The End of GCode

Jake Read
MIT Center for Bits and Atoms

Table of contents

The End of GCode: Machine Architectures for Feedback Systems Assembly

Neil Gershenfeld
Director, Center for Bits and Atoms
Massachusetts Institute of Technology

Nadya Peek
Associate Professor, Human Centered Design and Engineering
University of Washington

Jon Seppala
Director, nSoft Consortium
National Institute of Standards and Technology

1 Abstract

GCode is an antiquated interface that lies between machine users (and planning algorithms) and low-level controllers. It prevents innovations to machine building that would cross the boundary between high- and low-level planning and control tasks, it makes it more difficult to invent new machines, and harder to train new machine builders and users. This thesis aims to replace GCode in two steps.

First, I develop and implement a distributed systems interconnect model that re-casts machine controllers as networked systems that are modular across hardware and software. This involves networking over heterogeneous links, developing clock synchronization routines, and writing lightweight transport and serialization layers. I develop a distributed name service (to automatically discover configurations) and functional discovery services (to automatically generate interfaces). Second, I develop a series of machine control libraries and applications within this model. This involves developing a flexible motion controller that can interface with process models, developing suitable intermediate representations of motion for distributed systems, and managing distributed data flows for loop closing at multiple levels.

To show the viability and promise of these contributions, I deploy them in three experimental systems. (1) A flexible motion controller that performs across a heterogeneity of mechanical ar-

chitectures, automatically tuning kinematic models, (2) A 3D Printer that can autonomously learn its own optimal control parameters by modeling material properties, and (3) a CNC Router that autonomously learns feeds and speeds for new tools and materials.

In these formulations, machine controllers are no longer black boxes; they are distributed algorithms made of recognizable design patterns familiar to most contemporary programmers and engineers, regardless of domain. With successful industrial adoption, this paradigm will enable the next generation's engineers and scientists to rapidly invent, modify and deploy novel machine systems as they work through the next decades' most pressing issues.

2 Goals or Problem Statement

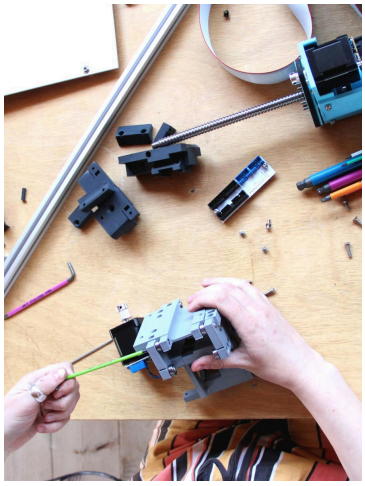
2.1 Making, Maintaining and Using Machines is Difficult

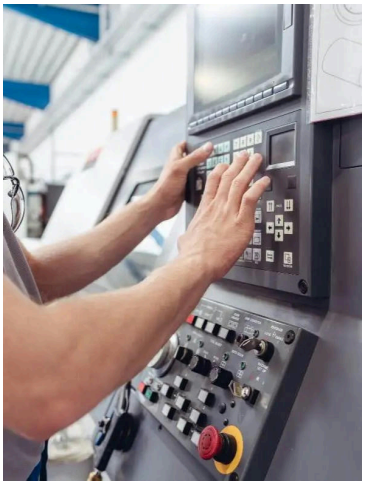
Machine building is a core competency for our industrial society, and is becoming increasingly democratized. Engineering firms have long used digital fabrication machines to prototype their outputs, and are increasingly building machines in-house for automation or process-specific tasks. Alongside these engineers are hobbyists, STEM educators, and scientists who all use machines to learn - about craft, about maths and artistry, and about our world.

The democratization of digital fabrication has led to a trend where machinery is being developed and deployed by individuals whose *main concern* is not machine building (or use) itself, as was formerly the case where the purchase of a new machine normally involved also hiring a full-time operator for that machine, and machines were developed only by specialist machine-building firms.

For example, educators and students are using machines as one of many tools in the classroom [1], [2], craftspeople are using them as *components* of larger workflows [3], and scientists are using them to automate their laboratory work [4], [5].

This motivates the central question in this thesis: **how can we make the making, maintenance and use of machines simpler?** In particular, I am interested in replacing a pervasive, niche intermediary representation of machine control (GCode) with more interpretible and extensible representations that are intuitive (based on physics) and authored in an inspectable, easily modified systems architecture.





Machine building necessarily covers a range of disciplines from electrical engineering to computer science and controls - not to mention mechanical design. Systems integration across these disciplines is the name of the game, but in the state of the art this core task is made cumbersome by the continued use of historical control architectures that were originally developed for unchanging, feed-forward industrial equipment.

This historical architecture places real-time control tasks underneath a low-level task representation (GCode), and hides configurations in firmwares[1] that are difficult to modify - either accidentally (because firmware is naturally less inspectible than i.e. software written in a scripting language) or intentionally (because it is treated as proprietary). This means that it is difficult to reuse machine components between tasks, we have to develop new circuits and firmwares in order to accomplish *new* tasks, and we cannot readily access low-level controller states even when they have large ramifications for our machines' behaviour.

The pervasive use of GCode (and the way it relates to path planning softwares), also makes it difficult for anyone to *use* machines because it requires that users develop extremely specific, often idiosyncratic settings for each new material or machine they use.

[1]We call the low-level computer codes that run on micrcontrollers 'Firmware' - the name reflects the difficulty in changing it, and its closeness to hardware. Firmwares typically run on small computing environments (which today means MHz and MBs), vs. Software which normally runs on i.e. a laptop, on top of an operating system. Firmwares are an important component in any hardware system, since they allow programmers to have tight control over their program's timing. Operating systems and high-level language interpreters (on the other hand) sometimes interrupt program execution to switch threads or run garbage collection routines.

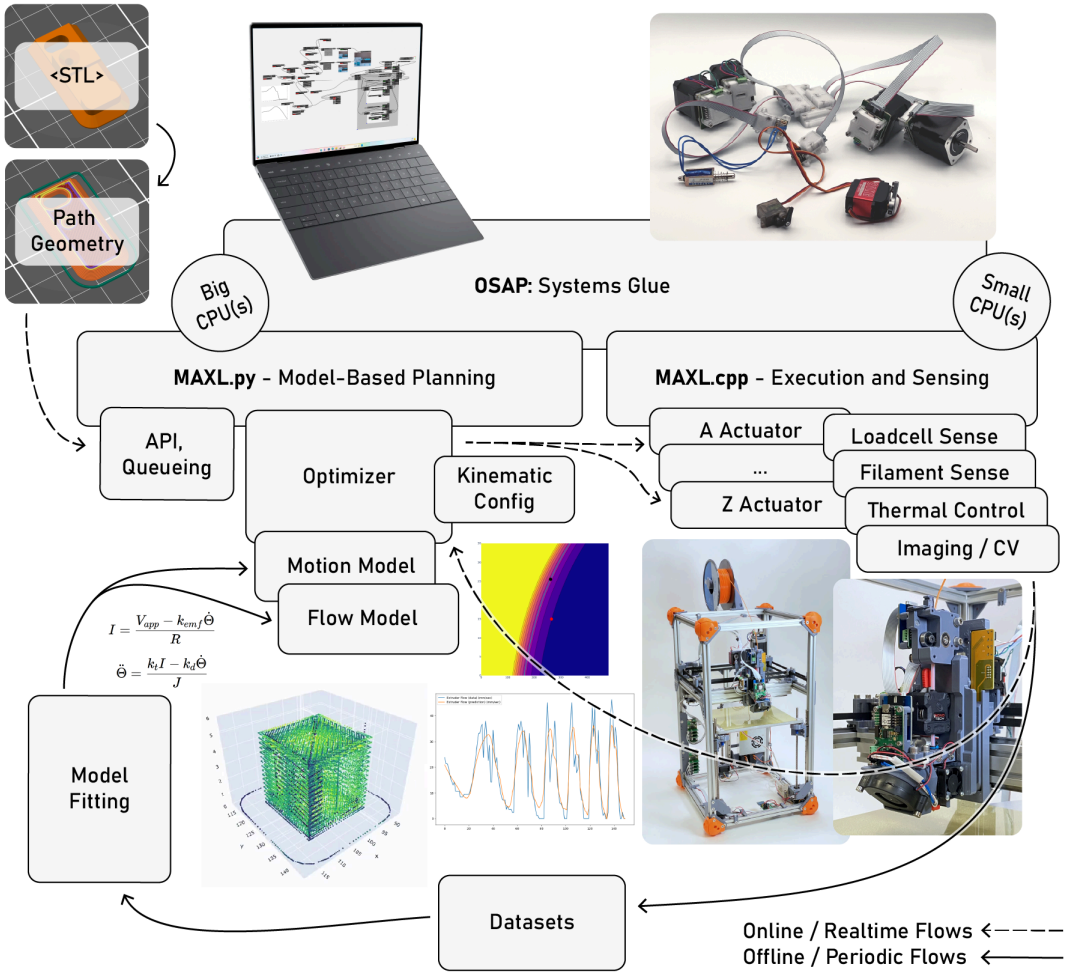


Figure 1: A systems overview of the contributions made in this thesis. I refactor a mostly feed-forward workflow (see Figure 3) and replace parameter selection (see Section 3.3) with online optimization (see Section 5.1). In this architecture, machines can continuously improve datasets and update models of their own physics, which they use in order to control their outputs.

In this thesis I continue an historical arc of CBA research based on machine *virtualization*, revisiting key components of machine control architecture in an effort to make the *making* of machines more flexible and fluid, and the *usage* of machines more intuitive. First, I develop a networked control architecture as a series of modular components that can be rapidly integrated using patterns familiar to *almost anyone who can write a computer program.* Those patterns allow me to move a core machine control task off of the small, slow computer chips that run low-level firmwares and into a high-performance modern parallel computing environment (made accessible by machine learning researchers). In doing so, I can then refactor machine control from low-level, feed-forward instruction following, to higher level model-based approaches that are more reflective of real-world physics, and that are more robust, easier to get *up and running* when machines or materials are modified, and are more performant.

As I discuss in Section 3.3.a, Section 3.3.b and Section 3.3.c, much of the work that machine users need to do to operate digital fabrication equipment is an implicit optimization. In this thesis, I try to make that optimization explicit, and base its operation on models that are generated in-situ on the machines in question.

2.2 Machines are Thoughtless

In the state of the art, **digital fabrication equipment[2] does not think about what it is doing,** nor can it tell a user what it is capable of, anticipate errors that may arise from a particular program, or be easily re-configured to do a task that its original programmers did not imagine. Instead, most machines programmed to carry out very simple instructions (GCodes, see Section 3.1) very reliably. They leave high level thinking to offline algorithms and when mismatches arise between pre-defined plans and real conditions they either fail or throw errors. For all of its advance, most digital fabrication seems about as sophisticated as the injket printing we are all familiar with: sometimes straightforward and unsurprising, or else error prone and frustrating, but never enlightening.

Figure 2: Most of the world's 3D Printers and CNC Milling machines' realtime controllers know very little about what they are doing, they simply consume and execute series' of low-level instructions called *GCodes.* Those codes are generated in software that knows little about the machine (and has no way to verify what it does know).

[2]In this thesis, I will take *Digital Fabrication Equipment* to mean CNC Machines, 3D Printers or other direct-write fabrication equipment (not including *automation* equipment more broadly).

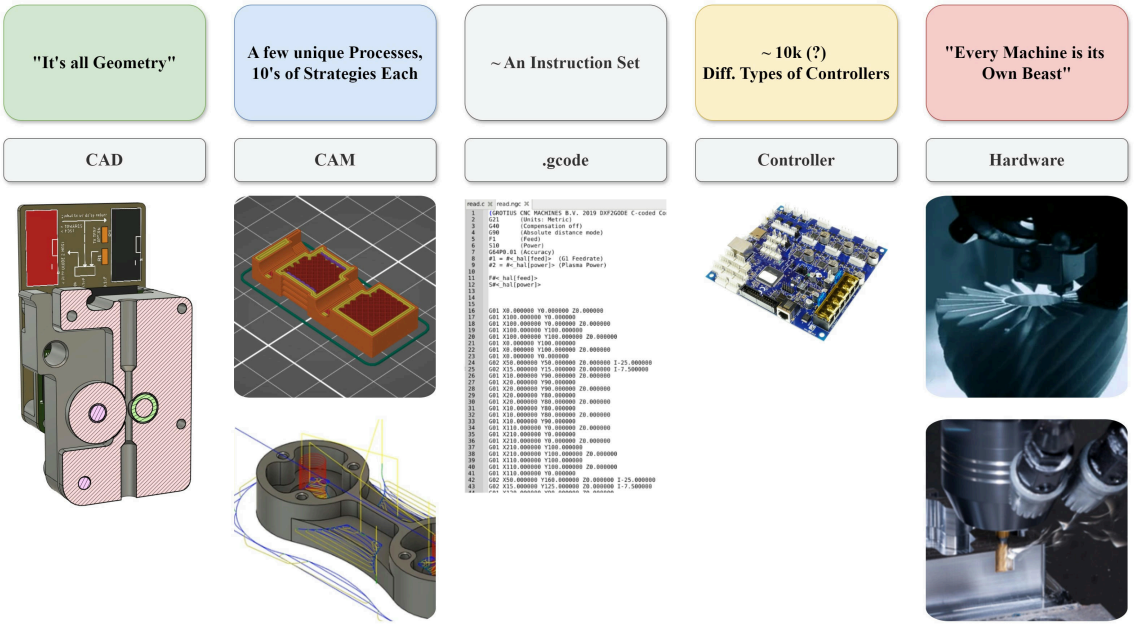


Figure 3: GCodes are written by upstream softwares known as CAM tool (for Computer Aided Manufacturing). In the 3D printing world, these tools are known as "slicers" since they typically process parts layer-by-layer. Since there is no feedback from machines to CAM tools, they must be carefully configured by machine users in order for the whole process to work. This works reasonably well when machines or materials don't change very often, but makes it is difficult to debug when things go wrong (since parameters aren't related to any real physics) and it presents a challenge when new materials or machines are developed. CNC Milling Machines, where multiple cutting tools are used on any given job, configuring CAM parameters is a full-time job because each new tool requires careful consideration.

For example, a CNC Milling Machine itself has no 'knowledge' of the physics of metal cutting, chip formation (see Figure 8), or structural resonances even though these physics govern how a block of aluminum down into a desired shape. In the same way, a 3D printer knows nothing of the rheology involved in heating, squeezing and carefully depositing layers of plastics in order to incrementally build 3D parts (see Figure 6). It is difficult to say whether any computer system has 'knowledge' of anything; in this case I mean that our machines do not use any information about the material physics (and very little about their own physics) they are working with while they operate[3].

In order to operate succesfully, machines need to have some understanding of these physics embedded into the low-level instructions they do receive, but those are developed in CAM tools that *only encode them as heuristics* and as *abstract parameters* that don't have direct correlations to

[3]Where state-of-the-art machines *do* use sensing and feedback it is normally relegated to one subsystem: for example most industrial machines have encoders for positional feedback on each axis, and some high-precision equipment measures temperature along each axis - but these are used in the subsystem that positions that respective axis. In this thesis, I try to close a longer loop across the machines' global control controller, using matched computational models of the whole system (process *and* motion).

process physics (see Section 3.3). This poses a problem for machine builders (who need to carefully tune their motion systems) and machine users (who need to intuitively tune a slew of CAM parameters). It also means that many machines do not operate near their real world optimal limits (since hand-tuned parameters tend to leave considerable safety margins), leaving some performance on the table.

2.3 Machines are Poorly Represented

While most machines don't know *what they are doing,* most machine programmers have a hard time ascertaining and changing *what they are.* A more precise way to say this is that *machine representations are lossy* - they are also often misaligned with reality, and difficult to correct.

At some levels, this is a crushingly simple problem: when we issue a code to `G1 X100` (G1 basically means means "move to" - see Listing 1 for an example of a complete GCode program), we need our mental model of the machine (which axis is "X") to align with the controller's model (and wiring diagram).

That *configuration* is normally locked away in the controller's firmware: we can't query it to see which plug corresponds to which axis, and if we're not using an open source control board we probably can't modify it or read the source code to figure it out. It is simple enough to test when you are physically colocated with the machine in question, it causes all kinds of issues for proprietors of CAM software who need to write GCode for a heterogeneity of machines.

It also causes problems for i.e. educators, hobbyists, and scientists who want to re-purpose off-the-shelf controllers for their own inventions, especially if their machines deploy nonlinear or novel kinematics. One common result of this problem is to find machines in the wild whose controllers are convinced that they are a 3D Printer when in reality they are i.e. a gel extruder [6] or a liquid handling robot [7], or cases where authors have had to develop their own adhoc control systems (rather than re-using available designs) [8], [9].

What a machine builder normally wants is a *computational representation* of their machine that is *aligned with reality* - this is half semantics (any axis is "X" if we declare it to be) and half tuning and model fitting (how are the kinematics arranged, how much torque is available, and what happens if we apply some amount of it over the course of one second?).

In this thesis, I try contribute work that automatically generates low-level interfaces to modular machine hardware in Section 5.4.b - this ensures that machine builders see consistent device abstraction layers when they are programming. On top of that, MAXL (Section 5.1) provides some simple interfaces with which machine builders can describe their systems' kinematics (see Section 5.1.a), and uses an intermediate representation for motion that enables us to rapidly edit kinematic descriptions without recompiling and re-flashing device firmwares.

I am proposing to evaluate these systems by working with machine builders in a plotter-building workshop (Section 5.1.b) to take place this coming January.

2.4 Feedback Machine Systems

The central issue with most state of the art machines is that they are feed-forward devices that contain a lot of built in assumptions about what they are, what materials they will use, and how they should be controlled. Success in this thesis' endeavour would mean that most of the world's machines would be *built **and** controlled* using the principle of *feedback.*

In ***building** with feedback* I mean using systems (like those prototyped in this thesis) that can automatically generate computational representations of their real instantiations. There are two main aspects to this.

The first is to generate the very nuts-and-bolts communications interfaces that we use to connect to modular hardware (I discuss network interfaces in Section 5.4 and software interfaces in Section 5.4.b). This could prevent the laundry list of errors that arise when a programmer's model of the hardware is mismatched to the real world, and tools for reflective programming[4] may make it easier to write flexible software for heterogeneous hardware.

The second is the generation of models for motion (see Section 5.1, Section 5.1.b and Section 5.1.a) and for process physics (see Section 5.2.c). This should culminate in the (mostly) automatic generation of any given machine's *digital twin.*

In ***controlling** with feedback* I mean generating control algorithms that use *automatically aligned models* to optimize machine motion and process outputs (motor torques, heater voltages, etc). I discuss this in detail in Section 5.1 and reduce it to practice in the Rheo Printer (Section 5.2) with the key result that the use of models allow us to greatly reduce the space of input parameters that users need to provide (Section 5.2.d). This is effectively the outer loop pictured in Figure 1: we use sensor-equipped machines to fit models for process and motion, which we then use to optimize controller outputs. I have a proof-of-concept for this approach with the Rheo-Printer that I am aiming to deploy and extend for continous (layer-by-layer) learning (i.e. building and updating models as a print progresses). I am also proposing to extend this technique to a second machine and process in CNC Routing (Section 5.3), where I hope to show that the same approach can help us cut parts quickly in many materials while avoiding resonant chatter.

Taken together, these contributions will enable machine builders across lab automation, industry, STEM and in the arts to make machines that are more rapidly composable, more robust to change, and more available for hacking. Machines developed using these strategies should be easier to operate and tune, harder to break, and easier to integrate into larger workflows.

