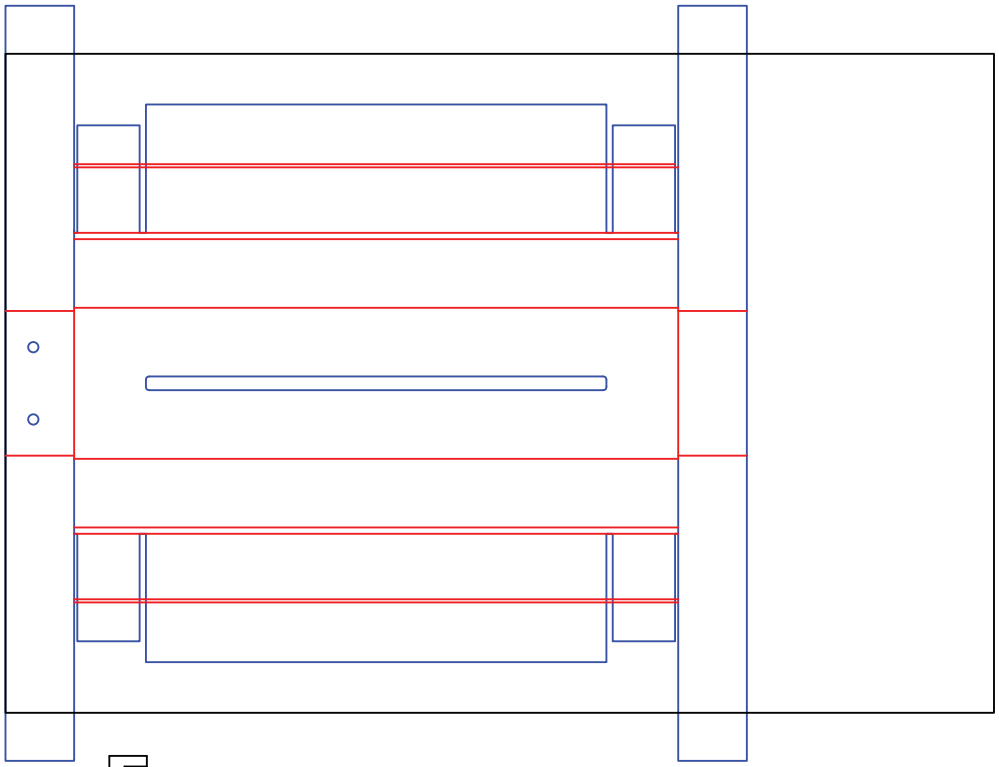




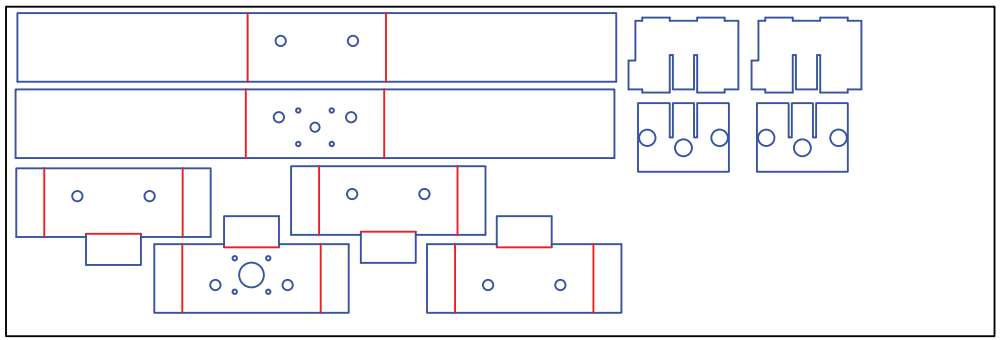



Power Supply
IC8
IN
OUT
GND
GND2
TL780
PAD
+24V
+5V
C20
C21
200nF
200nF
GND
22-27-2021-02
J9-1
J9-2
J2
MISO
SCK
RST
VCC
MOSI
GND
RESET
+5V
AVR ISP
IC1
MEGA48/88/168-AU
PC6(/RESET/PCINT14)
PB6(XTAL1/TOSC1/PCINT6)
PB7(XTAL2/TOSC2/PCINT7)
PC0(ADC0/PCINT8)
PC1(ADC1/PCINT9)
PC2(ADC2/PCINT10)
PC3(ADC3/PCINT11)
PC4(ADC4/SDA/PCINT12)
PC5(ADC5/SCL/PCINT13)
ADC6
ADC7
PD0(RXD/PCINT16)
PD1(TXD/PCINT17)
PD2(INT0/PCINT18)
PD3(INT1/OC2B/PCINT19)
PD4(T0/XCK/PCINT20)
PD5(T1/OC0B/PCINT21)
PD6(AIN0/OC0A/PCINT22)
PD7(AIN1/PCINT23)
PB0(ICP1/CLKO/PCINT0)
PB1(OC1A/PCINT1)
PB2(SS/OC1B/PCINT2)
PB3(MOSI/OC2A/PCINT3)
PB4(MISO/PCINT4)
PB5(SCK/PCINT5)
VCC
AVCC
AREF
AGND
GND
R6
10k
Q1
18.432MHz
C6
C7
22pF
22pF
X_STEP
X_DIR
STEP_RESET
X_MS1
X_MS0
X_EN
X_REF
RX
TX
RS485EN
BUSY/SYNC
ERROR
STOP
FABNET_SWITCH
MOSI
MISO
SCK
User Interface
R9
10k
S1
LED1
560
R8
A AXIS
IC3
A4982
STEP
DIR
MS0
MS1
/EN
/RESET
/SLEEP
REF
ROSC
VREG
GND13
GND24
PAD
VDD
OUT1A