2.5 Why the CBA, and MAS

The Center for Bits and Atoms has been the site of nearly two decades of machine-building research and the birthplace of numerous machine companies. Our sponsors include companies like Hurco (who make machine tools), Autodesk and Solidworks (who make CAM software), and NIST (who set measurement standards in the USA). We also help to coordinate a global network of Fab Labs, where machine building is taught to beginner hardware engineers [10]. At MIT, we teach

[4]By *reflective programming* I mean approaches that are introspective: that check available resources at runtime before using them. In a way, this is synonymous with *feedback programming* - codes that look at codes.

a machine building class [11] and an intro to fabrication class [12]. This gives us unique insight into the real-world problems faced by practitioners in the machine buiding disciplines - and users of machines.

Machine building involves the merging of many disciplines: meche, ee, controls, modeling, and in the case of this thesis also embedded networking, programming systems, high performance computing - CBA has covered each of these domains in its history. Indeed, this thesis is essentially one monster systems integration problem that I am enabled to do not because I am an expert in any of the constituent disciplines, but because I have peers and friends who are nearby in the CBA and MAS who can guide me in the right directions: for compiler and programming and optimization quandaries I talk to Erik Strand or Sean Hickey, for architectural insights and *difficult maths problems* I talk to Quentin Bolsee, and for MechE and Circuit troubles I find Zach Fredin, Miana Smith, Alfonso Parra Rubio, and Alan Han.

3 Background

I want to start with the a systems-level overview of the state of the art *in practice* as it pertains especially to CNC Milling machines and FFF 3D Printers, starting with a primer on GCode (Section 3.1), then comparing how state of the art CAM and Control relate to the physics they are wrapped around (Section 3.3 and Section 3.3.c). Finally, I will take a broader look at background for the optimization (Section 3.4), modelling (Section 3.5 and Section 3.6) and architectural (Section 3.7) work that I do in this thesis.

I want to note again that many of the *components* of the systems I have developed as part of this thesis are present in the literature: for example many models exist to explain the rheology of FFF printing (Section 3.5) and to optimize motion (Section 3.4), but to my knowledge no-one has assembled these all into one online system, and certainly these approaches are not today available in off-the-shelf solutions. The central contribution in this thesis (a controls framework that combines process and motion optimization Section 5.1) is nowhere to be found in the literature. I cannot verify that it has not been done internally by some commercial machine-building firm.

3.1 A Small GCode Primer

Understanding how GCode works, what niche it fills, why we might want to replace it (and why it is so pervasive) is important background for this thesis. To that end, let's look at a basic GCode "program" that, for example, cuts a square out of a piece of stock using a router (a subtractive tool).

Listing 1: An example GCode program that cuts a small square from a 1/8” piece of stock using a CNC Router.

```
L01   G21                   ; use millimeters
L02   G28                   ; run the homing routine
L03   G92 X110 Y120 Z30     ; set current position to (110, 120, 30)
L04
L05   G0  X10 Y10 Z10 F6000 ; "rapid" in *units per minute*
L06
L07   M3  S5000             ; turn the spindle on, at 5000 RPM
L08
L09   G1  Z-3.5 F600        ; plunge from (10, 10, 10) to (10, 10, -3.5)
L10   G1  X20               ; draw a square, go to the right,
L11   G1  Y20               ; go backwards 10mm
L12   G1  X10               ; go to the left 10mm
L13   G1  Y10               ; go forwards 10mm
L14   G1  Z10               ; go up to Z10, exiting the material
L15
L16   M5                    ; stop the spindle
L17   G0  X110 Y120 Z30     ; return to the position after homing (at 6000)
```

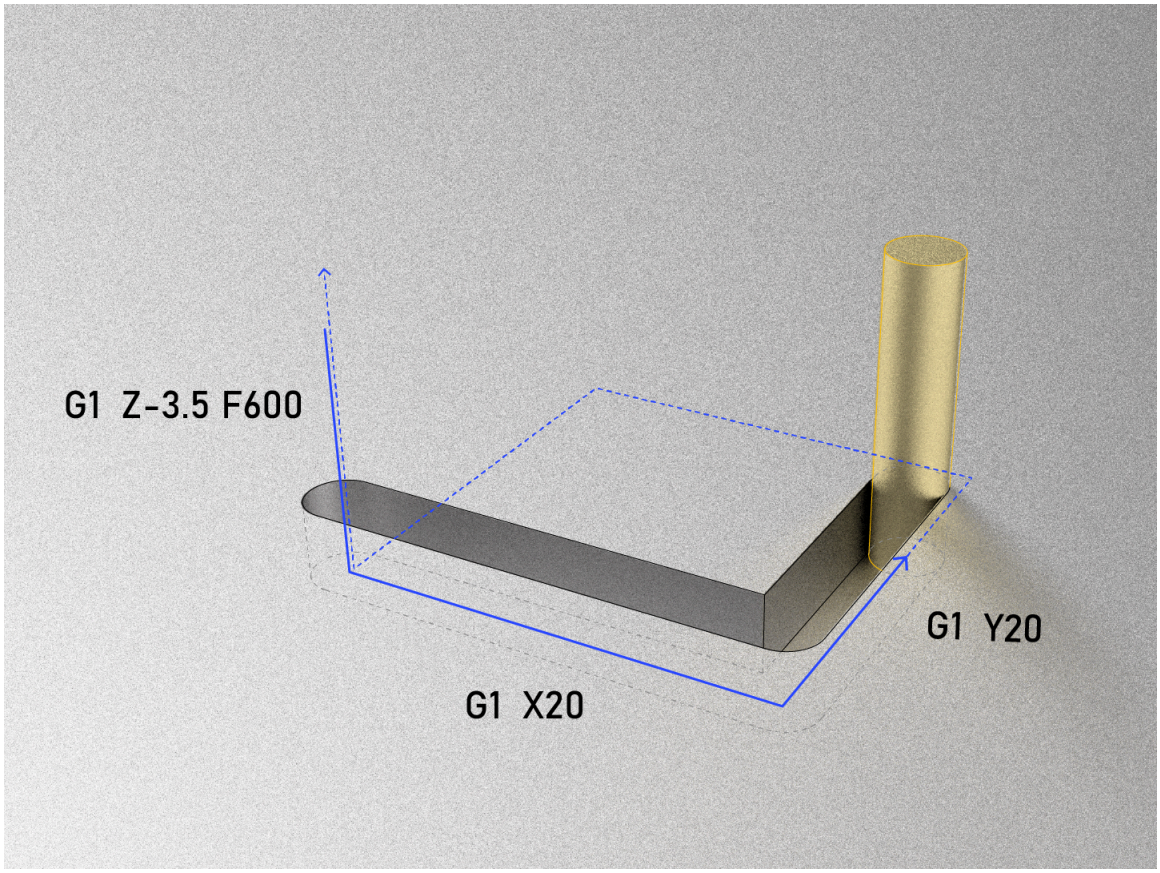


Figure 4: The actions (if properly configured) of the GCode snippet from Listing 1.

GCode was developed *around* the same time (and in the same place) as programming languages, compilers, and instruction sets. Programming languages allow software engineers to design and describe algorithms without recourse to their actual implementation in low-level codes [13] (handing that problem instead to a compiler), and compilers rely on instruction sets (ISAs) to generate low-level codes without concern for how they are actually implemented in hardware (see Figure 3).

GCode was meant to take the same place in machines, allowing machine programmers (formerly *machinists*) to develop manufacturing routines that could be deployed on variable hardware [14]. This probably seemed like a good idea at the time, but it is stymied by two important facts.

Firstly, while there are only a few ISAs for compilers to contend with (and even then, a real homogeneity around x86 and ARM in practice), there are perhaps tens of thousands of unique GCode interpreters each with a unique "flavor" of GCode. I diagram this relationship in Figure 3. For example, in `L10` above, we tell the machine to move the `X` axis 10mm to the right (from its previous position established in `L05`), but it is impossible to know from the code alone which motor-controller is actually going to execute this move; there is a system configuration that is hidden from us. This seems like a simple problem (simply write down the configurations, right?) but is a major hurdle for companies like AutoDesk, whose CAM software (discussed next) must interface to many machines, and needs to know exactly how (for example) the X axis moves relative to the rest of the machine. The same problem is even more problematic for advanced machines, like multi-DOF mill-turn "screw machines," where most users result to programming jobs manually (i.e. writing GCodes directly) - a clear failure of what is meant to be a low-level layer.

Secondly, physical processes themselves are also heterogeneous: even if we run the same job on a machine thousands of time, each is bound to be different: cutting tools wear out, incoming stocks are of slightly different sizes and compositions, external factors like heat soak and ambient air temperature all change. Whereas computer science goes to great lengths to ensure the homogeneity of the lowest layers of its stack (the implementation of ISAs), it is simply not possible in manufacturing to do the same. All the while, GCode provides no recourse to use real-time information in the execution of manufacturing plans: if we want to describe an algorithm that controls a machine intelligently based on physical measurements, GCodes are simply not an option.

GCode also contains a mixture of simple instructions like G1 (goto position) alongside larger subprograms like G28, a homing routine that may involve coordination of many of the machine's components.

3.2 Equivalent Codes using MAXL and OSAP

In the paradigm introduced in this thesis, the same program as presented above in Listing 1 is a 'real' program (a python script) rendered in Listing 2, and the homing subroutine is available for inspection Listing 4.

Listing 2: The equivalent low-level program as the GCode presented in Listing 1, here written using the python API presented by MAXL. While these are more verbose, they are semantically meaningful and reduce hidden state.

```
await machine.home()
machine.set_current_position([110, 120, 30])
await machine.goto_now([10, 10, 10], target_rate = 100)

await spindle.await_rpm(5000)

await machine.goto_via_queue([10, 10, -3.5], target_rate = 10)
await machine.goto_via_queue([20, 10, -3.5], target_rate = 10)
await machine.goto_via_queue([20, 20, -3.5], target_rate = 10)
await machine.goto_via_queue([20, 10, -3.5], target_rate = 10)
await machine.goto_via_queue([10, 10, 10], target_rate = 10)

await spindle.await_rpm(0)

await machine.goto_now([110, 120, 30], target_rate = 100)
```

Listing 3: Low level codes can be contained in higher order functions like this one, that presumably contains much of the same logic as that rendered directly in Listing 2. Living in a complete computing language means that we can readily add useful abstractions like this to our systems.

```
await machine.route_shape(svg = "target_file/file.svg", material = "plywood,
3.5mm")
```

Listing 4: This is MAXL's internal logic for limit switches, itself composed of a mixture of MAXL API calls and other hardware interfaces (to see how those are generated, see Section 5.4.b). Living in python means that we can easily move between big system layers (like Listing 3) or simple layers like this one.

```
  async def home(self, switch: Callable[[], Awaitable[Tuple[int, bool]]], rate:
float = 20, backoff: float = 10):

    # move towards the switch at <rate>
    self.goto_velocity(rate)

    # await the switch signal
    while True:
        time, limit = await switch()
        if limit:
           # get the DOF's position as reconciled with the limit switch's actual
trigger time
            states = self.get_states_at_time(time)
            pos_at_hit = states[0]
            # stop once we've hit the limit
            await self.halt()
            break
        else:
            await asyncio.sleep(0)

    # backoff from the switch
    await self.goto_pos_and_await(pos_at_hit + backoff)
```

Our codes are longer because they embed more information, and they are *semantically meaningful* - for example, I haven't had to use comments in the scripts above because the function names themselves are human-readable. We can also easily describe dynamic or interactive controllers in this paradigm, as is the case with the example below (Figure 5 and Listing 5), where we use CV in conjunction with MAXL to 'play' a robot xylophone.

Figure 5: **Click here for a video.** In this demo, I worked with Quentin Bolsee who authored a computer vision system that detects a user's fingers above piano keys - that vision system is integrated with a MAXL motion controller to position the hammer of a robotic xylophone (at left) before the key is struck. MAXL's flexibility allows us to generate complex machine systems like the Rheo Printer (Section 5.2), or use a subset of the available components to build systems like this. It also lets us generate interactive controllers that interface to more complex algorithms like Quentin's vision system in real time.

Listing 5: The key code snippet from an interactive robo-xylophone demo that I produced with Quentin Bolsee as a part of HTMAA [12] Machine Week '24. In this snippet, Quentin opens a connection to a separate python process (not figured) that calculates his fingers' positions from a video frame. That process sends commands to this snippet, which interfaces directly with MAXL to control the machine hardware.

```
async def handle_echo(reader, writer):
    print("Connected")

    stop_requested = False
    while True:
        data = await reader.read(100)
        if not data:
            break

        msg = pickle.loads(data)
        if msg.get("running", False):
            stop_requested = True
            break

        reply = {"ACK": True}
        writer.write(pickle.dumps(reply))
        await writer.drain()

        if "hit" in msg and msg["hit"]:
            using_a = True
            if "note" in msg:
                p = note_to_pos(msg["note"])
                pa = dof_a.get_position()
                pb = dof_b.get_position()
                if abs(pa-p) < abs(pb - p):
                    await dof_a.goto_pos_and_await(p)
                    using_a = True
                else:
                    await dof_b.goto_pos_and_await(p)
                    using_a = False
            if using_a:
                await fet_a.pulse_gate(0.85, 6)
            else:
                await fet_b.pulse_gate(0.85, 6)
        else:
            if "note" in msg:
                p = note_to_pos(msg["note"])
                pa = dof_a.get_position()
                pb = dof_b.get_position()
                if abs(pa-p) < abs(pb - p):
                    await dof_a.goto_pos(p)
                else:
                    await dof_b.goto_pos(p)

    writer.close()
    print("Close the connection")
    await writer.wait_closed()
    print("done")
    if stop_requested:
        flag_running.set()
```

To a complete newcomer to both machines and programming, the comparison is kind of mute: both are new codes, either of which would take some time to comprehend. But python (and programming in general) is more of a lingua franca in science (and especially in the discipline of controls) than GCode (which is a niche language we encouter only in the context of digital fabrication equipment in particular). Python also lends itself to systems integration more readily: the language is attached to a package manager [15] that contains some tens of thousands of libraries including powerful machine learning algorithms (that I deploy in this thesis) and computer vision. The users we are interested in helping to develop new machines are probably "already here." Conversely, many machine users unwittingly begin to learn some principles of computing when they learn how to read and write GCode - they could just as easily be learning the "real deal" during their practice.

3.3 CAM: Parameters vs. Machine Control

In Section 2.2 I discussed how CAM (software used to generate machining plans: 'slicers' and CNC Milling software) and control are separated in the state of the art.

It is worth mentioning that when someone *learns how to make things with digital fabrication,* much of their effort is devoted to learning how to use CAM softwares. Each has its own quirks, but most follow a similar pattern: a 3D file is imported and positioned in a virtual work volume, and then various parameters are configured such that the software can generate a *path plan* (basically, GCode) for the selected machine. Figure 7 and Figure 9 give examples of printing and milling parameter sets respectively.

In this thesis I am trying to provide an interface between the two sides of machine operation using physical models, but in the state of the art CAM tools make extensive use of *parameters* to configure their behaviour. In this section I want to show that those parameters are often disconnected from the physics that governs machine behaviour and machine control.

3.3.a Direct Parameters vs. FFF Phenomenology

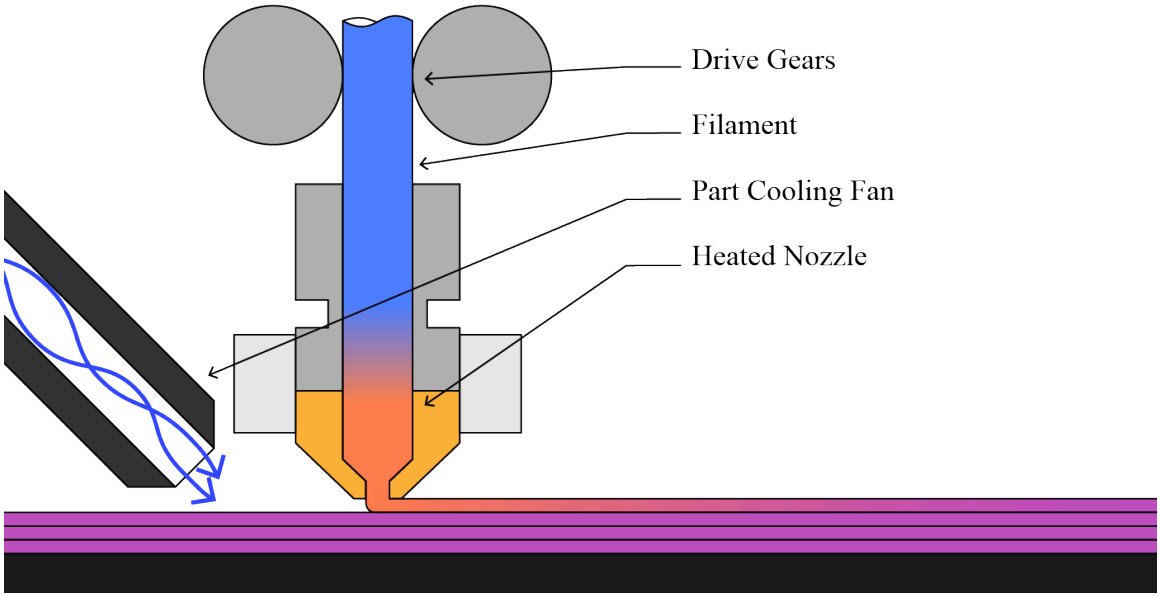


Figure 6: The physics of 3D printing is simple in principle (heat, then squish filament out of a nozzle) but can be complex in practice. Filament between the drive gears and nozzle exit compresses before it extrudes, putting some phase lag in the system. The amount that extrudes is a function of the chamber pressure, but also of the flow temperature and material properties. The actual flow temperature is history dependent: if we have been extruding near the machine's limit, melt flows are normally colder than the nozzle's measureable temperature (since the filament does not spend much time in the heat zone). To add to all of this, filaments exhibit *die swell* - when they are extruded at high pressures, they retain some memory of their previous width (as a 1.75mm diameter rod) and try to expand back to that width as they cool, putting residual stresses in the print.

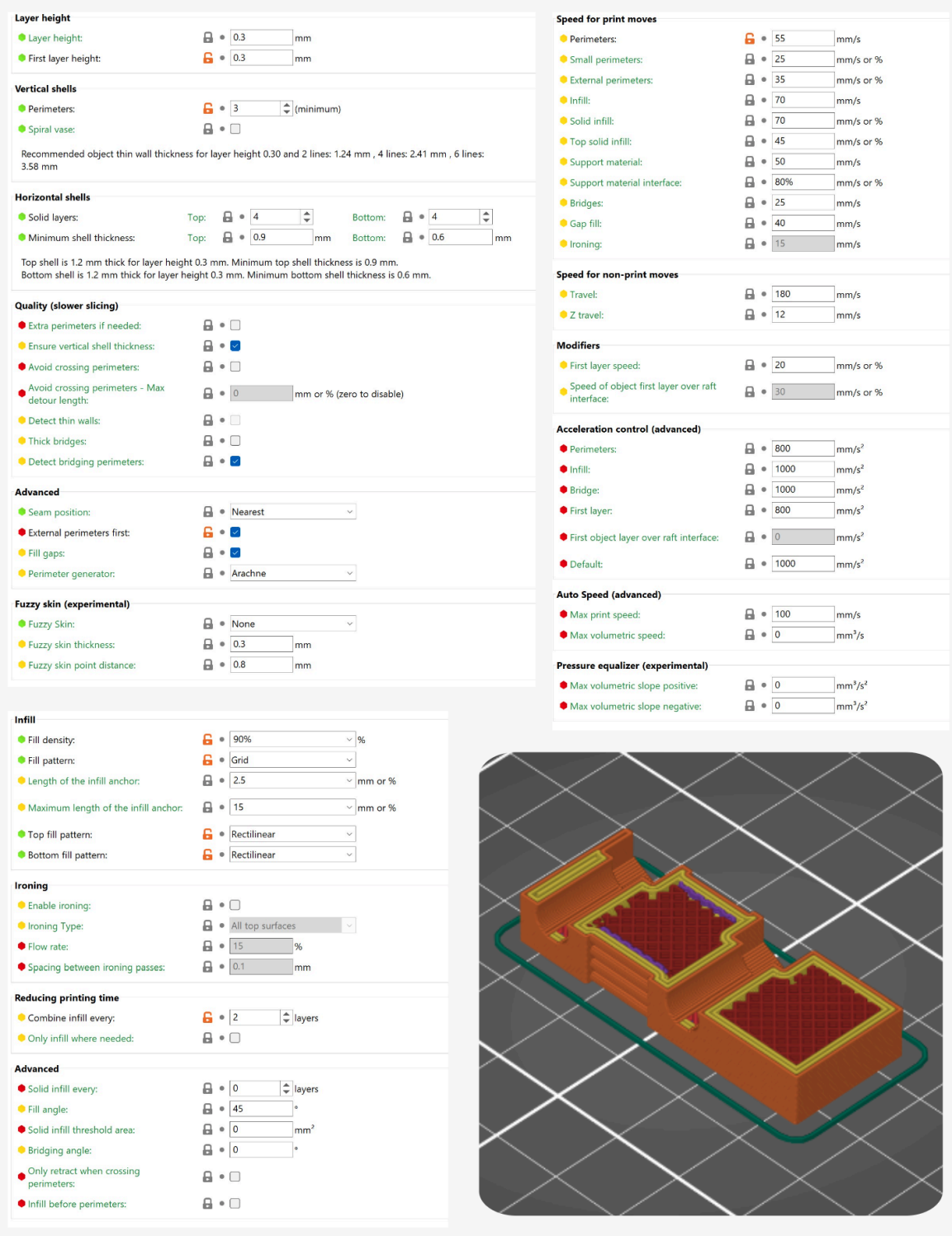


Figure 7: A screenshot of parameters available to tune an FFF print job, from PrusaSlicer [16]. The set which is displayed is incomplete; there is another page of parameters for extrusion temperature, print cooling fan settings, and bed temperature.

Let's look at 3D Printing: users select some geometric parameters: layer height, track widths, external perimeter counts, and infill patterns and percentages. They also select speed parameters: linear feed rates (how fast the nozzle moves in cartesian space) for different features: outside perimeters, interior perimeters, etc. Finally, the temperatures: for the nozzle, for the print bed, and (sometimes) for the chamber. These all relate directly to the machine instructions (how fast to travel, how thick to make the layer, etc).

But the outcome of these plans is not related to machine instructions, it is related to the *physics* of 3D Printing, which is mostly concerned with *flow rates* and temperature of the polymer melt (see Figure 6 above) [17]. For example if we double the layer height but retain the same speeds we effectively double the flowrate of the polymer melt: this has huge ramifications for the physics of the process, but that is not reflected anywhere else in the parameter set. Changing print speeds to reflect the new layer height would mean updating *six* other parameters (excluding those that can be set by a percentage). Nor are those parameters related to longer-term outcomes like the resulting thermal history of the print's inter-layer joints, which are a main indicator of part strength [18]. Inversely, if we increase the nozzle temperature we often unlock more flow rate, but there is no way to update print speeds as a function of temperature. Instead of having any knowledge of print physics built into the slicer, users need to intuit how these many parameters will relate to the machine's operation.

3.3.b Direct Parameters vs. CNC Milling Phenomenology

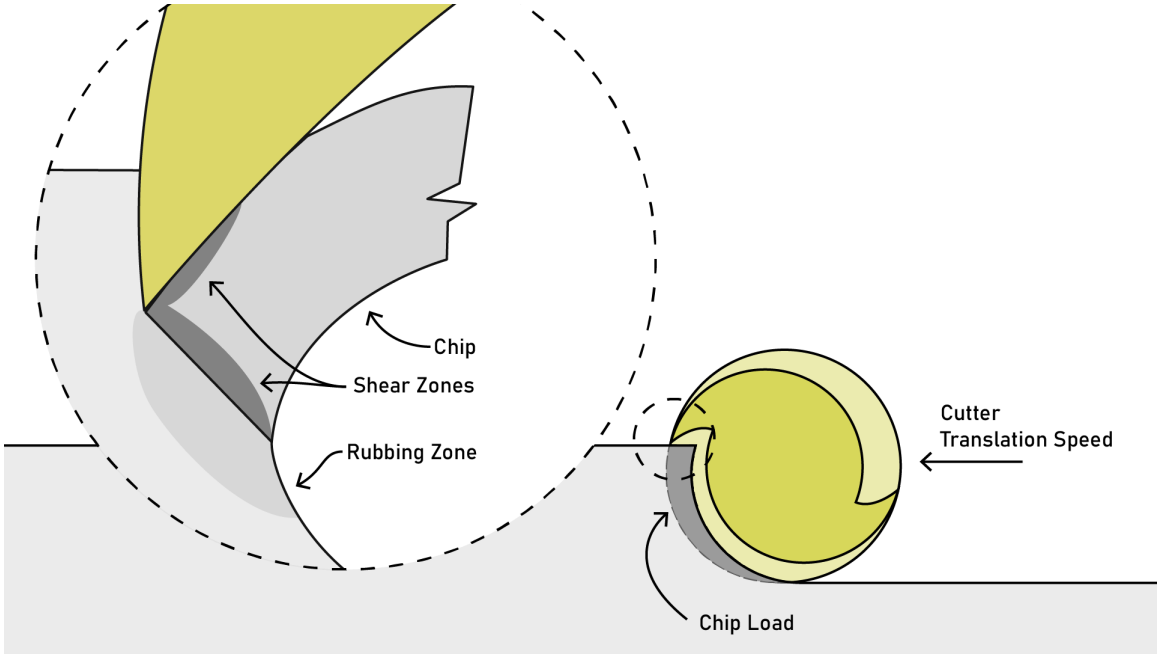


Figure 8: The physics of maching are arguably more intense than their counterparts in FDM, involving extreme shear rates, thermodynamics, tool life and geometry. The *core* of it is about chip load, which depends on the ratio between spindle RPM and translational velocity. Cutting a chip that is too big means too much cutter force (leading to broken tools and an underpowered motion system), not enough chip load means that the physics becomes are rubbing dominated and tools basically dull themselves to a shorter lifespan.

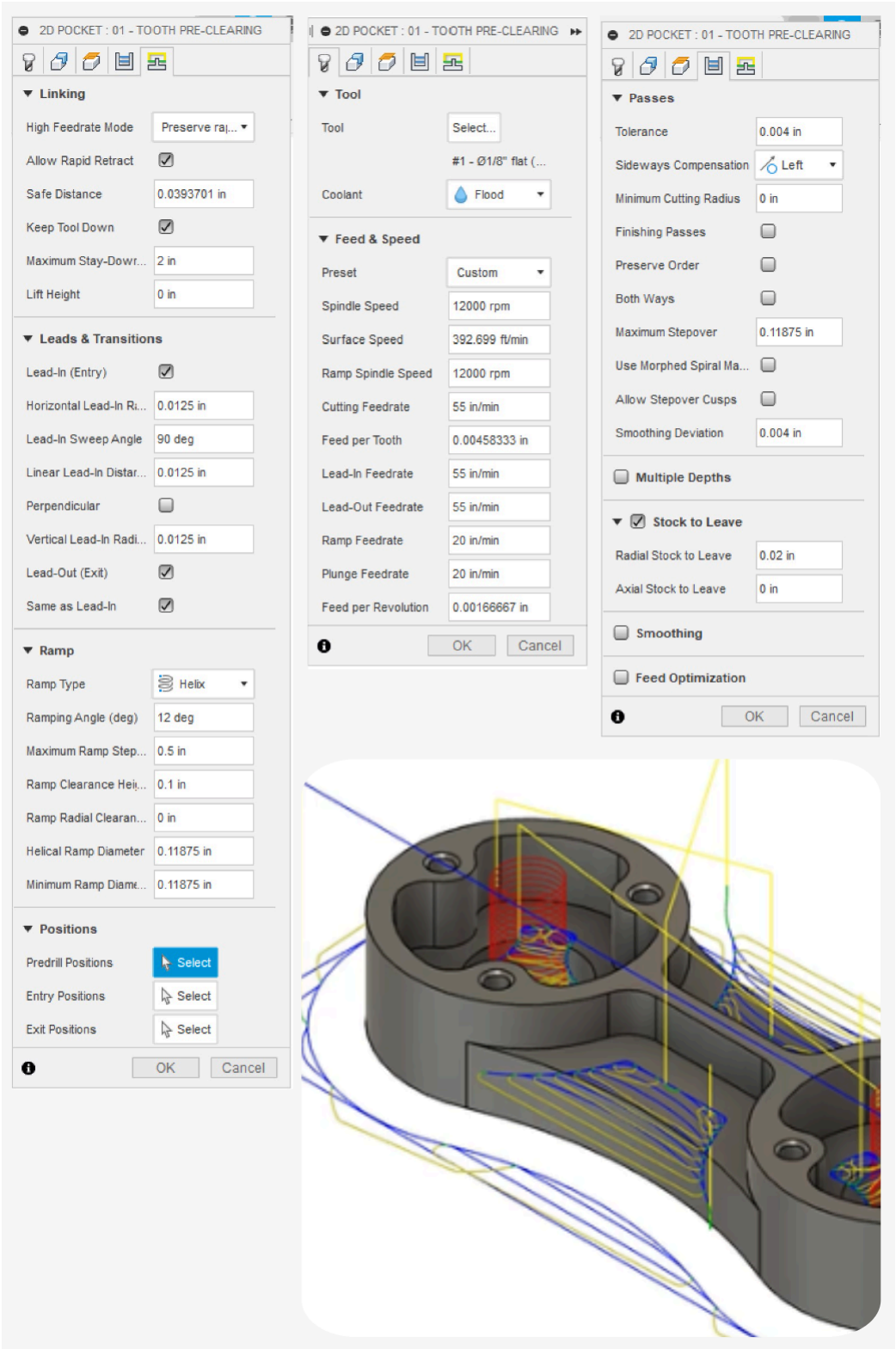


Figure 9: This is a screenshot of the parameters available to configure one cutting strategy (a 2D Pocket) in Autodesk Fusion [19].

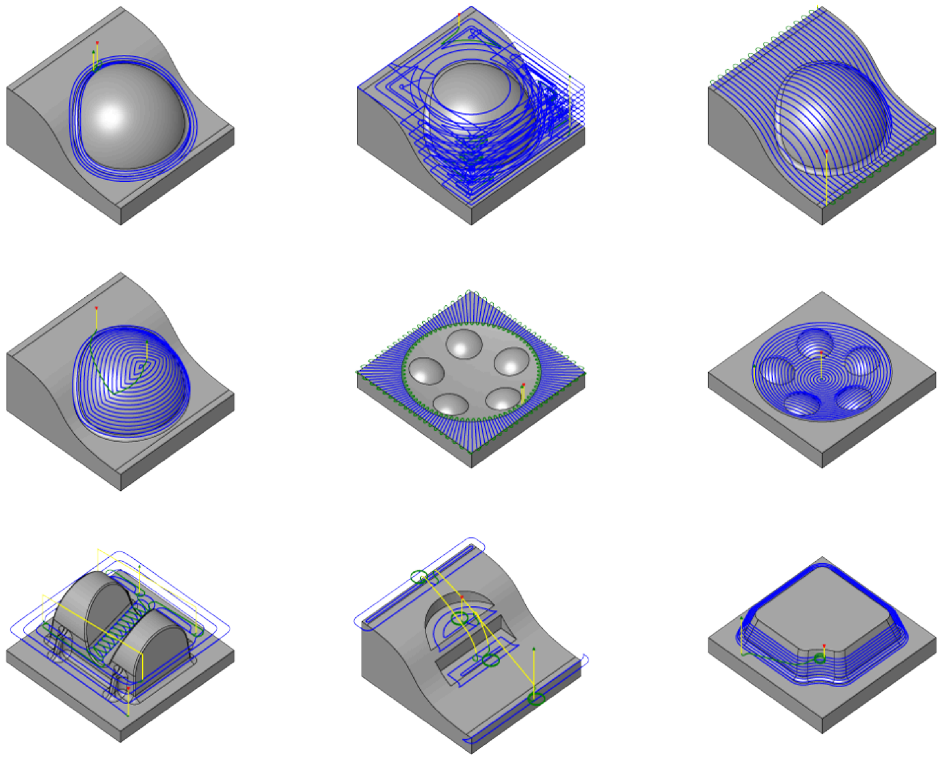


Figure 10: These are just a few of the many CNC machining *strategies* available in one modern CAM tool (Autodesk Fusion). Each strategy has some unique parameters to tune, though most reuse a similar subset.

CAM for 3D Printing is at least simpler than CNC Machining because each job is one operation: slice into layers, produce perimeters and then infill for each layer. A machining job can include many (tens or hundreds) of different path planning strategies and tools for different geometric features, each selected manually. For example a tightly toleranced hole may require the machinist to select a purpose-made reamer, threaded features require threading tools, radiused or chamfered corners require fillet or chamfer end-mills, etc. I show a selection of these strategies in Figure 10.

The physics of CNC milling is the physics of chip formation, which is mostly related to *surface speed* (how fast the cutter edge is moving with respect to the material), and *chip load.* These are both functions of the tool diameter, spindle RPM, as well as the machine's linear traversal rate (which are the parameters we set in most CAM tools). Chip formation then generates overall loads on the machine's structure that is related to the depth and width of the cut. Most complex of all in CNC Machining is the issue of resonance: any machine has a given natural frequency and if chip formation happens to excite the machine near that frequency, the whole system vibrates, causing innaccuracies or even tool breakage.

Milling parameters, then, are similar to FFF parameters: a user sets geometric and speed-related parameters that are only indirectly related to the underlying process physics that govern the system. Instead they are directly related to the low-level instructions (GCodes) that we send to the machine.

Essentially, the state of the art workflows have users writing low-level parameters directly, rather than describing high level goals or behaviours. *This is akin to writing computer programs by directly authoring assembly language.*

3.3.c The Hidden Optimization

Expert CAM users know that process physics alone are not enough to intuit where a job's parameters should be set: the machine's physics also present constraints. Milling machines need to be strong and stiff enough to handle generated loads without deflecting or vibrating, and printers have limited top-end speeds and accelerations depending on their kinematic arrangements, the flying mass of the end effector, etc. Printers also have a stiffness issue: when timing belts are loaded rapidly they also vibrate - this limits how fast a perimeter can be printed without seeing "ringing" artefacts in a printed part.

This is where we find perhaps the most eggregious hidden actor in the parameter tuning journey: controller firmwares perform an optimization on the instructions they are given so as to avoid issuing instructions to the hardware that would be impossible to achieve: instantaneous changes in speed and direction, or instantaneous changes to i.e. flowrates, spindle speeds, etc. To make things more complicated, this step needs to happen in a near real-time computing environment, i.e. controller firmwares that are difficult to interrogate or modify.

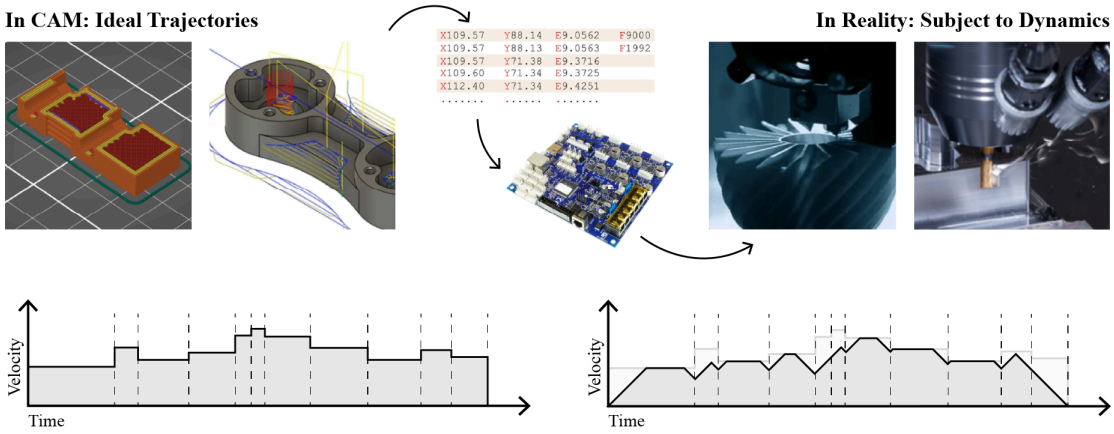


Figure 11: Machine controllers perform a real-time optimization of the *speeds and feeds* that are requested of them in GCode files. This optimization is effectivley a speed scaling that prevents excessive loads on the machines' structure and motors according to pre-configured limits to acceleration and velocity.

All of the physics we discussed are tightly coupled to the machine's speed. If we slow the printer down, we also need to slow down extrusion rate, and if we cannot extrude fast enough to pro-

duce a track of a given width, we need to slow the printer down. This is handled easily enough in printer firmware using proportional speeds, but extrusion is dynamic: we need to compress the filament between the drive gears and the nozzle before we generate enough pressure to see any flow. A somewhat recent advance in FFF printer firmwares is to compensate for compression of the filament between the drive gears and the nozzle using an approach called *linear advance* [20]. However - this is a *firmware* setting. Not all controllers allow this to be modified without re-compiling a firmware, and those that do allow this need the slicer to tell them where to set it. However, filament compression is not just dependent on the filament (whose stiffnesses vary wildly), but on the printer design (some have better extruder designs than others). To add to the complexity, *maximal flow rates* (which are often the key limiter to print speed [21]) are not even dependent on the filament alone: they are a complex function of the extruder's design, motor torque, historical flowrate, hotend thermodynamics, etc.

With CNC milling, deccelerating into a corner can change chip size dramatically, but because we cannot rapidly slow down spindle RPM (too much inertia!) we cannot simply scale the spindle RPM as a function of machine speed. Instead, we simply have a lossy layer: where we may setup a job to run at a particular chip size, some large proportion of the work might be done at much lower speeds with a much smaller chip size[5].

In the figure below, I show the actual feedrates realized by a classical motion solver (using trapezoids) under two conditions: one sets a target of 100mm/sec, the other targets 500mm/sec. We would expect that the second is five times faster, but in reality it is only **14%** faster because the solver's maximum acceleration is limits to the same maximum acceleration of 1000mm/sec^2 in both cases. This is a simple example of what I mean by a *lossy* layer: the parameters that we set in any given CAM tool are often not realized in practice, something that has stymied other authors as they develop optimizations for milling machines [22].

[5]Many readers will be familiar with High-Speed-Machining, a series of milling strategies developed somewhat recently. These generate tool-paths that effectively minimize cornering radii for the majority of the work that they do. These were a kind of magic when they were introduced, and were realizeable only when modern machine controllers could readily handle the complexity of the instructions produced. It is my intuition that this insight was not so much about maintaining high feedrates, but that the resulting geometries mean that *more of the actual milling time* is spent *at the target feedrate* - i.e. tight corners, which produce feedrates lower than those set in parameters, were minimized.