OUT1B
OUT2A
OUT2B
SENSE1
SENSE2
CP1
CP2
VCP
VBB1
VBB2
KK-4P
J3-1
J3-2
J3-3
J3-4
C5
C11
100uF
0.1uF
C1
C2
C3
C4
0.1uF
0.22uF
R1
R2
R3
R4
R5
0.1
FABNET
IC6
75176AD
DE
D
RE
R
VCC
A
B
GND
R11
R31
R32
120
10K
RS485-A
RS485-B
J1
J4
STEPPER REFERENCE
R7
R10
5K
C8
1nF
X_REF

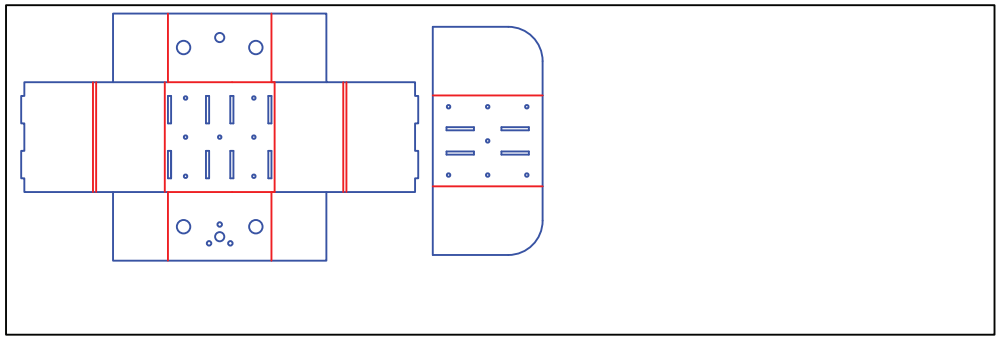

gestalt node
rev 0.4, jan 2016
LED1
R8
C7
C6
J4
J1
J2
Q1
R9
IC6
R31
C1
C11
C20
R32
IC1
C8
R1
C4
R5
C3
C2
R3
R21
R11
IC3
R2
C5
J3
J9
A4982
1
2



Face up



Facedown



Face up

Appendix B

Mods benchmarking

The *mods* framework for workflow composition that is detailed in Chapter 4 has several modules built in for benchmarking the setup that is running. These include the processing power benchmarks and connectiviity tests that are shown in Figure B-1.

Benchmarking machine performance (such as the time from sensor read through signal processing to motor action) is an important method for characterising their capabilities. It is very easy to build benchmarking suites with the tools provided here.