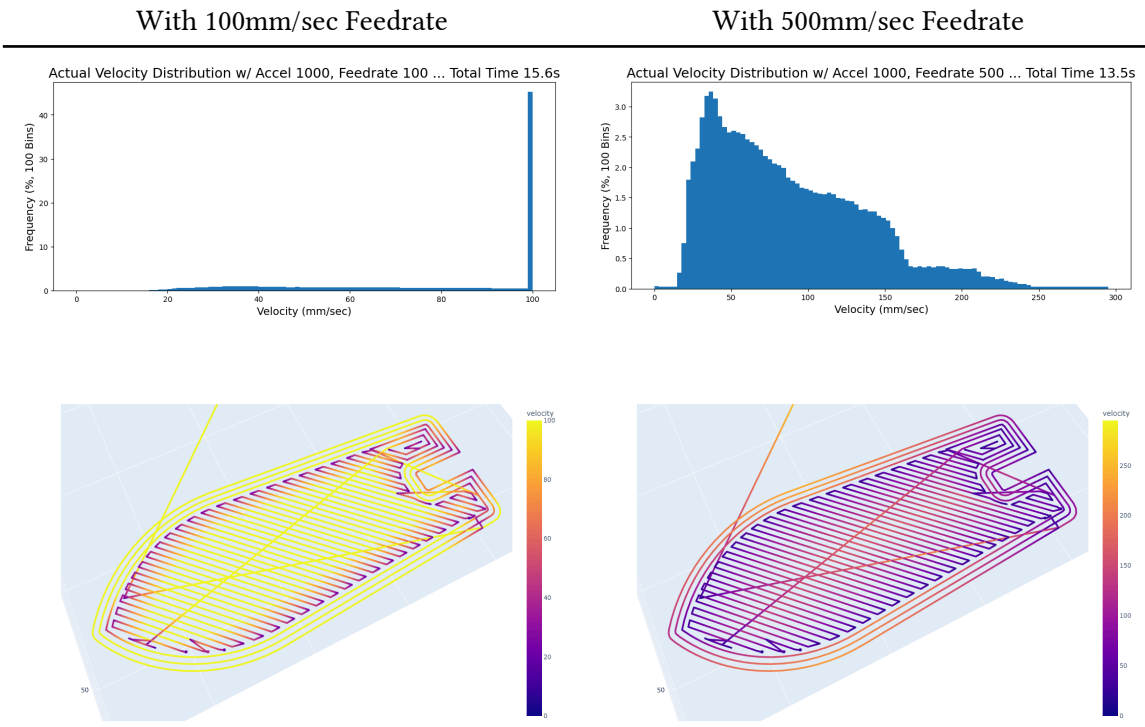


Figure 12: Here I show the result of two print layers that have been optimized by a trapezoidal motion solver (the likes of which are present in most machine control firmwares). In both cases, acceleration is limited to **1000mm/sec^2.** At the left, we target 100mm/sec feedrate and the histogram shows that the resulting path is executed mostly at that feedrate. On the right, we target 500mm/sec, but see that this speed is *never reached* during the print: the job is acceleration dominated and instead of seeing a 5x increase in speed, we realize only 14% increase.

Switching directions quickly is important for any machine, and machine builders have developed novel kinematics to increase performance in this regard. One of which is CoreXY [23]. CoreXY (and many other arrangements) is *anisotropic* in its physical speed and acceleration limits: it has to move much less mass in the X direction than in Y - while it has just as much available motive force in either direction. This means that a part's orientation with respect to the machine's kinematics can be hugely important for overall print speed, as shown below in Figure 14. Slicers, however, are unaware of this property, meaning that they can't optimize print speeds by (for example) changing the primary direction of an infill pattern.

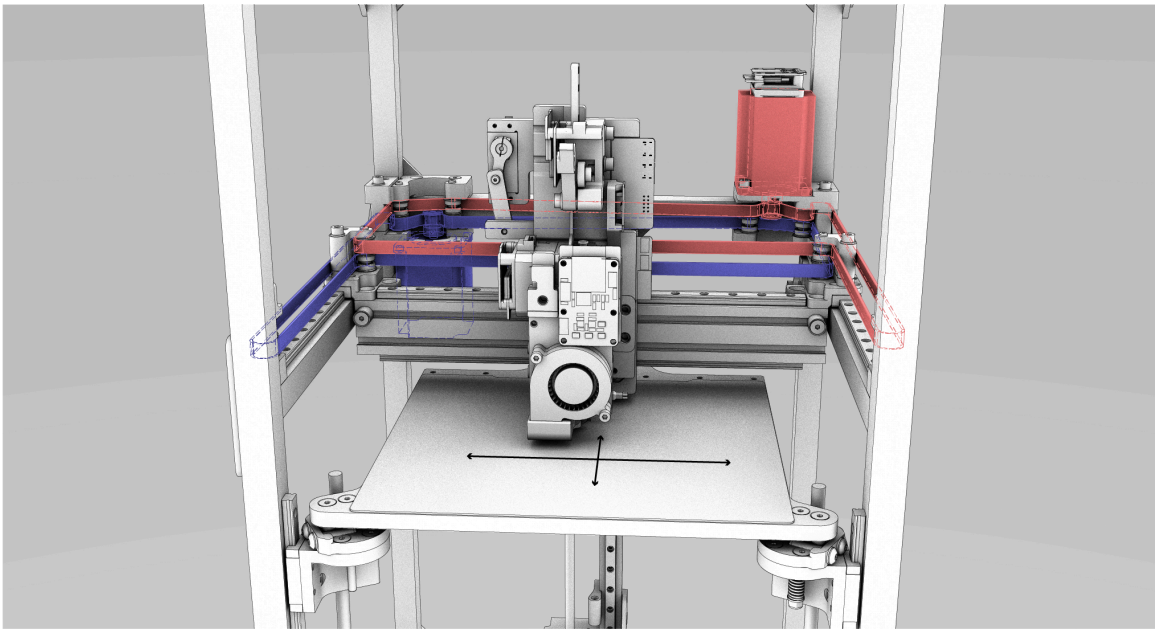


Figure 13: Like many machines, CoreXY layouts have anisotropic dynamics. Both motors work together to move the machine in X and in Y, but the moving mass in X (just the end effector) is significantly lower than that in Y (which includes the end effector along with the y-beam).

With Infill Misaligned to Anisotropic Acceleration Limits	With Infill Aligned to Anisotropic Acceleration Limits

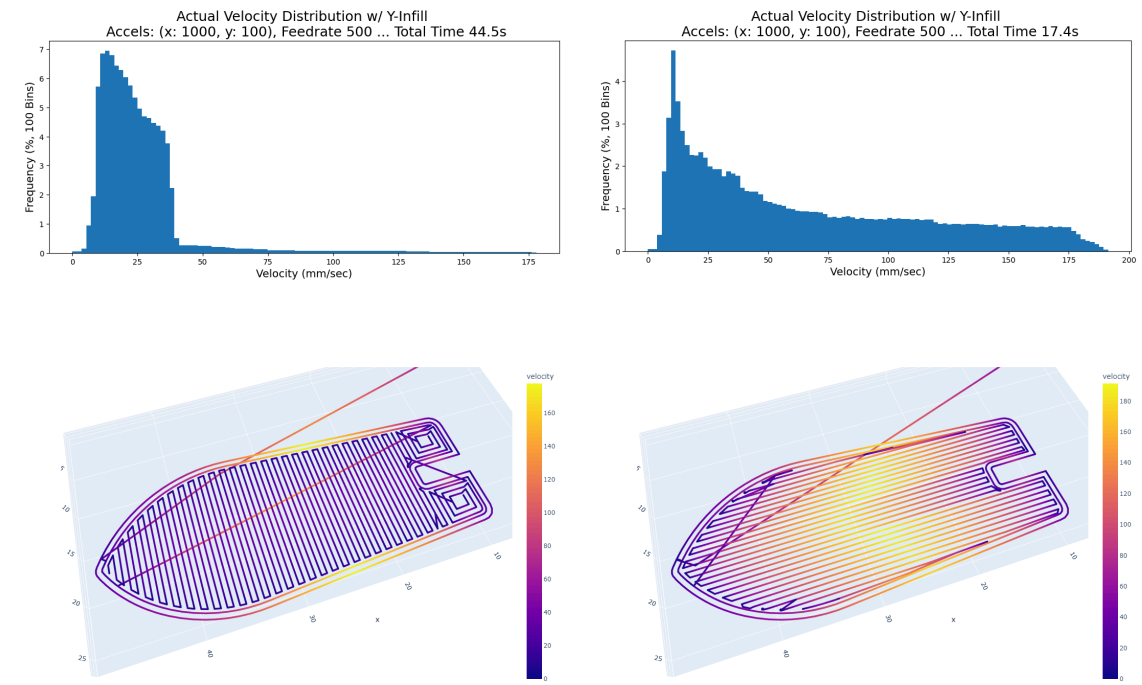


Figure 14: In this example, I show how a machine with 10x more command over its X-axis speed can cut 2/3rds off of a print time by aligning infill with the dominant axis. On the left, I show a speed histogram and heatmap for a print layer with infill in the Y axis, and at the right, the same system with infill aligned on X.

Between this 'hidden' speeds optimization and parameters that are disconnected from process physics, state of the art workflows for FFF printing and CNC machining have important components spread across multiple disconnected systems. Some are implicit in CAM parameter space and some are governed by the machine's firmware. Getting a machine to operate successfully involves the user aligning all of these settings intuitively, requiring them to implicitly understand process physics physics and machine physics, but also to understand how parameters and firmware configurations will translate into real-world operation of the machine.

3.4 Constrained Optimization and Differentiable Simulation

State of the art machine workflows are implicit constrained optimization problems: users want to make their parts *precisely* and *quickly,* but they are limited by process physics, material properties, and machine dynamics. As I've explained, this optimization is currently distributed across two disconnected systems (CAM and the Controller), and is articulated somewhat awkwardly with indirect parameters leading to difficult-to-discern differences in outcome. The optimization is not made explicit anywhere, nor are the physics written down anywhere - people rely on heuristics above all and the whole process is feed forward: we tune systems until they break.

In this thesis I propose to replace big parts of these workflows with models and numerical optimization, and bring them together into accessible, interpretable programs. I am enabled to do this largely because of new availability of *autograd* tools, that automatically generate gradients for *any given program* (a key ingredient for optimization). In particular, I am using JAX [24]. With JAX, any system that can be articulated as a purely functional python code can be automatically differentiated. We can then use the function's gradient to optimize the function's inputs, and JAX conveniently pairs with optimizers from the OPTAX library [25]. The performance of this system is only viable because JAX also includes a just-in-time compiler that turns the whole system into code that can run in parallel on high performance hardware (i.e. a GPU).

MAXL uses JAX as a basis to implement a Model Predictive Controller [26]. MPC controllers work in much the same way as you and I do: it uses a simulation of the future to optimize future and current control outputs. The classic example comes from driving: when we are entering a corner, we know that we need to slow down before we turn. To pick a braking point, we use a mental model of our car's dynamics (how long it takes to slow down) to "simulate" (imagine) how long it will take to come to the appropriate entry speed for the corner. With computational MPC, we simulate and optimize control outputs over a horizon of up to a few seconds *at each time step,* but only issue the immediate next control output to our system. For example in the solver that I have implemented, I optimize 250ms of control outputs at a 4ms interval, every 4ms. MPC is slightly more complex than other common controllers because it involves building system models, but its flexibility makes it well suited to many tasks, probably the most popular of which is in robot quadrupeds [27] quadrotors [28] and humanoid robots [29].

Most MPC researchers use software libraries for these tasks: CasADi [30] and ACADOS [31]. CasADi lets developers author symbolic representations of their system dynamics and cost functions, the package then generates C code for evaluating derivatives and integrating dynamics models. ACADOS is a solver that can directly use those C codes to minimize cost functions in real-time. This follows a similar pattern to the one I am deploying with JAX, but in this case I am using JAX both to define the system (using pure python functions) and solve the system (using a JIT-compiled optimization step). The key difference is that the CasADi / ACADOS workflow targets execution on embedded devices. It is probably much more efficient, but harder to update with new models (requiring a more explicit compilation step).

MPCs are typically used for short-order optimizations, operating over time horizons around one to five seconds. They usually run somewhere between *50Hz* and *500Hz* since the compute required to solve them is intense. For longer horizons of control, the current best practice uses simulated systems to train *policy controllers* [32] that can issue higher-level control commands. Policy controllers are normally coupled with lower level, faster feedback controllers and classical controls components like kalman filters, [33] is a particularely clear example of how all of these components come together to produce performant controllers. Policy controllers are often trained without gradients because it is hard to differentiate across long time spans, but recent work deploys differentiable simulation to overcome this issue [34].

All told, optimization based control is a vast, complex discipline. I have not mastered any of it, but the available tools have become sophisticated enough that I can generate a solver using my own

models and deploy it in practice. Above all, this is a testament to the community of researchers and engineers working together to advance the practice, who are committed to sharing reproduceable, useful tools with one another.

3.5 Modelling FFF Polymer Flows

Model based optimizations require models. Luckily there is a lot of interest in FFF printing in the literature, and so the applied physics are quite thoroughly understood [35], [17]. I also provide a simple drawing of the main phenomena in Figure 6, and an overview of the physics involved in Section 3.3.a, with some more details in #sec-hidden-optimization.

Of particular relevance to this work is a line of research begun by TJ Coogan and David Kazmer, who developed an instrumented extruder similar to ours in [36]. Filippos Tourlomousis takes credit for bringing this idea to the CBA and implementing the first version of the hardware, and I extend a design pattern developed by an FFF youtuber to implement the filament sensor [37] (adding a feedrate encoder, and calibrating width measurements). Kazmer continues work to characterize the *dynamic* behaviour of filament during extrusion in [38], which helped me to develop the dynamics model I use in the current iteration of the FFF flow model. These works were also fundamental to my work in [39] (rendered Figure 29 and Figure 30), where we combined online model-building with parameter selection in a reduced space to print with unkown materials.

A selection of the literature disccusses performance limits to FFF printing, notably [21] discusses absolute limits to print speeds based on nozzle thermodynamics and [40] models flowrates through a hotend as a function of nozzle temperature. The focus of the work in this thesis is not necessarily to push those limits, but to develop controllers that more routinely run at or near those limits (pushing performance of existing systems).

Another selection studies the relationship between slicer settings and print performance (strength, precision) [41]–[45], but these studies all operate in an outer loop around the state of the art workflow - i.e. they optimize *slicer* parameters, whereas (as I discussed in Section 3.3.a) these are a somewhat lossy abstraction over the *as processed* parameters (due to firmwares' speed scaling).

Researchers are also interested in modelling inter-layer weld strength [18], [46], which is a function of the weld's thermal history (more time above the plastic's glass transition temperature equates to more diffusion of polymer chains across the weld), and the inter-layer pressure exerted by the pressure during extrusion. It should be possible using the framework developed in this thesis to describe those physics as an optimization target, for example adding to the solver's cost function a reward for maximizing weld temperature - but doing so would probably involve much more complex simulation of the print *as a whole,* whereas my approach only considers shorter time spans.

Despite all of this research, there is limited literature that combines rheological modeling with online controllers and motion systems - except for [47]. They make significant progress in adapting online control to optimize extrusion during machine operation, using flow models fit from line-laser scans in conjunction with servo control of the extruder motor in a hybrid system, eliminating many extrusion defects like under- or over-extrusion. To compare with work in this thesis,

our approach combines similar models for extrusion with more advanced models of machine motion and kinematics, which should yield overall improvements to print speed - taking the machine system *as a whole,* whereas their work focuses mainly on the optimization of extruder commands. Ours also integrates the additional loadcell and filament sensors, which allows us to bootstrap models on previously unseen filaments.

I would also like to note that the proliferation of research in this domain is probably due to the proliferation of accessible and often open-source designs and documentation of FFF printers, and an active community of FFF enthusiasts online - this goes to show that efforts in building extensible systems architecture may enable similar proliferations of research for other digital fab processes.

3.6 Modelling CNC Milling

Machining models will become important when I begin work in earnest on Section 5.3. Models that predict cutting loads will be of particular interest, [48] and [49] (a CBA undergraduate researcher who I mentored three years ago) are two theses that provide a good overview of simple models for the same. Both outline relatively simple models that output radial and tangential cutting forces as a function of straightforward geometric parameters (cutter size, cut depth, spindle rpm and linear feedrate), with four coefficients to fit for cutting force and edge force coefficients, each with tangential and radial coefficient respectively. Sharma shows that coefficients can be estimated using data generated on-machine, using measured spindle power and force exerted on the workpiece (a loadcell). This is a promising indication that I will be able to replicate his work using my frameworks. Dunwoody follows a similar path and adds resonance measurements: he uses a solenoid hammer to generate a step response of the tool, measuring the tools 'ring' using an inductive probe as a displacement sensor. Using this system, he generates a stability plot across spindle RPMs and depths of cut.

Resonance measurements are likely to be important for performant milling, as excitation at these frequencies generate resonant chatter [50]. In [51], the authors develop a system that communicates with a Heidenhein machine controller at 10kHz to detect chatter in real-time using motor currents, and [52] develop a model-based adaptive controller to mitigate chatter and [53] implements a "machining digital twin" to optimize feedrates of a milling center in real-time to predict and automatically prevent chatter. Finally, for a broader overview of machining dynamics, we also have [54].

All told, there is great background in the literature to show that the proposed contribution in Section 5.3 is viable. Resonances of the machine seem easy enough to measure, and most resonances are under 1kHz, meaning that I should be able to sample fast enough to generate these data. However, I will have to manage frequency domain systems representations alongside the time domain representations that are dominant in path planning. I suspect that I can articulate these in the cost function of my optimizers. The contribution in this thesis will be to implement these systems in a manner that requires minimal intervention from the user, and that works to automatically bootstrap models when new materials or tools are used. It will also be a strong demonstration of the flexibility of the systems architectures developed in this thesis, to deploy

on two very different machining processes, capturing and extending state-of-the-art practice in each case.

3.7 Systems Architecture Background

The work in this thesis is enabled by a flexible machine control architecture that combines modular hardware with software. This model was originally formalized by [55] and [56] at the CBA as *Object Oriented Hardware.* The CBA also has a history of developing small networks for inter-device internetworking [57] and building modular robotics [58], [59].

Work on modular physical computing is active in the HCI community [60], [61] and has a long history in STEM education [62], [63]. PyBricks [64] is an active project that deploys python interfaces on Lego modules. I made one contribution in this domain with Modular-Things [65] alongside Quentin Bolsee and Leo McElroy, where we developed a new set of hardware modules and tested their use in a machine building session at MIT. That work contained early prototypes of OSAP (Section 5.4) and MAXL (Section 5.1); I also formalized some of MAXL's design patterns in [66], adding *time-sychronized distributed trajectories* as a design pattern for organizing motion across modules.

Efforts are also ongoing to improve interfaces for digital fabrication machines, [67] and [68] develop interactive machine interfaces in Grasshopper using a python script as an intermediary to send GCodes to an off-the-shelf machine controller. In [69], computational notebooks are used as an interface for machine workflows: their system also implements an intermediary software object that communicates with off-the-shelf controllers using GCode, but presents a more useful API to the notebook.

The Jubilee project [70], [71] is a machine platform that implements a modular tool-changer, and has been successfully deployed by researchers to automate duckweed studies (a popular model organism) [4] and to study nanoparticles [5]. Jubilee *also* uses an intermediary python object to interface with an off-the-shelf GCode controller, and shows the value of integrating motion systems with application-layer scripting languages.

Work in this thesis aims to extend these efforts by providing lower level motion control interfaces in the same scripting languages, reducing distributed state in the overall control architecture and making systems easier to debug and develop; consolidating configuration state was a topic discussed during and NSF sponsored workshop that I attended on open source lab automation tools [72] where we used Jubilee machines. OSAP also extends other modular physical computing frameworks by enabling the use of a multitude of link-layers, whereas i.e. JacDac and Gestalt are limited to custom embedded busses.

Object Oriented Hardware for machine control presents many practical challenges: control over networks introduces timing overheads not present in digital controllers that add constraints to control algorithms [73]–[75]. Some of these challenges can be overcome by distributing models throughout a system, trading computation for bandwidth [76] - MAXL (Section 5.1) takes some inspiration from this approach, allowing motors to incorporate simple local controllers that can take-over in the event of network failures.

Developing networks for real-time systems is itself a challenge, luckily there is well established practice in this domain. In particular, I borrow a scheduling pattern from [77] and clock synchronization patterns from Network Time Protocol [78] and high-performance counterpart [79]. I have also studied simpler approaches from explicitly real-time domain [80].

4 Research Questions and Evaluations

4.1 Can We Replace Parameter Tuning with Model Building?

Advances in differentiable simulation and solvers have made it possible to refactor machine controllers as online optimization routines, but it is still unclear how to put all of the constituent pieces together. This raises a number of questions.

Firstly, we need to know **how to architect our system so that we can connect enough compute to run the optimization in real-time with embedded systems that realize those optimized outputs** - OSAP (Section 5.4) is one half of my contribution in this regard, and MAXL's (Section 5.1) partitioning (using basis splines to transmit discrete parts of a path plan at a fixed time interval) is the other. We also need to learn **what kind of models** are appropriate to use in this context, balancing fidelity with simplicity. We also need to learn **how to develop and fit those models,** in this work I propose adding instrumentation to machines directly, and building bootstrap models using canned routines, but then improving them during machine operation. I discuss resolutions to these questions for FFF in in Section 5.2.a, Section 5.2.b and Section 5.2.c, though I am not yet sure exactly how to formulate the models that will be required for the CNC Router proposed in Section 5.3.

A simple metric for evaluation that I will discuss in those sections is to measure *how many input parameters are required* in order for the processes to work (comparing to state of the art CAM), i.e. how much input is needed from a machine user in order to get *up and running* with a machine and material.

The proposed optimizer also has clear performance metrics to measure: it should help machines run closer to their maximum limits, minimizing time to run jobs and maximizing precision.

4.2 Can We Make Motion Control Modular and "Easy?"

I am also interested in developing this control approach in such a way that it is approachable for machine *builders* to deploy on new systems. This raises some structural questions about **how we should represent controllers to machine developers** - in particular, how they reconcile their mental model of their machine's kinematics with the controller's own model. **Can we generate these representations automatically, or at least tune them automatically? How do machine builders debug their controllers, when misalignments are present, and how do they tune and evaluate their machines' performance?** I propose evaluating MAXL in this regard by running a workshop that I describe in Section 5.1.b.

I also want to see **how modular** we can make machine control. Success here would mean that we could implement *almost* any kinematic system (or end-effector) using a small set of re-useable

components. To see whether or not MAXL and OSAP succeed, I can track the breadth of machine systems deployed using the systems.

4.3 Can we Automatically Generate Control Interfaces from Hardware?

Machine building also involves the generation and use of software interfaces for hardware devices. Making and maintaining these interfaces can be a pain in the state of the art, as I discuss in Section 5.4.b. In developing OSAP's network- and presentation-layer codes, I have been interested to see **to what extent can we automatically generate these intermediary representations of modular hardware.** In this work, I contribute *automatic RPC* tools for compile-time generation of interfaces to generic functions, and I am proposing to finish a long standing interface for the development and debugging of systems-level interfaces, as I discuss in Section 5.4.d.

5 Expected Results and Contributions

Each of these contributions are built on top of OSAP (covered in Section 5.4). Here I am listing them in the order of relevance to the research questions posed in the prior section.

5.1 MAXL: Model-Based, Modular Acceleration Control, Coordination and Execution Library

MAXL is a machine control framework that I have authored over the course of my time at MIT. It runs on top of the systems glue I have developed alongside it (OSAP, see Section 5.4) and consists of a few key components:

- Firmware libraries that allow embedded device authors to expose functionality to motion planners.
- Software libraries that allow machine builders to author kinematic models of their machines.
- Software libraries that allow machine builders to author optimization-based solvers for their machines.

MAXL's contribution is to provide a generalizeable framework to control *almost any machine* using a set of re-useable software modules. It presents motion control systems as assemblies of functional blocks, where the aim is to allow machine developers to build controls systems from modules that are each easy to comprehend, but whose various combinations can span a large space of possible machine designs - as well as provide a proving ground for other systems described earlier in this proposal, like automatically calibrated kinematic arrangements, or model-based optimizations.

With MAXL, I contribute a system that answers many of the questions posed in the previous section. It uses just-in-time compiling of JAX codes to run an online simulation of a given machine. The simulation (generated using a receding horizon of control outputs) is differentiable with respect to a cost function that defines the optimization. Any system that can be simulated using an integrator written in python can be simulated and optimized, subject to compute performance constraints.

In Figure 15 and Figure 16 below, I show some early results of the solver operating on a motion system alone: because the MAXL solver can use a more nuanced motor model than off-the-shelf

trapezoidal solvers, it manages a 15% increase in overall speed and produces a smoother velocity plot.

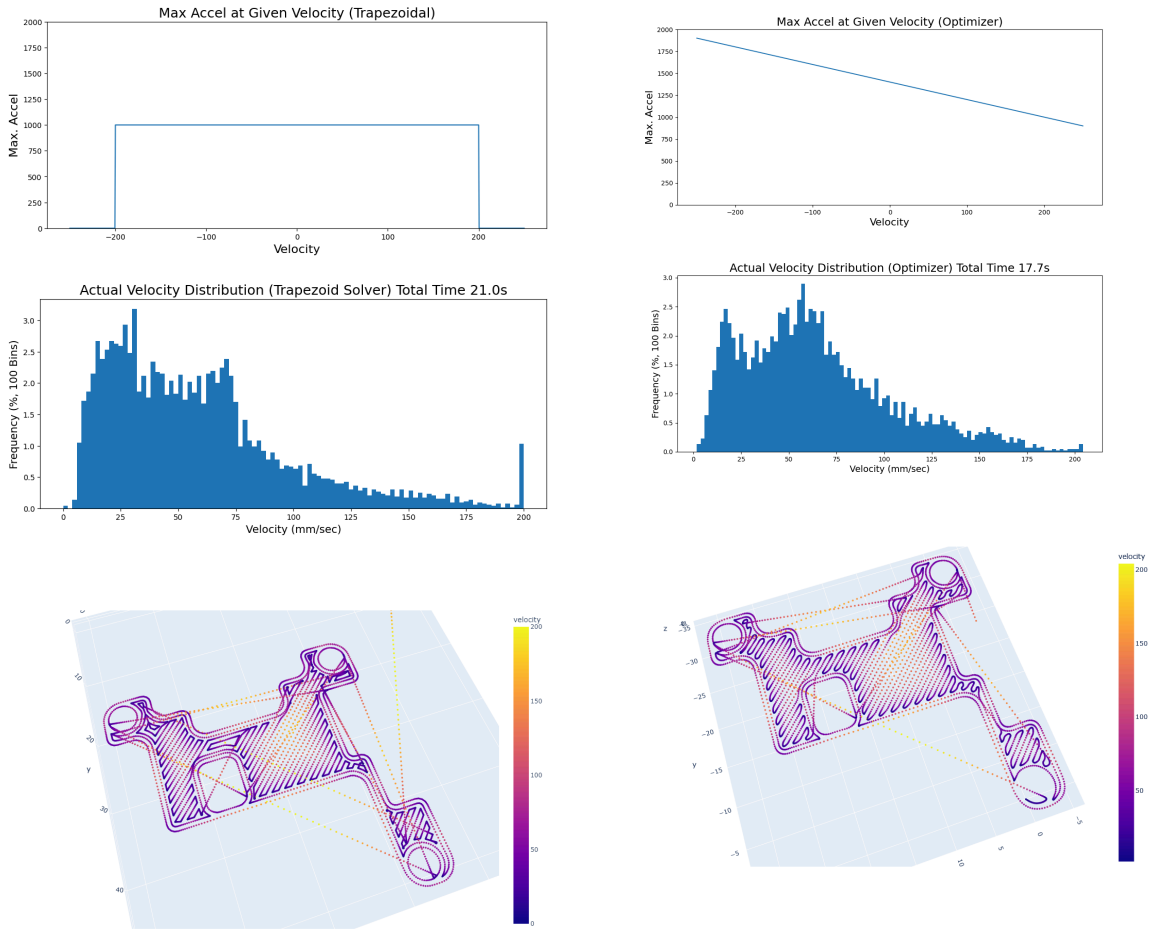


Figure 15: Here I compare a classical solver (using trapezoids, left) with MAXL's model based solver (right). The top of both columns show the equivalent motor models being used (these effectively represent the solvers' understanding of the motors' torque curves). In the middle are velocity histograms of each solvers' outputs, showing that MAXl's outputs effectively move the peak of the velocity distribution to the right (big number good, small number bad). The bottom of each column renders the actual trajectory with a heatmap of resulting velocities at each position on the path.

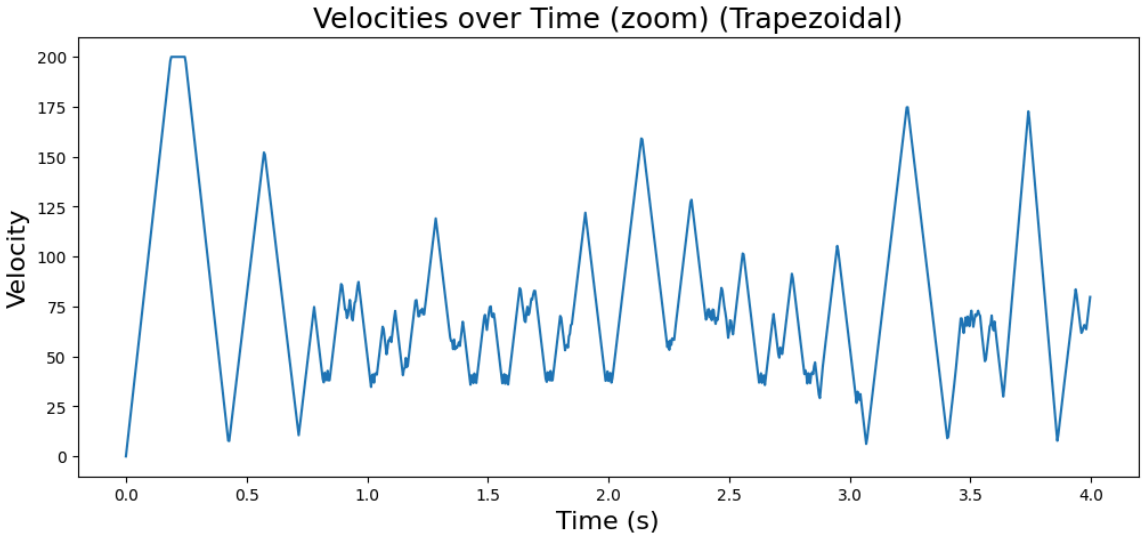


Figure 17: traps

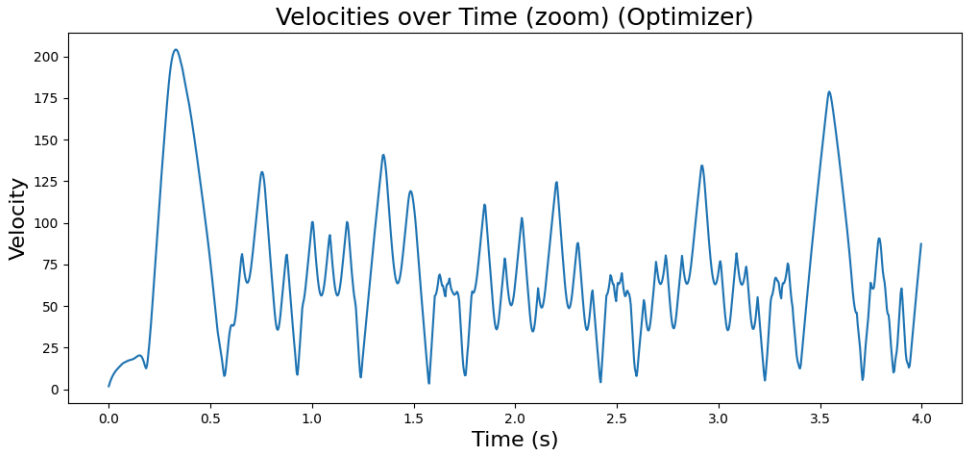


Figure 18: optim

Figure 16: Here I show time-series plots of velocity for the trapezoidal (top) and MAXL (bottom) solvers over a subset of the total interval. Velocity units are mm/sec. MAXL produces *smoother* velocity plots, and eliminates instantaneous changes in acceleration. This should help to reduce excitiation of machines' resonant modes.

To span the partition between hardware and software, MAXL breaks motion into fixed-interval components of basis splines. Devices are all time-synchronized using OSAP, which also helps to manage systems-level configuration. This partition is what allows us to run the online optimizer in a high-power compute environment (i.e. a laptop), alongside a flexible set of embedded devices that operate motor controllers (and other output devices) and sensors.

I provided an architectural overview of the scheme in Figure 1 and I go into more detail of these models (and the solver) in the section below on the Rheo-Printer (Section 5.2). In particular, Figure 31 shows a screenshot of the solver's visualizer.

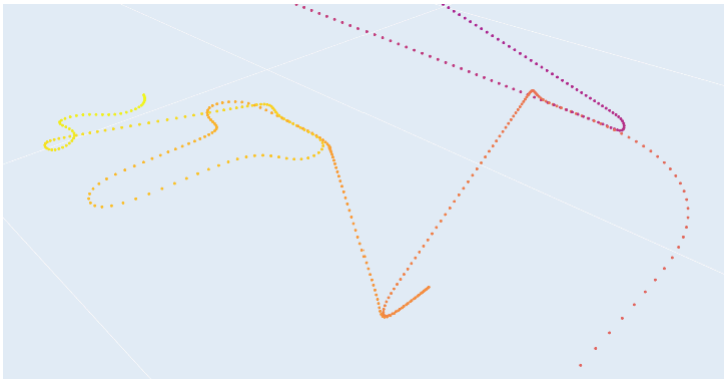


Figure 19: MAXL uses basis spline interpolation as a generic intermediate representation for motion. In this figure, I render a subset of spline control points that were generated by MAXL to stream to motors during a print job.

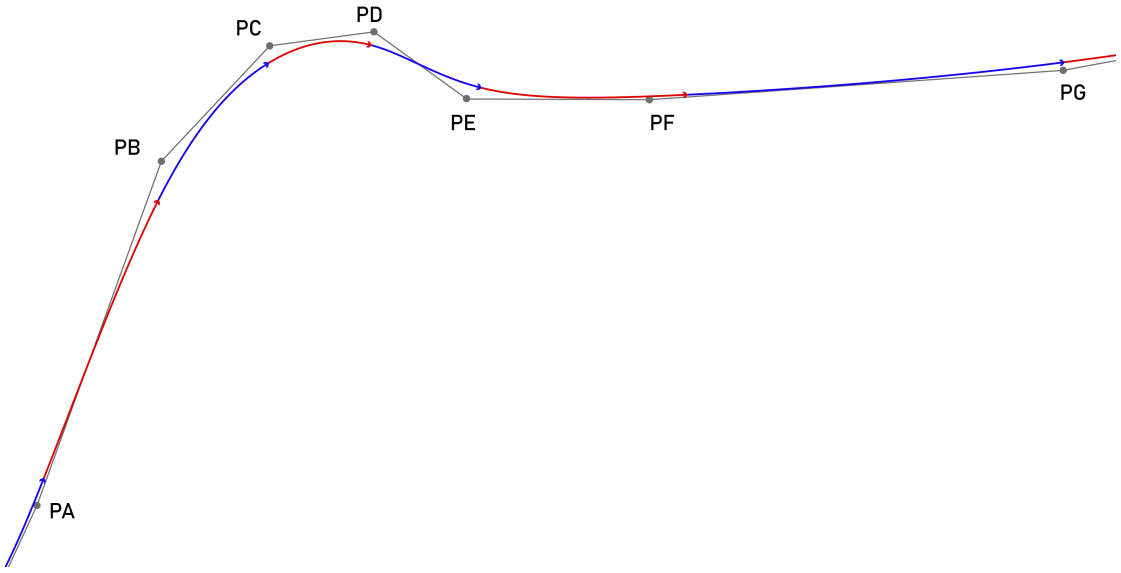


Figure 20: A 2D subsection of a basis spline, showing a stream of control points (PA ... PG) and the resulting curve.

$$P(t) = [1 \;\; t \;\; t^2 \;\; t^3]\frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}\begin{bmatrix} P0 \\ P1 \\ P2 \\ P3 \end{bmatrix} \tag{1}$$

Equation 1 is the cubic basis-spline form that MAXL uses. t spans a fixed interval, and the interval is set at some integer value of microseconds that is a power of two, between 256us and 16384us. Using these intervals means that the spline can be evaluated using fixed point arithmetic in embedded devices. Basis splines have the helpful property that we can always add new points to the end of a stream, meaning that at each interval we only need to stream one new position (whereas i.e. a linear segment of similar length would require much more information). This works well for motion because the splines' own properties are well matched to moving systems [81]. Fixed-interval tends to work because detail in motion tends to correlate to slower velocities (and so we end up packing more points in intricate parts of the path). While using these splines has been hugely productive for rapid machine development (their deployment means that I don't have to modify motor firmwares in order to develop new motion schemes or kinematic models, etc). Splines *are a lossy abstraction.* They are not a direct interpolation, and at long time intervals they can result in motion that deviates from planned positions. I am overdue to carefully quantify these losses and their tradeoffs, and I should do so as I finish the PhD.

Where full-blown simulations are not required, MAXL also includes a classical trapezoid-based motion solver that interfaces to the same set of modular hardware and firmware.

5.1.a Generation of Physical Machine Representations in MAXL

In the section below on OSAP (Section 5.4), I discuss how I automatically generate software interfaces to machine systems. I use MAXL to attach physical meaning to these representations.

In the simplest sense, MAXL's interfaces need to accurately reflect what will happen in the real world when we move the motors on the `x` or `y` axes on our machines. This seems like a simple problem, but it is pervasively difficult to debug, and leads to real trouble for CAM companies like Autodesk, who have to interface to thousands of different machine vendors.