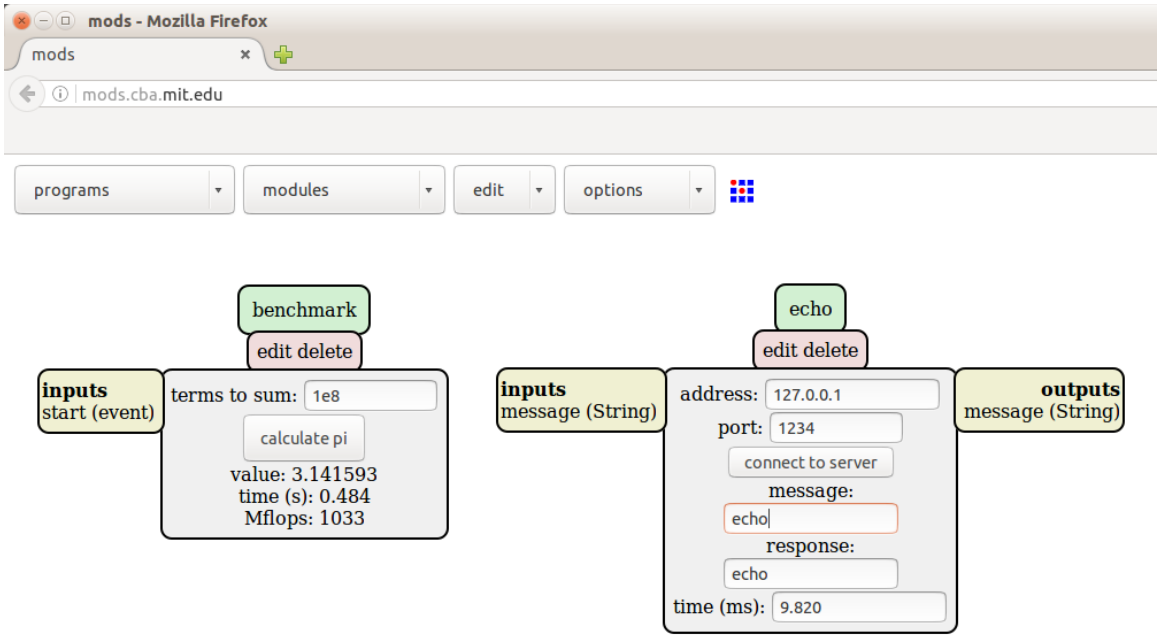


Figure B-1: *Mods* benchmarking modules, left for benchmarking processing power (on this computer 1033 Mflops), right for benchmarking connectivity (here 9.8ms round trip with the server).

Appendix C

HDPE Snap Reuse

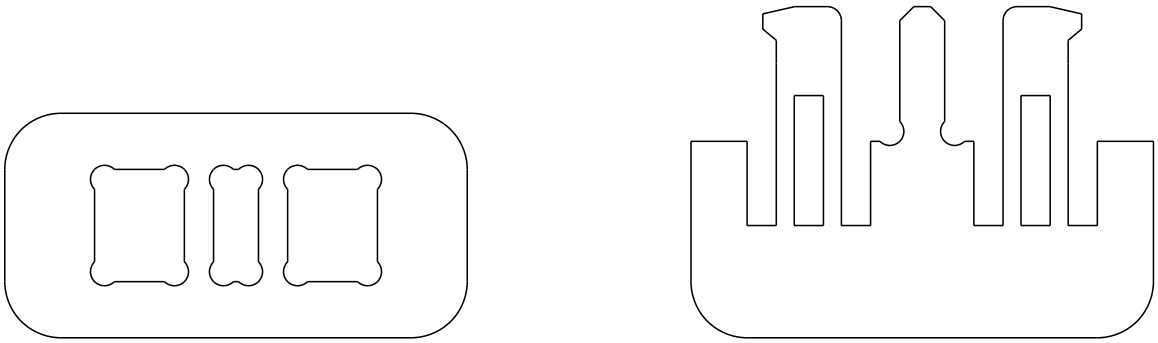


Figure C-1: A design for a plastic snap.

The MTM Snap is a milling machine for small items and circuit boards that is made out of HDPE. Different iterations and instantiations are depicted in Figure C-2. It was originally designed in 2010, with a final design iteration in 2012 which is included in Chapter A. Like I explain in Chapter 1, the project was not exactly a success. It was difficult for others to replicate the machine or modify it for their own purposes. However, parts of the MTM Snap design were readily adopted by others building rapid prototyping machines.

Figure C-2: Many iterations and instantiations of the MTM Snap, including versions not made of HDPE plastics.

The MTM snap was a generalisation of one of Jonathan Ward's earlier machines from 2009, the MTM A-Z. The A-Z was made of plywood. One of the first A-Zs was adopted by the Mobile Fab Lab for travelling around and demonstrating making machines with other digital fabrication machines. In the Mobile Fab Lab it was subjected to changing weather and especially humidity conditions, which caused the plywood to warp and the machine to bind. It used fasteners that required some finesse in the assembly process.

Making the machine out of a material which would not warp with humidity became a priority. We found out that HDPE was used in the restaurant industry as cutting boards, and was therefore readily available in large quantities. Cutting boards designated for different foods were colour-coded, giving us a palette to work with as well. To cut the plastic, we used a serrated single flute up-spiral end mill, which we ran very slowly—30 inches/minute at 14k rpm. By cutting buckles into the HDPE, we

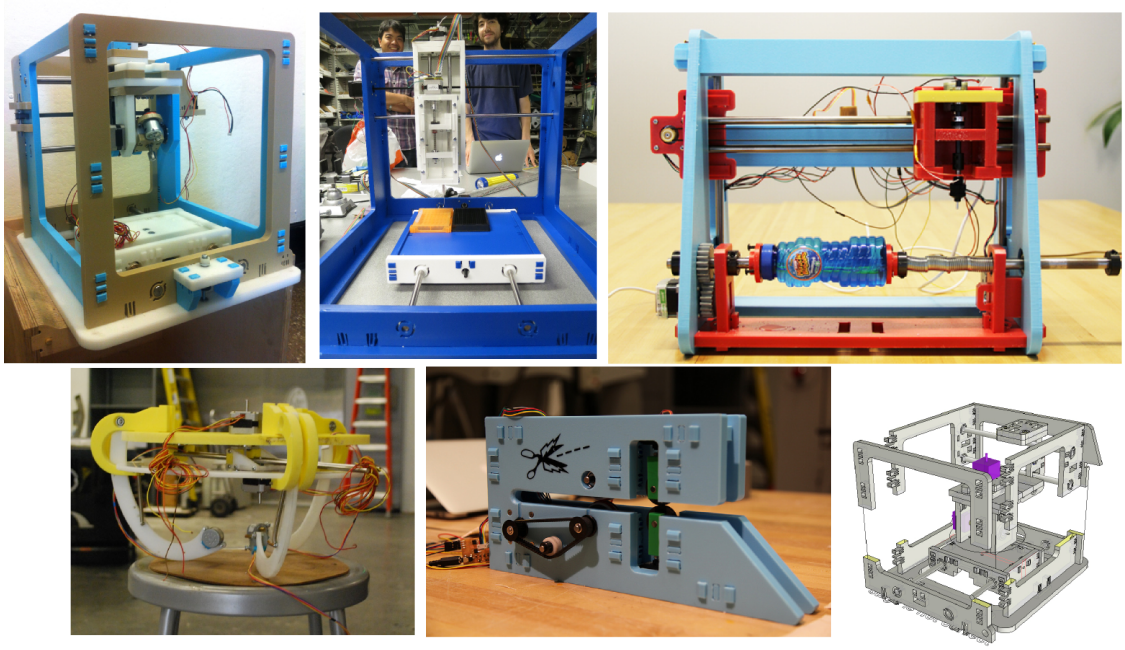


Figure C-3: Examples of other digital fabrication machines built with snapped HDPE. Clockwise from top right: the 5-axis MTM Snap by James Coleman; the Juicebot Liquid Handler by the author, Charles Fracchia, Emzo de los Santos, and Scott Livingston; the Additive Lathe by Yoav Shterman; detail of the 5-axis MTM Snap; A Creepy-Crawly Fabric Cutter by Sam Calisch; and a hanging 3D printer by Ben Peters.

were able to assemble structures without needing fasteners or special tools.

While the MTM Snap as a machine might not have been widely replicated, the snap structure in HDPE cutting boards was reused in many machines. Figure C-3 shows some examples of how the snaps were used as construction techniques.

The Open Source Hardware Definition [3] is very clear about how to share stand-alone works such as the MTM Snap. But in this case it turned out to be only a certain process (milling plastics and fitting them together with snaps) that was useful to a broader community. It is my hope that object-oriented hardware will make it easier to share these kinds of small but useful machine building components.

Appendix D

Sourcing and Supply Chain

Figure D-1: A marketplace for electronic components in Huaqiang Bei, Shenzhen.

Unlike for software, replicating designs of machines requires sourcing components and materials for production. To be able to make the cardboard modules shown in

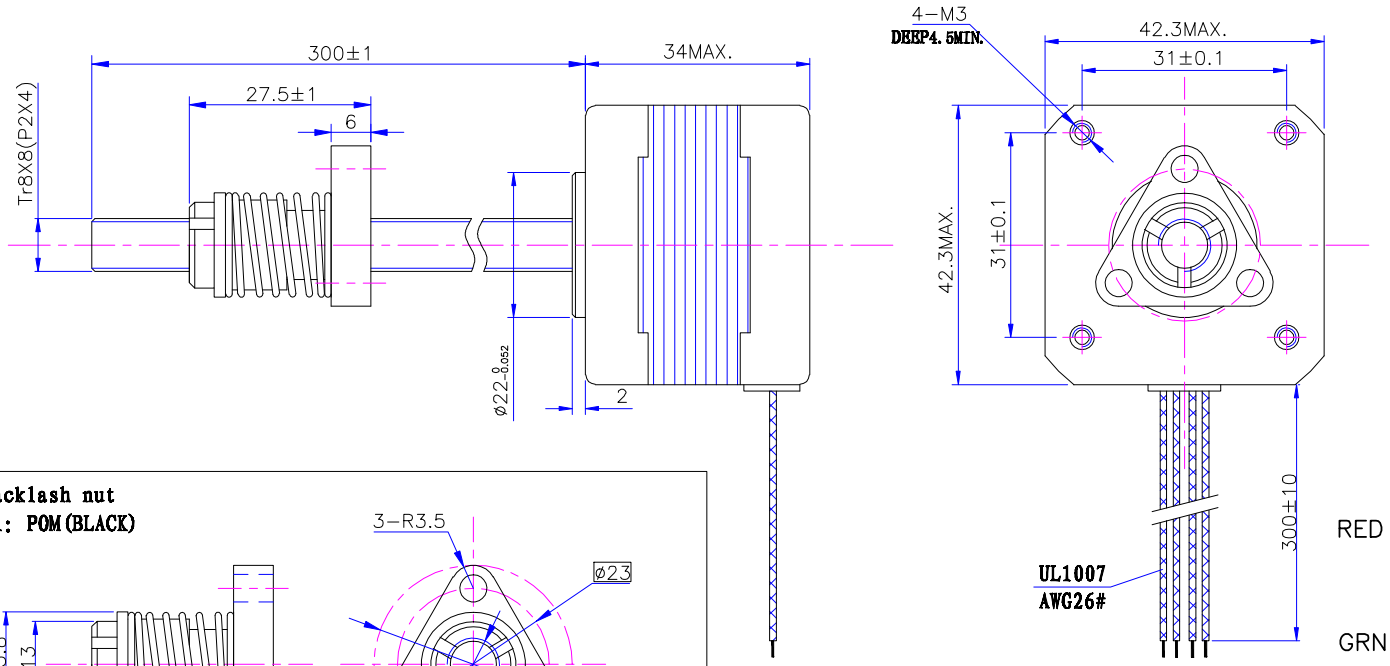
Chapter 7, we needed guide shafts, bushings, fasteners, cardboard, and motors.