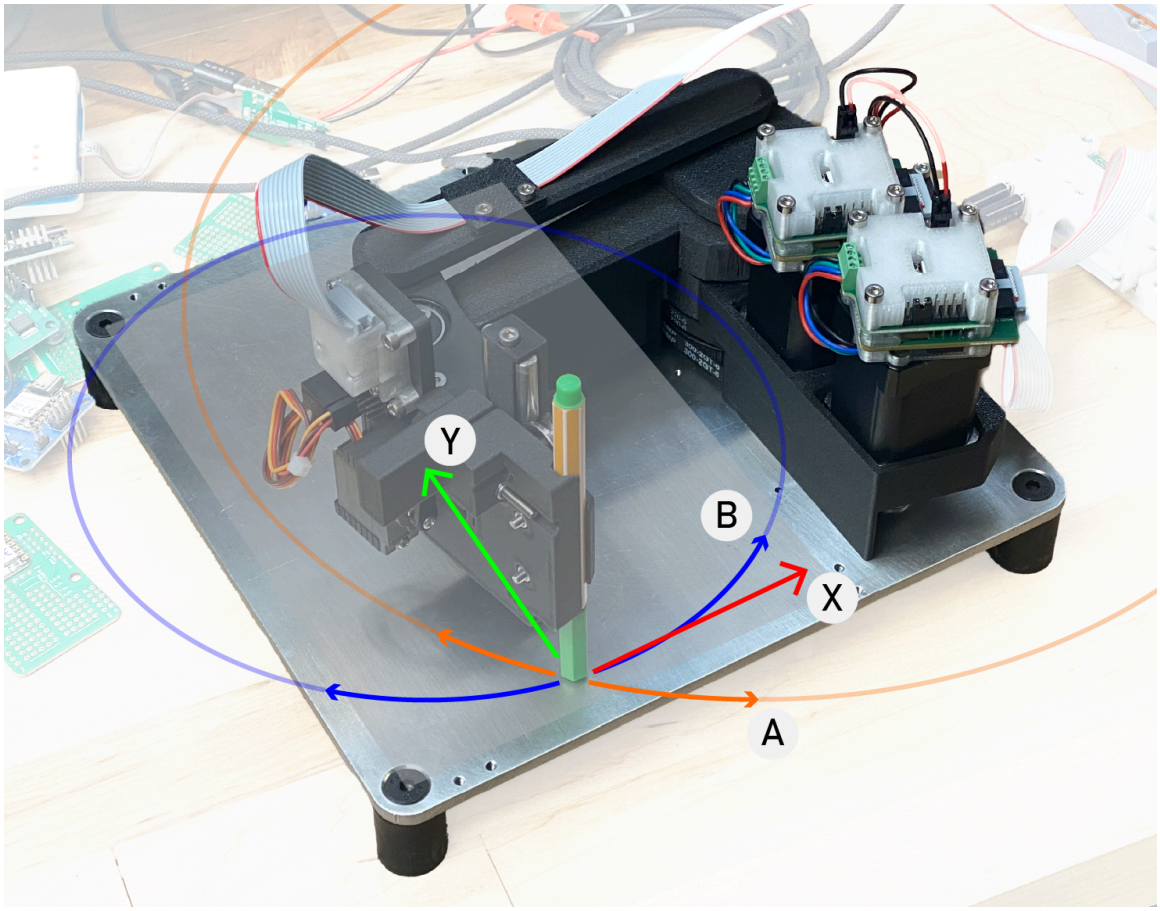


Figure 21: This small drawing machine has some nonlinear kinematics - the mechanism is one common to robot dogs: we have two rotary actuators (A and B) linked in an arm, each controlled by a motor that is rigidly mounted to the chassis. Kinematic arrangements like this can have serious advantages: they reduce total moving mass, and they package well (keeping the robot small while its work volume remains large). They can also be difficult to control, since mappings between actuator coordinates (in rotations) and work coordinates can be complex; sometimes there are not even closed-form solutions. In this thesis, I want to see how we can expose these complexities of the machine controller to machine builders in a *composable* manner, to help them more successfully build, tune and debug their kinematics.

Depending on the machine, this step alone (mapping from software-controlled actuators to real-world space) can involve some tricky kinematic calculations, and even in simple cases (where i.e. one motor moves one axis in a straght line), often warrants some calibration and correction.

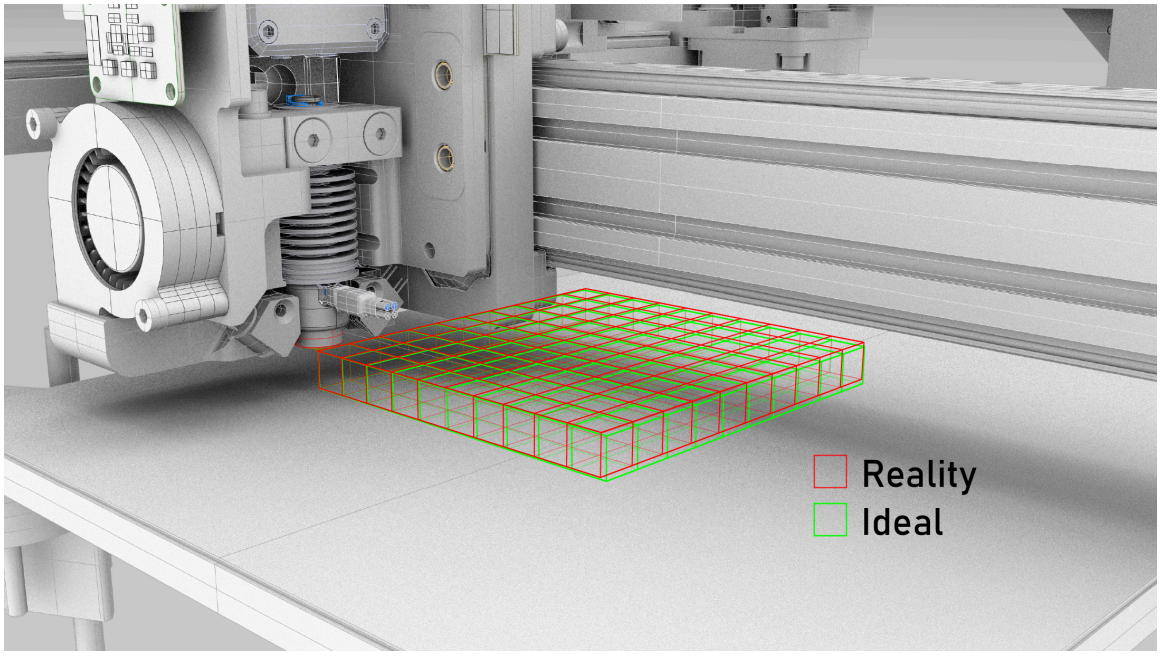


Figure 22: Machines never go together *exactly* as their designers intend. XY stages are never exactly perpendicular to one another, beds are never perfectly level with respect to XY travels, etc. We can correct for these errors using the machine's controller (as has become common practice), but need to develop integrated metrology on each machine in order to do so. For FDM printers, the z-axis is particularely sensitive to errors and so it is normally calibrated using a touch probe (I use the loadcell already integrated for rheology). There are not *general* solutions for this problem, but it is the kind of task that a controller which is well integrated with other optimization tools (i.e. in python) is well suited to solve.

Most machine builders already have a good sense of how their machine will work: they can develop a set of kinematic equations that describe their machine's motion. However, *fine tuning* these models is more difficult, and we normally use a combination of metrology and model regression to do so.

While it is tempting to explore automated methods for both sides of this problem, in this thesis I want to focus on the first step; how should we expose the kinematics module of our controls architecture such that machine builders can easily test and verify their models against the real world? The important thing to evaluate here would be *machine builders' experience* of getting a machine *up and running* using our controllers. An ideal system would be comprehendible, consistent, and interrogateable. It is also typical that machine designers will have relatively complete CAD designs of their hardware - our system should take advantage of that to build digital twins that enable visual debugging of kinematic descriptions.

A well-matched kinematic description of a machine also has meaningful performance advantages, since it would enable us to develop more complete digital twins of the dynamical system; i.e. allowing for controllers to track the complete energy and inertial states of each component.

5.1.b The Plotter Comp: Evaluating a Plenitude of Machine Kinematics

To evaluate MAXL, I want to co-host a Plotter Competition at MIT this January (2025). The format will be based on a project I co-hosted with Ilan Moyer and Leo McElroy where six participants each developed a pen-plotting machine over the course of an intensive two-day workshop, using an early version of MAXL.

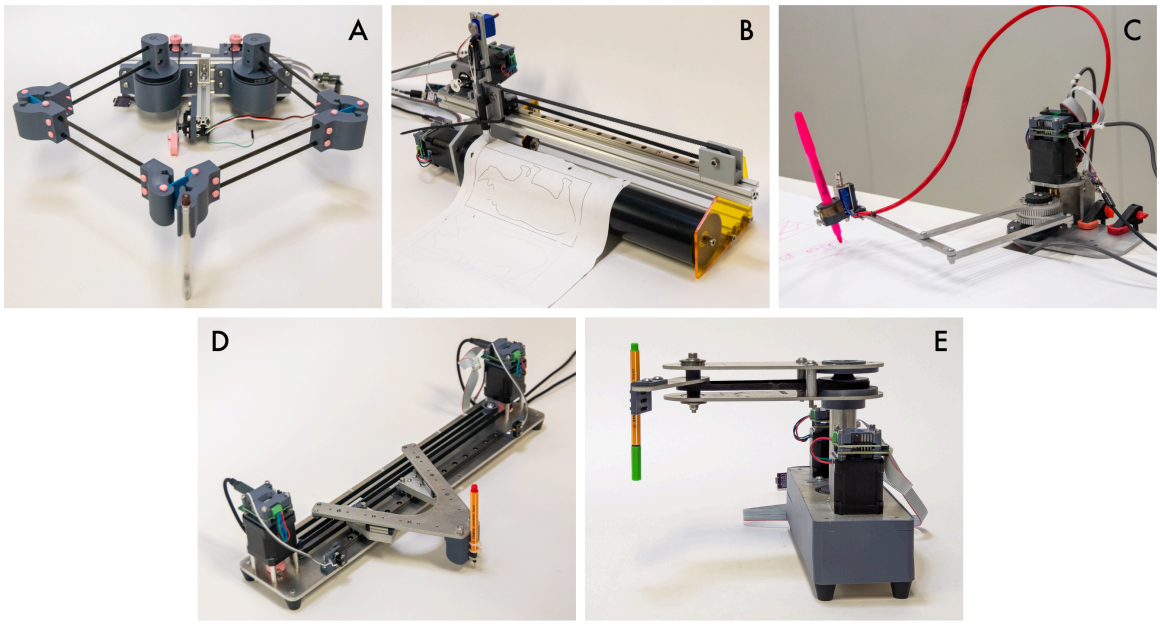


Figure 23: A series of plotters developed by participants in a workshop that I co-hosted at the CBA with Ilan Moyer and Leo McElroy. Participants were each experienced machine builders, but not experienced controls engineers. We were able to use MAXl to develop working kinematic models for each of these machines.

During that workshop, I personally helped each participant to control their machine using the MAXL and OSAP libraries I have authored to date. In the next workshop, I want to see if machine builders themselves can successfully use MAXL to develop a control system for their machines. To do this, I will document the MAXL python library and two working machine examples, and provide a set of ready-to-run (but otherwise unconfigured) controllers. I will interview machine builders before, during and after the workshop to garner their feedback on MAXL's implementation, and their experience using it.

Evaluation would then be based on machine builders' success in controlling their machines - first, whether they are able to successfully develop and debug their kinematic models using MAXL, then whether they are able to tune and improve those models using feedback (or heuristics). MAXL also provides interfaces tools that allow machine-builders to inspect their systems, using velocity plots, histograms, and other time-series data directly from the controller - I will be curious to see if anyone can use these successfully to improve their machine.

Evaluation would also be based on the heterogeneity of machines developed during the workshop, and other quantitiative metrics like controller performance (which depends also on networking performance via OSAP), and reliability.

I have some stretch goals for the workshop, listed below.

(1) The development of an integration of machine builders' CAD models with MAXL to complete it's internal model of machine motion with a virtual render of the machine (a *digital twin*) to aid in controller debugging. I prototyped one such system for the Rheo-Printer (see Figure 26), but only on an ad-hoc basis.

(2) For example, it should be possible to use time-series information from closed loop motor controllers to ascertain how well the kinematic models provided are fitting to reality. It would be interesting to learn how well this can work, and to what extent it could be a helpful tool for machine builders. For example, a bearing that is slightly misaligned during machine assembly may double or triple the friction in one of the machine's axes. This would show up in time-series data from motor controllers, and that could be fed back to designers. A robust digital twin would also be able to inform designers of where their performance bottlenecks are arising: highlighting an undersized motor for example.

(3) It should also be possible to deploy computer vision systems to automate kinematic systems description: build a machine, and then use video to determine the system of equations that govern its motion. While automatic, that approach may generate solutions that are computationally intense (making them unsuitable for use in a real-time controller) - that is another interesting question that I would *hope* to visit, but can't promise to resolve. At best, the controls architecture developed in this thesis will enable other researchers to work more confidently on those problems.

5.2 The Rheo-Printer

The Rheo-Printer is an FDM 3D printer that combines much of the work discussed in this thesis. It uses online instrumentation to build models of its own motion systems and process physics, and then deploys those models in a controller that combines motion optimizations with process optimizations in one online loop. There are a few distinct contributions / results of note in this project.

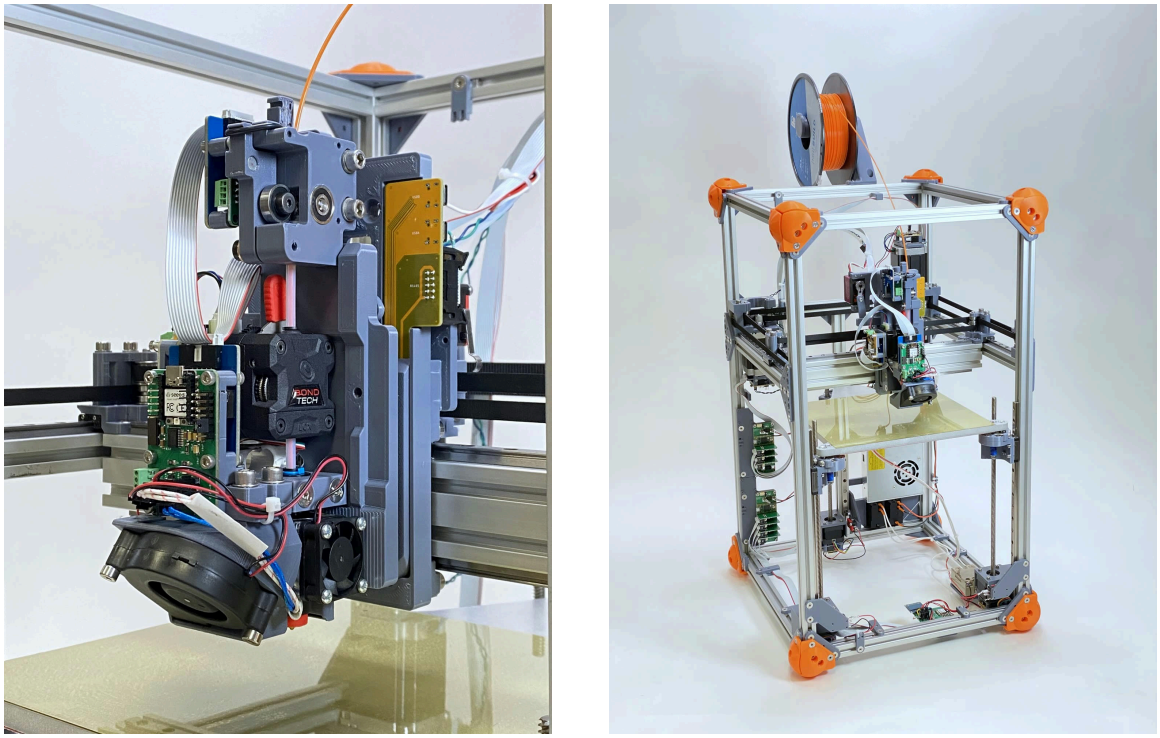


Figure 24: The Rheo-Printer is an FDM 3D Printer that deploys the controllers I have developed in this thesis, and includes instrumentation for flow- and motion model building.

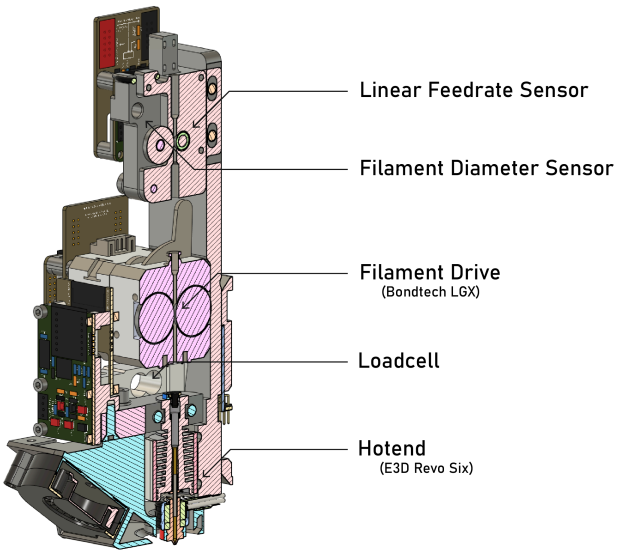


Figure 25: The printer's hotend features a loadcell between the hotend and extruder motor (which allows us to measure nozzle pressure), and a filament sensor that measures linear flowrate as well as filament thickness. The controls are assembled from modular electronics and connected using OSAP, controlled using MAXL.

5.2.a Deploying MAXL to Generate Time-Series Data

My control approach requires that machine and material models be fit to the particular machine they are deployed on, and this means that MAXL and OSAP need to enable us to run data collection routines. It also involves the development of those collection routines.

For this task, I lean on OSAP's time synchronization and generate time-stamped data streams from sensors embedded in the machine. These let me generate detailed time-series datasets that are reconciled to the machine's real motion trajectories. Those can then be used to fit models, using optimizers not dissimilar to the one which is used during online control (i.e. gradient-based search).

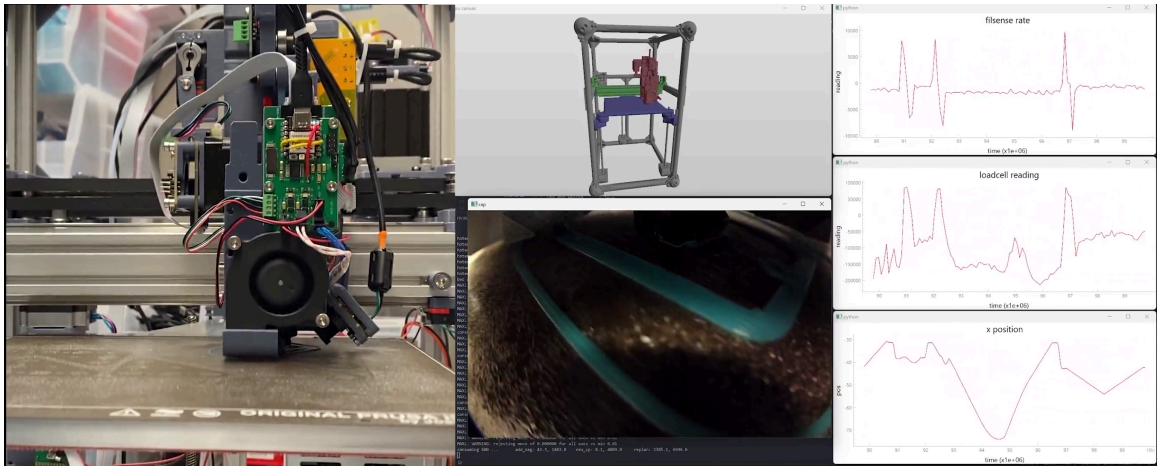


Figure 26: **Click here for a video.** Earlier in Figure NUM we looked at data collection for steady-state FDM model building, in this figure we see a screenshot from a data collection routine on the Rheo-Printer where dynamics data are being collected form the loadcell and filament sensor, time-synchronized to the motion controller's states. At the time of writing, we can collect dynamical flow data from the first layer only due to our relatively naive computer vision setup - a major part of the next step in this research arc is to outfit the printer such that we can collect high frequency data at each layer.

5.2.b Experimental Designs for FFF Model Generation

A question posed earlier had to do with **how to safely generate models of machine components** for use in an online optimization: i.e. running non-destructive tests that are explicative of where failure *would* occur, or that let us operate machines near their limits without exceeding those limits.

A trend emerging at this point in the research is to use simple models to 'bootstrap' a new machine or process, and then use time-series data generated from the operation of the machine using those models to fit or generate higher fidelity models.