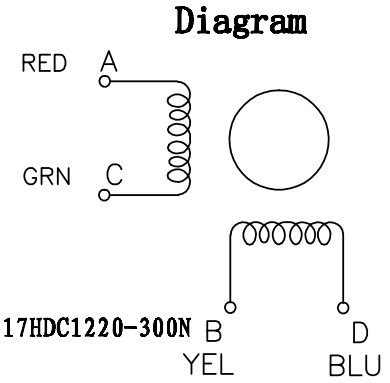
We would usually source our parts from off-the-shelf vendors such as McMaster-Carr. For this project we decided to try to find a less expensive option to be able to accommodate more machine builders. At first we tried to find them in marketplaces in Shenzhen such as shown in Figure D-1. However, we wanted to be able to source more reliably.

To do so, we partnered with *AQS* inc. in Dongguan, China for sourcing and production. We ordered bipolar NEMA 17 1.8 degree stepper motors with a 4-start lead screw and 2mm pitch (TR8x8) and an ABS wear-compensating lead screw nut. The datasheet is shown on the next page.

The first order we placed for motors was for less than 20 units. While it used to be that having parts manufactured in China required high volumes, that is no longer entirely true. Especially with the assistance of a manufacturing partner, it become feasible to have parts custom made at much lower volumes.

Making machines that make helps us gain access to the precision of CNC for low volume production. To make those machines though, it is good that we also have access to low volume production of machine parts. Thanks to AQS and Bunnie Huang for their assistance.





备注：原电机型号17HDC1517-300E已更改为新型号17HDC1220-300N

Items	Specs	Items	Specs
Winding Number	2 Phases	Step Angle / resolution ratio	1.8° ± 5%/0.04mm
Rated Voltage	DC 6.0V	Rated current	DC 1.0A / Phase
Resistance(20° C)	6.0×(1±10%)ohm	Inductance	12×(1±20%) mH
Max.axial pull force	≥8Kgf(25mm/s)	Detent Torque	0.5Kgf REF
Directlon of rotation	A-AB-B- UP	Starting line velocity	≥ 6 mm/s
MAX.line velocity	≥ 16 mm/s	Insulation Resistance	≥ 100 MΩ (DC 500V)
Dielectric strength	AC600V/1mA/1S	Insulation class	B
Rotor Inertia	37 g.cm²	Weight	0.50 Kg REF.

REV	ECN NO	Description	Made by	Check	Appr	Date

		View		17HDC1220-300N		
		Scale	1：1			
			A0			
Des	Check	Appr	Date	Factory Code	RB1.001.R0139.2301-01	page

Bibliography

[1] Bok Y. Ahn, Eric B. Duoss, Michael J. Motala, Xiaoying Guo, Sang-Il Park, Yujie Xiong, Jongseung Yoon, Ralph G. Nuzzo, John A. Rogers, and Jennifer A. Lewis. Omnidirectional printing of flexible, stretchable, and spanning silver microelectrodes. *Science*, 323(5921):1590–1593, 2009.

[2] C. Anderson. *Makers: The New Industrial Revolution*. Crown Publishing Group, 2012.

[3] Open Source Hardware Association. Open source hardware definition 1.0. `http://www.oshwa.org/definition/`, accessed May 2016.

[4] Jonathan Bean and Daniela Rosner. Making: Movement or Brand? *interactions*, 21(1):26–27, January 2014.

[5] M.A. Bell, C.M.E. Graves, and K. Dumont. 3d printer for printing a plurality of material types, July 7 2016. US Patent App. 14/986,373.

[6] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the internet: The end-to-end arguments vs. the brave new world. *ACM Trans. Internet Technol.*, 1(1):70–109, August 2001.

[7] Richard J. Boland Jr, Kalle Lyytinen, and Youngjin Yoo. Wakes of innovation in project networks: The case of digital 3-d representations in architecture, engineering, and construction. *Organization science*, 18(4):631–647, 2007.

[8] T. Bonswetch, Daniel Kobel, Fabio Gramazio, and Matthias Kohler. The informed wall: applying additive digital fabrication techniques on architecture. *Proceedings of the 25th Annual Conference of the Association for Computer Aided Design in Architecture*, 2006.

[9] Kenneth K. Boyer, G.Keong Leong, Peter T. Ward, and Lee J. Krajewski. Unlocking the potential of advanced manufacturing technologies. *Journal of Operations Management*, 15(4):331 – 347, 1997.

[10] Johannes Braumann and Sigrid Brell-Cokcan. Real-time robot simulation and control for architectural design. In *30th eCAADe Conference, Prague*, 2012.

[11] Sigrid Brell-Cokcan and Johannes Braumann. A new parametric design tool for robot milling. In *Proceedings of the 30th Annual Conference of the Association for Computer Aided Design in Architecture*, pages 357–363, 2010.

[12] Kenneth C. Cheung, Erik D. Demaine, Jonathan R. Bachrach, and Saul Griffith. Programmable assembly with universally foldable strings (moteins). *IEEE Transactions on Robotics*, 27(4):718–729, 2011.

[13] Kenneth C. Cheung and Neil Gershenfeld. Reversibly assembled cellular composite materials. *Science*, 341(6151):1219–1221, 2013.

[14] David Clark. The design philosophy of the darpa internet protocols. *ACM SIGCOMM Computer Communication Review*, 18(4):106–114, 1988.

[15] James Coleman, Craig Long, Andrew Manto, and Trygve Wastvedt. Lots of parts, lots of formats, lots of headache. *XRDS*, 22(3):54–57, April 2016.

[16] James S Cybulski, James Clements, and Manu Prakash. Foldscope: origami-based paper microscope. *PloS one*, 9(6):e98781, 2014.

[17] Erik de Bruijn. On the viability of the Open Source Development model for the design of physical objects: Lessons learned from the RepRap project. Master's thesis, Tilburg University, 2010.

[18] Shawn M. Douglas, Ido Bachelet, and George M. Church. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335(6070):831–834, 2012.

[19] A. Dunne. *Hertzian Tales: Electronic Products, Aesthetic Experience, and Critical Design*. MIT Press, 1999.

[20] Willemijn S Elkhuizen, Tim Zaman, Wim Verhofstad, Pieter P Jonker, Joris Dik, and Jo MP Geraedts. Topographical scanning and reproduction of near-planar surfaces of paintings. In *IS&T/SPIE Electronic Imaging*, pages 901809–901809. International Society for Optics and Photonics, 2014.

[21] Henry Etzkowitz, Andrew Webster, Christiane Gebhardt, and Branca Regina Cantisano Terra. The future of the university and the university of the future: evolution of ivory tower to entrepreneurial paradigm. *Research policy*, 29(2):313–330, 2000.

[22] K. R. Fall and W. R. Stevens. *TCP/IP Illustrated*. Addison-Wesley professional computing series. Addison-Wesley, 2011.

[23] S. Felton, M. Tolley, E. Demaine, D. Rus, and R. Wood. A method for building self-folding machines. *Science*, 345(6197):644–646, 2014.

[24] 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 *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 277–282, May 2013.

[25] N. Gershenfeld, N. Peek, K. Cheung, and D. Watson. Methods and apparatus for online calorimetry, June 25 2013. U.S. Patent 8,473,093.

[26] Neil Gershenfeld. How to Make Almost Anything: The Digital Fabrication Revolution. *Foreign Affairs*, 91:43–57, 2012.

[27] W. Gibson. *Distrust That Particular Flavor*. Penguin Publishing Group, 2012.

[28] Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2485–2492. IEEE, 2010.

[29] Jamison Go. High-throughput extrusion-based additive manufacturing. Master's thesis, MIT, Cambridge, 2015.

[30] A. Goldberg. *SmallTalk-80: The Interactive Programming Environment*. Addison-Wesley Series in Computer Science. Addison-Wesley, 1984.

[31] F. Gramazio, Gramazio & Kohler, and M. Kohler. *Digital Materiality in Architecture*. Springer, 2008.

[32] F. Gramazio and M. Kohler. *Made by Robots: Challenging Architecture at a Larger Scale*. Architectural Design. Wiley, 2014.

[33] Gregory M. Gratson, Mingjie Xu, and Jennifer A. Lewis. Microperiodic structures: direct writing of three-dimensional webs. *Nature*, 428(6981):386–386, 2004.

[34] Gregory J. Hamlin and Arthur C. Sanderson. Tetrobot modular robotics: Prototype and experiments. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems' 96, (IROS 96),*, volume 2, pages 390–395. IEEE, 1996.

[35] Mark Hatch. *The maker movement manifesto: rules for innovation in the new world of crafters, hackers, and tinkerers*. McGraw Hill Professional, 2013.

[36] Elliot Hawkes, B. An, N. M. Benbernou, H. Tanaka, S. Kim, E. D. Demaine, D. Rus, and R. J. Wood. Programmable matter by folding. *Proceedings of the National Academy of Sciences*, 107(28):12441–12445, 2010.

[37] Neville Hogan. Controlling impedance at the man/machine interface. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 1626–1631. IEEE, 1989.

[38] R. Johnstone and J.E. Kurtzhaltz. Flexible manufacturing system, September 18 1984. U.S. Patent 4,472,783.