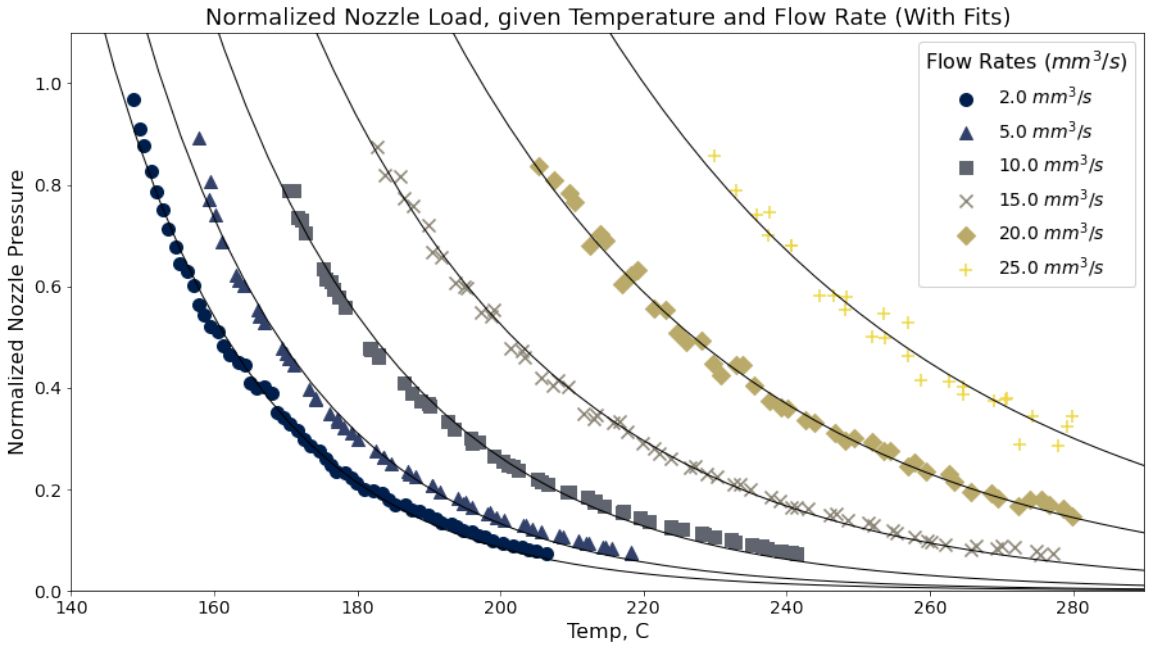


Figure 27: Test data from an experiment for steady-state flow data - which was fit to the model in FIGURE ABOVE. Developing this test routine was the key 'invention' for that initial piece of work on parameter reduction: the routine allowed us to effectively test the workable range of the hardware *without* exceeding that range (and i.e. breaking / clogging the nozzle or shredding filament).

5.2.c Model Selection for Online Optimization of the FFF Process

Related to overall compute performance, we need to learn what fidelity of model is required for our real-time optimization task. Candidates include classical models (that are hard to build but easier to fit, and faster to run), various flavours of neural network (which are easier to train but slower and less relate-able to underlying physics), and even particle simulations.

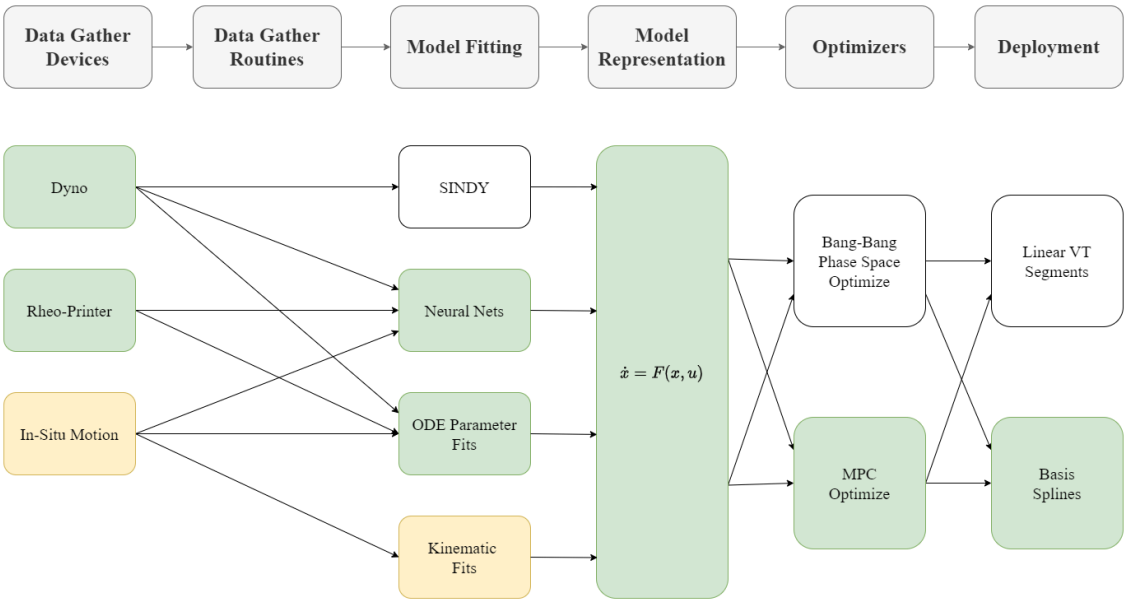


Figure 28: Here I show the littany of of model gathering and fitting techniques that I tested as I developed the Rheo-Printer, as well as a shared functional representation for the differentiable simulation (simple integrators) and optimizer. I also evaluated two solver architectures, and two different approaches for deploying synchronized motion segments.

Listing 6: At the time of writing, models are defined in my motion controllers by writing authoring integration functions. Provided that these are purely functional, they can be combined easily into the gradient-descent based optimizer that I have developed with JAX [24], a differential programming library. This integrator here describes how the Rheo-Printer's position changes over time, depending on commanded motor torques, it is 'half' of the system that we want to optimize in order to operate the machine.

```
def integrate_pos(torques, vel, pos, del_t):
    params_torques = jnp.array([2000, 1000])  # per-axis motor effort to torque
    params_frictional = jnp.array([2, 2])     # per-axis damping

    torques = jnp.clip(torques, -1, 1)
    torques = torques * params_torques

    acc = torques - params_frictional * vel
    vel = vel + acc * del_t
    pos = pos + vel * del_t

    return acc, vel, pos
```

Listing 7: This integrator comprises the 'other half' of the problem we want to optimize when we run the Rheo-Printer: it describes how flow out of the printer's nozzle evolves over time, given input torques at the extruder motor.

```
def integrate_flow(torque, inflow, pres, del_t):
    k_outflow = 2.85        # outflow = pres*k_outflow
    k_tq = 1                # torque scalar,
    k_pushback = 10         # relates pressure to torque
    k_fric = 0.2            # damping

    torque = np.clip(torque, -1, 1)

    # calculate force at top (inflow) and integrate for new inflow
     force_in = torque * k_tq - inflow * k_fric - np.clip(pres, 0, np.inf) *
k_pushback
    inflow = inflow + force_in * del_t

   # outflow is related to previous pressure, simple proportional model for now
    outflow = pres * k_outflow

    # pressure rises w/ each tick of inflow, drops w/ outflow,
    pres = pres + inflow * del_t - outflow * del_t

    # that's all for now?
    return outflow, inflow, pres
```

The models that I am using at the moment (both of which are written down above, in the form in which they are integrated into the optimizer) are relatively simple: the motion model is essentially just a damped inertial system, and the extruder model is isothermal and uses a linear relationship between pressure and flow, even though we know from steady state data that this is not the case (see the figure below). These simple models are enough to capture the most critical dynamics, and even still are a step above the heuristics that off-the-shelf machine controllers use. The next step of my research plan involves outfitting the 3D Printer with data acquisition tools that will let us collect thousands of times more data (at hundreds of times higher frequency), so that we can fit more advanced models (and build them incrementally).

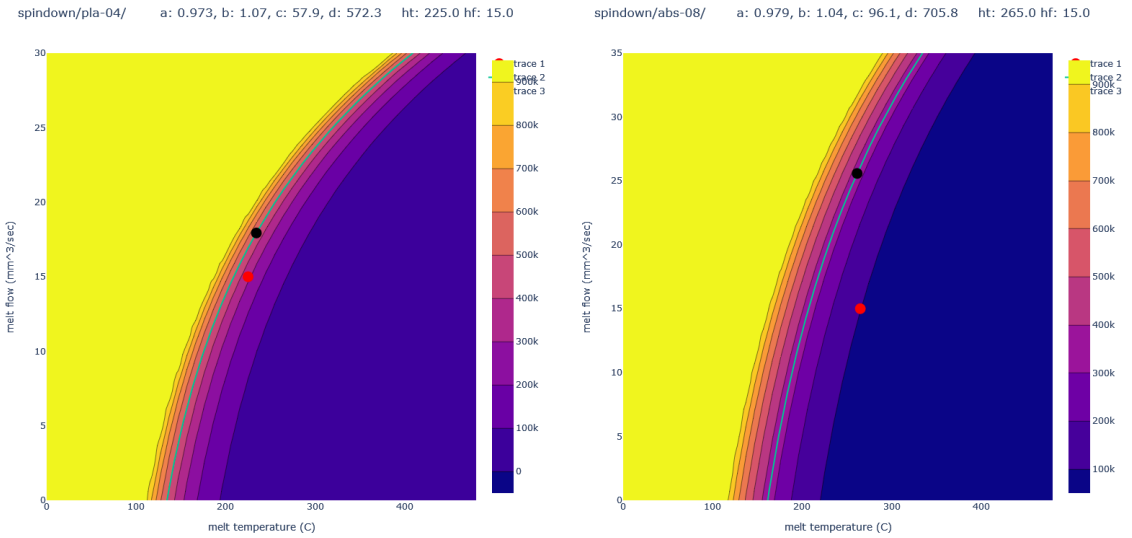


Figure 29: I developed this steady-state model for nozzle flow vs. nozzle pressure (across a range of temperatures) using the Rheo-Printer's instrumentation. It captures some complex rheology, but only for steady-state flows (which are rare during printing). In this figure, we see the model fit for PLA using a 0.4mm nozzle (at left) and a fit for ABS using a 0.8mm nozzle (at right). Red dots are where off-the-shelf parameters (heuristics) suggest the material should be used, black marks are where our parameter-picking algorithm suggests they should be used (ours normally improves flow).

$$P = \left(-c^{Q+d} + 1\right)^{T+eQ^2+f} \tag{2}$$

Equation 2 is the model (Figure 29) that I developed to fit steady-state flow data (Figure 27) across an FFF extruder's entire operating range. Here, P is a prediction of nozzle pressure given temperature T and flowrate Q. c, d, e and f are parameters that were fit using a scipy [82] optimizer. At the time of writing, my dynamics model for extrusion (Listing 7) ignores this complexity, but future work should merge models like this (which describes flow, given pressures) with the dynamics model (which describes pressure generation as a function of extruder torques and filament compression). I also experimented with using neural networks to fit time-series extrusion data, but found it difficult to generate a robust model with limited data.

Also of curiousity is whether these models could be transferrable across materials or machines, or even processes. For example, it is likely that motion control models can be very simple and easily ported from machine to machine, but process models may need to be re-trained (or fit) for any new machine, material, or even for small changes of a machine (such as a nozzle diameter change or the ilk).

5.2.d Parameter Reduction via Online Optimization across Flow and Motion Models

This is a core contribution: I show that we can control the printer using a novel scheme that replaces heuristics for flow and for motion with models for each, that are simultaneously optimized

using an online, receeding horizon approach. This vastly reduces the number of input parameters required in order to print a part, because it replaces parameters with models across both aspects of the problem (material / process dynamics and motion dynamics).

Figure 30: In earlier work, I showed that even steady-state models (see Figure 29) can be used to reduce FFF printer parameter space; these benchy's[6] [83].

[6]The "benchy" boat has become the de-facto test print in the FFF development community, probably because it contains a swath of difficult to print features (like overhangs and bridges), is small enough to print quickly, and because it is cute.

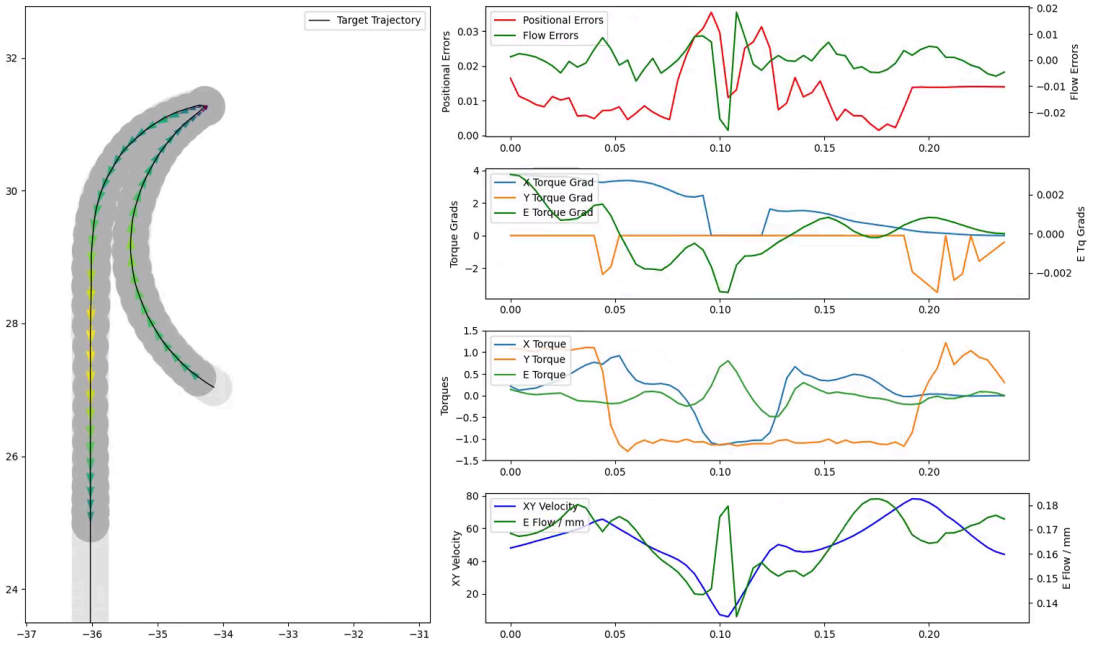


Figure 31: This is a plot generated from the online optimizer I have written that solves for the Rheo-Printer's motion system and polymer flow simultaneously, using models as shown in Listing 6 and Listing 7. At left, we see the simulated trajectory, with arrows color-coded to resulting velocity at each time step and with target extrusion (light grey) with resulting extrusion (dark grey). This simulation is differentiable with respect to a cost function that describes *optimal* control outputs as those which maximize printer velocity while minimizing errors. At right, various output plots from the same simulation; errors (top), solver gradients, solver torques (middle) and velocities (bottom). The solve is done over a quarter-second receding horizon window, with one time-step's control outputs (4 milliseconds long) issued each cycle.

The approach shares much in common with Model Predictive Control, which is a common approach to control for dynamic robots [27] - here we show that *when we use models that are developed in situ,* the same pattern can be applied to machine processes.

I will evaluate the success of this contribution by comparing the printer's dimensional performance, as well as its speed, to a converntional workflow and machine. I will also compare the amount of user input required for successful prints using this method vs. using traditional workflows.

5.2.e Finishing the Rheo-Printer

My work so far on the printer is promising: I have generated models extrusion using data generated in situ and merged them with motion models to generate an MPC-based optimizer that is performant to deploy in real-time. To finish the project, I need to merge that controller with the hardware and test its real-world performance. I also want to improve the fidelity of the models used for extrusion by generating new flow data at each successsive layer. To do this, I will complete a hardware update (Figure 32) that uses a line laser and downwards-facing camera to

generate a height map of each layer (similar to a system deployed by [47]). Since I have access to time-and-space synchronized motion information, I should be able to reconcile these scans with historical machine states and generate high-frequency time-series data.

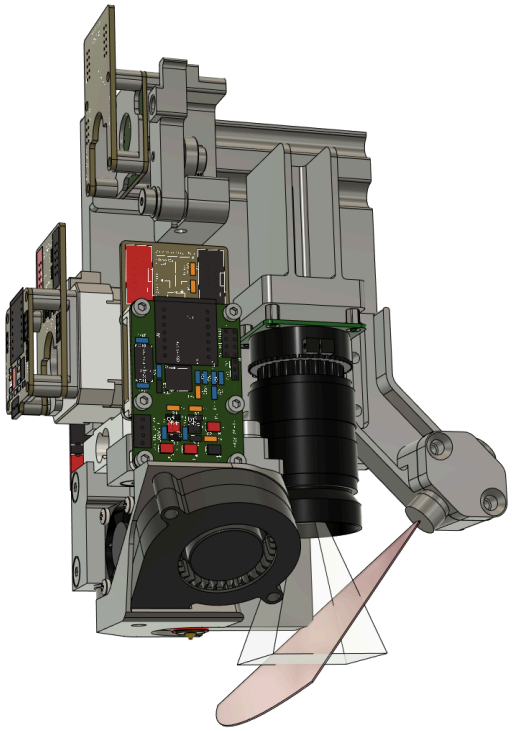


Figure 32: A draft of an updated design for the Rheo-Printer hotend that incorporates a line laser and downward facing camera to generate higher fidelity flow data after each print layer.

5.3 A Smart CNC Router

Besides FFF, '2.5D' CNC Routing (to produce mostly-flat parts) is probably the second most-common form of Digital Fab: with it, you can make furniture, project enclosures, mechanisms, *other machines,* quadcopter frames, etc.

Routing poses many of the same challenges as FFF printing: users need to hand-tune parameters for each new material and tool that they want to use. To demonstrate the flexibility of the controller architectures presented in this thesis, I want to extend MAXL to perform online optimization of process and motion models for CNC Milling. This will involve building models of cutting force, fitting them to time-series data, and bringing them online with motion models already developed in the thesis.

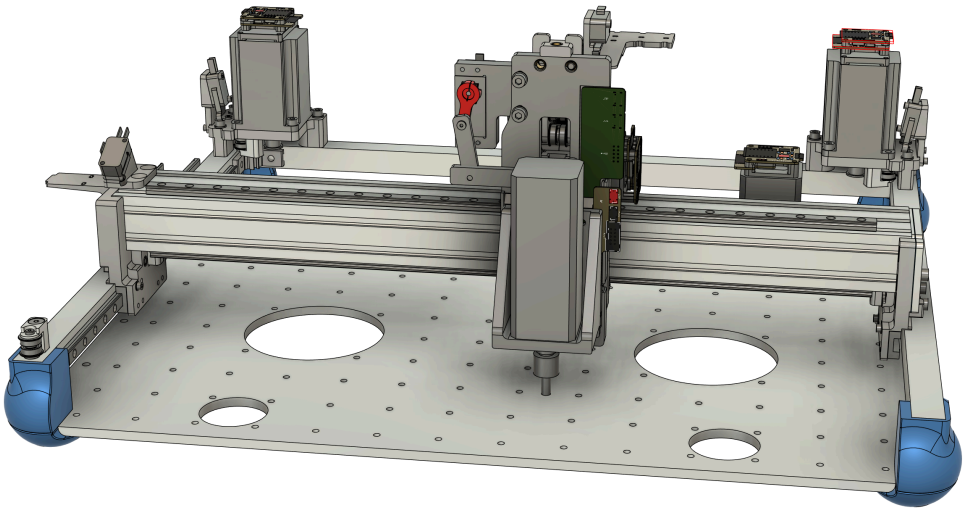


Figure 33: A completed design of a simple 2.5D milling machine. The hardware uses many of the same design patterns as those present in the Rheo-Printer, but is laid out for processing thin materials.