[39] Rhys Jones, Patrick Haufe, Edward Sells, Pejman Iravani, Vik Olliver, Chris Palmer, and Adrian Bowyer. RepRap—the replicating rapid prototyper. *Robotica*, 29(01):177–191, 2011.

[40] Matthew Keeter. Hierarchical volumetric object representations for digital fabrication workflows. In *ACM SIGGRAPH 2013 Posters*, SIGGRAPH '13, pages 84:1–84:1, New York, NY, USA, 2013. ACM.

[41] Y. Koren. *Computer control of manufacturing systems*. Mechanical engineering series. McGraw-Hill, 1983.

[42] Will Langford. Electronic Digital Materials. Master's thesis, MIT, Cambridge, 2014.

[43] Barton Lee. Where Gutenberg meets guns: The liberator, 3D-printed weapons, and the first amendment. *NCL Rev.*, 92:1393, 2013.

[44] J.A. Lewis. Direct Ink Writing of 3D Functional Materials. *Advanced Functional Materials*, 16(17):2193–2204, 2006.

[45] Joseph CR Licklider. Man-computer symbiosis. *IRE transactions on human factors in electronics*, (1):4–11, 1960.

[46] Silvia Lindtner, Shaowen Bardzell, and Jeffrey Bardzell. Reconstituting the Utopian Vision of Making: HCI After Technosolutionism. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 1390–1402, New York, NY, USA, 2016. ACM.

[47] H. Lipson. Homemade. *IEEE Spectrum*, 42(5):24–31, 2005.

[48] Jesse Louis-Rosenberg. Drowning in triangle soup: The quest for a better 3-d printing file format. *XRDS*, 22(3):58–62, April 2016.

[49] Yan Lu, Sangsu Choi, and Paul Witherell. Towards an Integrated Data Schema Design for Additive Manufacturing: Conceptual Modeling. In *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 1A, pages 32–43. American Society of Mechanical Engineers, 2015.

[50] N. P. Mahalik. *Fieldbus Technology: Industrial Network Standards for Real-Time Distributed Control*. Engineering Online Library. Springer, 2003.

[51] Evan Malone and Hod Lipson. Fab@home: the personal desktop fabricator kit. *Rapid Prototyping Journal*, 13(4):245–255, 2007.

[52] J. Martin. *Design of Man-Computer Dialogues*. Prentice-Hall International Series in Industrial and Systems. Prentice-Hall, 1973.

[53] Jarkko Moilanen, Angela Daly, Ramon Lobato, and Darcy Allen. Cultures of sharing in 3D printing: What can we learn from the licence choices of Thingiverse users? *Journal of Peer Production: Disruption and the Law*, 6, 2015.

[54] Javier Monedero. Parametric design: a review and some experiences. *Automation in Construction*, 9(4):369–377, 2000.

[55] Ilan Moyer. Core XY. `http://corexy.com/`, 2012. accessed 2016-01-09.

[56] Ilan Moyer. A Gestalt Framework for Virtual Machine Control of Automated Tools. Master's thesis, MIT, Cambridge, 2013.

[57] Ilan Moyer. Personal fabrication: From automated machines to augmented tools. *XRDS*, 22(3):28–31, April 2016.

[58] Caitlin T. Mueller and John A. Ochsendorf. Combining structural performance and designer preferences in evolutionary design space exploration. *Automation in Construction*, 52:70–82, 2015.

[59] D. F. Noble. *Forces of Production: A Social History of Industrial Automation*. Oxford University Press, 1986.

[60] Cagdas D. Onal, Robert J. Wood, and Daniela Rus. Towards printable robotics: Origami-inspired planar fabrication of three-dimensional mechanisms. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4608–4613. IEEE, 2011.

[61] Nadya Peek and James Coleman. Design machines. In *SIGGRAPH 2015: Studio*, SIGGRAPH '15, pages 2:1–2:1, New York, NY, USA, 2015. ACM.

[62] Nadya Peek and Ilan Moyer. 086-005. `http://github.com/imoyer/086-005`, 2015. accessed 2016-07-09.

[63] A. Pickering. *The Mangle of Practice: Time, Agency, and Science*. University of Chicago Press, 2010.

[64] G. Prytz. A performance analysis of ethercat and profinet irt. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 408–415, Sept 2008.

[65] M. Rader, B. Wingert, and U. Riehm. *Social Science Research on CAD/CAM: Results of a First European Workshop*. Physica-Verlag HD, 2012.

[66] Y. Reches, J. Livingston, I. Ferguson, D. Cranor, M. Lobovsky, and N. Linder. Three-dimensional printer, February 3 2015. US Patent D722,108.

[67] Aristides A. G. Requicha and Herbert B. Voelcker. Solid Modeling: A Historical Summary and Contemporary Assessment. *IEEE computer graphics and applications*, 2(2):9–24, 1982.

[68] Alec Rivers, Ilan E. Moyer, and Frédo Durand. Position-correcting Tools for 2D Digital Fabrication. *ACM Trans. Graph.*, 31(4):88:1–88:7, July 2012.

[69] Lawrence G. Roberts. The evolution of packet switching. *Proceedings of the IEEE*, 66(11):1307–1313, 1978.

[70] Analisa Russo, Bok Yeop Ahn, Jacob J. Adams, Eric B. Duoss, Jennifer T. Bernhard, and Jennifer A. Lewis. Pen-on-paper flexible electronics. *Advanced materials*, 23(30):3426–3430, 2011.

[71] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[72] A. Satariano and P. Burrows. Apple's Supply-Chain Secret? Hoard Lasers. *Bloomberg Businessweek Technology*, 4, 2011.

[73] W. S. Seames. *Computer Numerical Control: Concepts and Programming*. Delmar, 2001.

[74] A. H. Slocum. *Precision Machine Design*. Society of Manufacturing Engineers, 1992.

[75] Alexander H. Slocum. Design of three-groove kinematic couplings. *Precision Engineering*, 14(2):67 – 76, 1992.

[76] Taylor Soper. After raising $28M in record crowdfunding campaign, Glowforge delays initial shipments. `http://www.geekwire.com/2016/raising-28m-record-crowdfunding-campaign/-glowforge-delays-initial-shipments/`, 2016. accessed 2016-05-11.

[77] L. Suchman. *Human-Machine Reconfigurations: Plans and Situated Actions*. Learning in Doing: Social, Cognitive and Computational Perspectives. Cambridge University Press, 2007.

[78] Lucy A Suchman. *Plans and situated actions: The problem of human-machine communication*. Cambridge university press, 1987.

[79] Dmitry Sukhotskii. CNC Machine in one day: "123Mill". http://www.thingiverse.com/thing:167769, https://www.youtube.com/watch?v=U326jtAZlpo. Accessed: 2016-05-21.

[80] Ke Sun, Teng-Sing Wei, Bok Yeop Ahn, Jung Yoon Seo, Shen J Dillon, and Jennifer A Lewis. 3d printing of interdigitated li-ion microbattery architectures. *Advanced Materials*, 25(33):4539–4543, 2013.

[81] Ivan E Sutherland. Sketchpad: A man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, pages 6–329. ACM, 1964.

[82] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 2003.

[83] Skylar Tibbits. 4D Printing: Multi-Material Shape Change. *Architectural Design*, 84(1):116–121, 2014.

[84] Skylar Tibbits and Kenny Cheung. Programmable materials for architectural assembly and automation. *Assembly Automation*, 32(3):216–225, 2012.

[85] Turlif Vilbrandt, Evan Malone, Hod Lipson, and Alexander Pasko. Universal desktop fabrication. In Alexander Pasko, Valery Adzhiev, and Peter Comnino, editors, *Heterogeneous Objects Modelling and Applications*, pages 259–284. Springer, 2008.

[86] Shawn Wallace. Maker Faire 3D Printer Census. http://makezine.com/2012/05/19/maker-faire-3d-printer-census/. Accessed: 2016-03-30.

[87] George M. Whitesides and Bartosz Grzybowski. Self-assembly at all scales. *Science*, 295(5564):2418–2421, 2002.

[88] George M. Whitesides, John P. Mathias, and Christopher T. Seto. Molecular self-assembly and nanochemistry: a chemical strategy for the synthesis of nanostructures. Technical report, DTIC Document, 1991.

[89] Jan Willmann, Federico Augugliaro, Thomas Cadalbert, Raffaello D'Andrea, Fabio Gramazio, and Matthias Kohler. Aerial robotic construction towards a new field of architectural research. *International journal of architectural computing*, 10(3):439–459, 2012.

[90] Robert J. Wood, Srinath Avadhanula, Manas Menon, and Ronald S. Fearing. Microrobotics using composite materials: the micromechanical flying insect thorax. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1842–1849. IEEE, 2003.

[91] Willie Wu, Adam DeConinck, and Jennifer A. Lewis. Omnidirectional Printing of 3D Microvascular Networks. *Advanced Materials*, 23(24):H178–H183, 2011.

[92] XW Xu and Stephen T Newman. Making CNC machine tools more open, interoperable and intelligent–a review of the technologies. *Computers in Industry*, 57(2):141–152, 2006.

[93] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics Automation Magazine*, 14(1):43–52, March 2007.

[94] Amit Zoran and Joseph A. Paradiso. FreeD: A Freehand Digital Sculpting Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2613–2616, New York, NY, USA, 2013. ACM.

[95] Victor Zykov, Andrew Chan, and Hod Lipson. Molecubes: An open-source modular robotics kit. In *IROS-2007 Self-Reconfigurable Robotics Workshop*, pages 3–6, 2007.

[96] Victor Zykov, Efstathios Mytilinaios, Mark Desnoyer, and Hod Lipson. Evolved and designed self-reproducing modular robotics. *IEEE Transactions on robotics*, 23(2):308–319, 2007.