Whereas other contributions mentioned in this proposal are mostly complete, this project is mostly new. I have built milling machines in the past, and I do have a complete design for the system, but it will require that I develop one new circuit (to control the spindle, and provide feedback from the same), and that I develop and fit some new models. Milling is also less forgiving than FFF printing in practice, so I expect that debugging will be slightly more challenging. However, I think the challenge is manageable, and I think that showing portability across processes is an important feat for the thesis.

5.4 OSAP: an Open Systems Assembly Protocol

The headline contributions in this thesis are underpinned with a distributed systems architecture that I have developed during my time at the CBA that I call OSAP: an *Open Systems Assembly Protocol.* OSAP is based on a thread of research that goes back tens of years in the CBA's history based on *object oriented hardware,* that pairs modular hardware with modular software. I aimed to expand this architecture to span a broader heterogeneity of components and network configurations, to more easily add new firwmares and software integrations, and to enable the development of inter-device data flows (as opposed to star-shaped controller topologies).

I developed OSAP with the high-level goal of enabling asynchronous collaboration between machine developers, based on interoperability and modularity of functional components, noting how the same principles have driven the runaway success of open source software efforts [84].

I reason that if I can show that high-performance machine controllers can be built using modular, generalizeable parts (based on a principle of interoperability via protocol specification), others will be able to take components of this work and extend it.

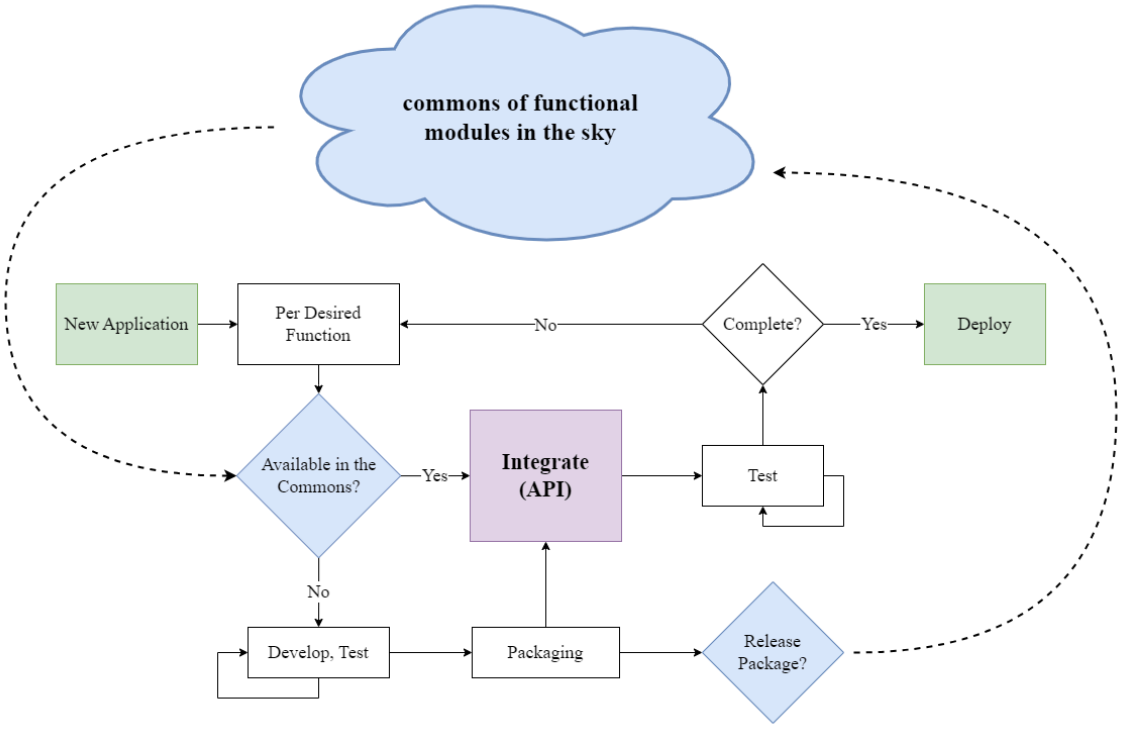


Figure 34: A diagram of how users of Open Source Software developers interchangeably use components from a commons of functional modules, and develop and publish their own. Software has many "built-in" tools for modularity, but hardware tends to resist generalization. Modular hardware approaches try to bridge this gap, to enable the development of a commons of re-useable devices.

OSAP is essentially an implementation of the Open Systems Interconnect model [85] that is guided by the end-to-end principle [86], both of which were foundational during the invention and proliferation of the internet, but neither of which have been rigorously followed in the internet's development [87]. Indeed, machine-scale modular hardware systems (of the kind we deploy in this thesis) are deployed on a heterogeneity of different network links and transport layers [88] whereas the internet is dominated by only a few (TCP, Ethernet, WiFi). The OSI model was meant to enable broad connectivity across heterogeneous link layers, but in practice the field of internetworking in hardware (and in industrial machine systems in particular) is fractured.

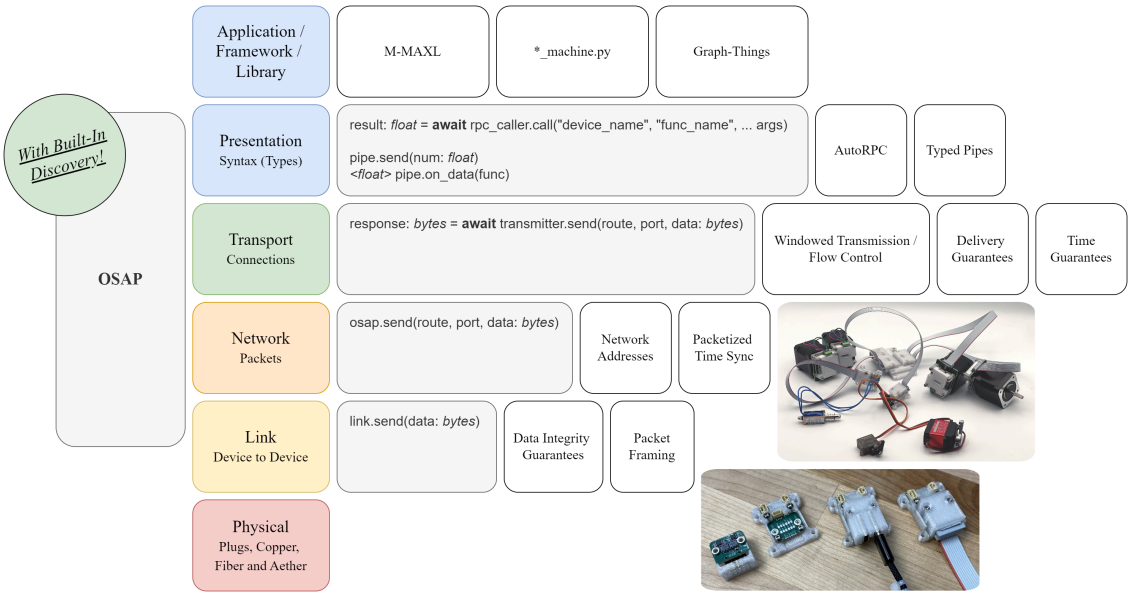


Figure 35: A simplified OSI layering, as implemented in OSAP. Each layer presents well defined software APIs to surrounding layers, in an effort to make components of the system easy to interchange: for example one of the challenges posed by modular hardware systems is that devices typically deploy on heterogeneous link layers (many choose CANBus, others use EtherCat, while simple devices may use I2C or UART based links). OSAP makes an effort to allow for combinations of link layers in any given system.

With OSAP, I am *not* aiming to introduce new standards (which should take more time, and more careful planning and democratic input than I can muster) - I want simply to show that *it is possible* to build performant machine systems using low cost modular components that can each themselves be re-purposed in multiple ways, that provide interfaces that are consumable at multiple levels (network links, RPCs, etc), and that can be connected over a heterogeneity of data link layers. To evaluate OSAP, I will measure its performance in terms of runtime overhead and program size overhead. I will also evaluate its flexibility in deploying across heterogeneous link layers. Qualitatively, I will be able to evaluate where OSAP's structures hampered me, and where they were helpful while deploying the other systems in this thesis.

5.4.a (Simple) Distributed Clock Synchronization

One of the key services that OSAP provides is clock synchronization, which is used as a basis for motion control and for time-series data collection (to build models). Since other clock sync algorithms are complex and consume large amounts of program memory, I developed a simple version from scratch.

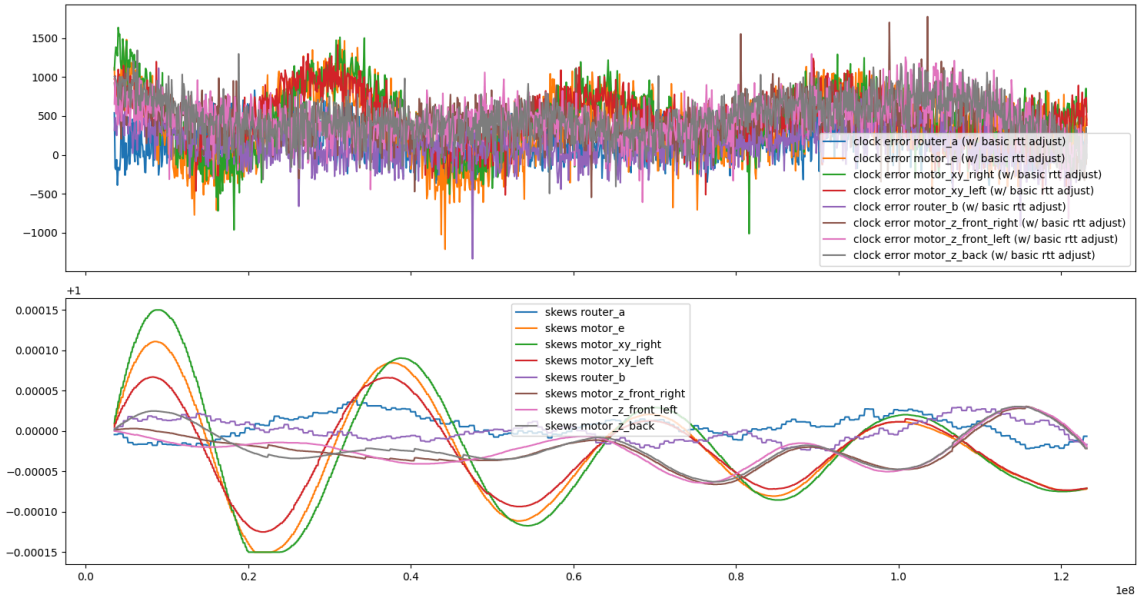


Figure 36: Here I show results from a clock synchronization test. The test polls eight devices (constituting a subset of the Rheo Printer's controller) as the distributed clock sync algorithm settles each devices' clock skew with respect to the chosen *grandmaster* (in this case, the laptop running the test). In the top plot, we see measured errors (these are noisy because packet round trip is not always the same, a key issue with packetized clock sync) - errors stay within +/- one millisecond during the duration of the test, and improve over time. The bottom plot shows each device's clock skew as calculated and updated by the distributed controller. These settle eventually, but reducing their oscillation is something I would like to investigate.

The algorithm is essentially a distributed diffusion routine: each device requests time stamps over all active links, picks the best source, and then skews its own clock in order to minimize errors. The algorithm works well enough for me to complete all of the tasks in this thesis, but I would like to evaluate it more rigorously, since high performance synchronization is a requirement of advanced control systems.

5.4.b Automatic Generation of Control System Components and Representations

In [84], [89] [lerner2002some] and [90], operations and economics researchers studying the success of Open Source Software note two key properties: existing modules should be easy to integrate into new systems, and new modules should be easy to generate and add to the ecosystem. Basically, using components of the ecosystem should be straightforwards, and generating new components should be just as simple. [91] in particular notes that *"In order for a project to be susceptible to sustainable peer production, the integration function must be either low-cost or itself sufficiently modular to be peer-produced in an interative process."*

The controllers developed in this thesis are distributed systems comprising both low-level firmwares and high-level planners / controllers. For those to work together, they need properly articulated software / network interfaces, i.e. they need to be easy to integrate with one another.

Since I have been building all of the modules themselves, it also serves me well if they are easy to generate; I have had a good toy problem for the broader context.

In the state of the art these integrations are maintained by hand: distributed systems are 'authored' twice (once during setup and again as a reciprocal code structure). GCode is effectively the same: new codes are added in firmware (by machine control developers) which are then written down as specs and communicated to CAM developers. In the Appendix on GCode Representations I work up an example of how one GCode is authored in firmware, then as a specification, and how it is used in a program (to compare to the workflow in the listings below). The inevitable misalignments that arise cause hard-to-diagnose errors, and the duplication of efforts wastes time during machine development (and makes it more costly to iterate or udpate designs).

In OSAP, I contribute presentation layer codes that automatically generate these intermediary representations. These codes allow firmware developers to turn any given function call on their device into an RPC (remote procedure call), a common type of interface in distributed systems. RPCs are effectively functions that are implemented on remote devices that can be called from some other device. The following listings work through an example of using one of these RPCs.

Listing 8: An example of a target function, implemented in firmware on a modular device, that we want to generate an interface for. To deploy one, the firmware author can use this `BUILD_RPC()` macro to rollup any given function (with some limits on argument and return types) as a remotely callable (and discoverable) function.

```
float readAvailableVCC(void){
  const float r1 = 10000.0F;
  const float r2 = 470.0F;
  // it's this oddball, no-init ADC stuff,
  // teensy is 10-bits basically, y'all
  uint16_t val = analogRead(PIN_SENSE_VCC);
  // convert to voltage,
  float vout = (float)(val) * (3.3F / 1024.0F);
  // that's at 10k - sense - 470r - gnd,
  // vout = (vcc x r2) / (r1 + r2)
  // (vout * (r1 + r2)) / r2 = vcc
  float vcc = (vout * (r1 + r2)) / r2;
  return vcc;
}

BUILD_RPC(readAvailableVCC, "", "");
```

Listing 9: The 'proxy' code for the function in Listing 8. I automatically authored author these using a script that uses OSAP's network-layer discovery routines in conjunction with the RPC discovery system (in the presentation layer). Even these interface can be generated at runtime, proxy codes are useful for scripting because they enable IDE autocomplete features. I include type hints as well, since type information can be very helpful when authoring scripts.

```
# auto-generated proxy,
class HBridgeProxy:
  # ...
  async def read_available_vcc(self) -> float:
      result = await self._read_available_vcc_rpc.call()
      return cast(float, result)
  # ...
```

Listing 10: An example application code that deploys the proxy shown in Listing 9 to interact with the firmware from Listing 8, following the *object oriented hardware* paradigm mentioned earlier. This small example is a simple script that I wrote to test the function of a generic H-Bridge module.

```
async def main():

    system_map = await osap.netrunner.update_map()

    hbridge = HbridgeSamd21DuallyProxy(osap, "hbridge_dually")
    await hbridge.begin()

    # turn it on
    print("... request voltage")
    await hbridge.set_pd_request_voltage(15)

    print("... await voltage")
    while True:
        vcc_avail = await hbridge.read_available_vcc()
        print(F"avail: ${vcc_avail:.2f}")
        if vcc_avail > 14.0:
            break

    print("pulse...")
    for _ in range(100):
        print("...")
        await hbridge.pulse_bridge(2.0, 1000)
        await asyncio.sleep(1.75)
        await hbridge.pulse_bridge(-1.0, 50)
        await asyncio.sleep(1)

    print("... done!")
```

The `BUILD_RPC` macro uses c++ template programming to generate a wrapper class around the provided function that provides network handles to it, that enables other devices on an OSAP network to query the function for its signature (to learn its name, return type, and argument types), and to call it remotely. OSAP also includes a network discovery routine (to find, name, and address modular devices). Using these two together, we can automatically generate a the interface codes (proxies) required to interface with whatever hardware is connected on the network.

To evaluate success on this front, I develop and deploy automatic proxy generation codes in the machine systems discussed in this thesis. I will also use them with a group of machine builders in *the plotter comp* to generate motion systems. These interfaces can be evaluated quantitatively for performance at runtime (incurring minimal compute or space overheads) and at compile time (minimal overhead program size), and they can be evaluated qualitatively on many fronts:

(1) They should be able to describe most of the breadth of descriptions possible with 'normal' programming (i.e. most common data structures).
(2) They should be consistent, reliable and require minimal programming overhead (burden on the programmer, not the computer) to deploy and ingest.
(3) They should be flexible across many use-cases.
(4) They should be descriptive enough so that they can be used with little documentation (or should contain accomodation for documentation).
(5) They should be easy to interrogate and modify: where the interface inevitably break down, or a lower-level of description is needed, that should be available.

5.4.c A Set of Re-Useable Hardware Modules

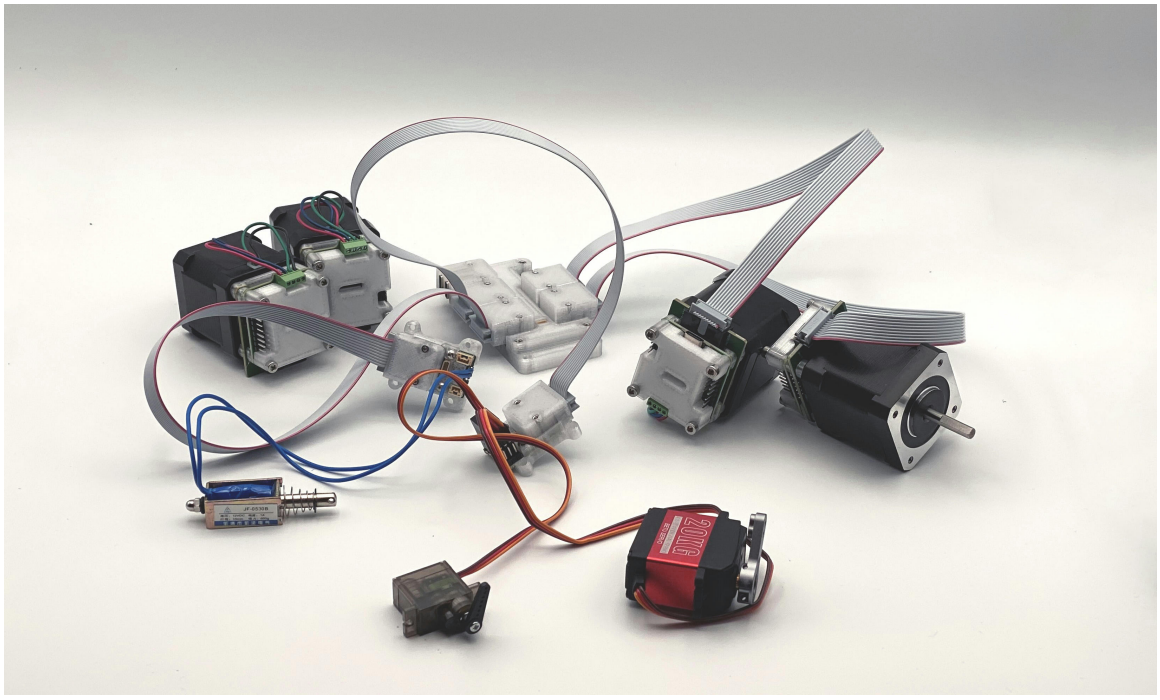


Figure 37: I have been developing this re-useable kit of modular circuits and firmwares over the course of my time at MIT. With the most recent set of these (pictured here), I focused on re-useability of individual circuits.

I built all of the circuits that run the systems in this thesis, and count them as a contribution. With these circuits, I want to provide an example of how small, re-useable hardware modules can be reconfigured across many use-cases without incurring vast size, power and complexity overhead.

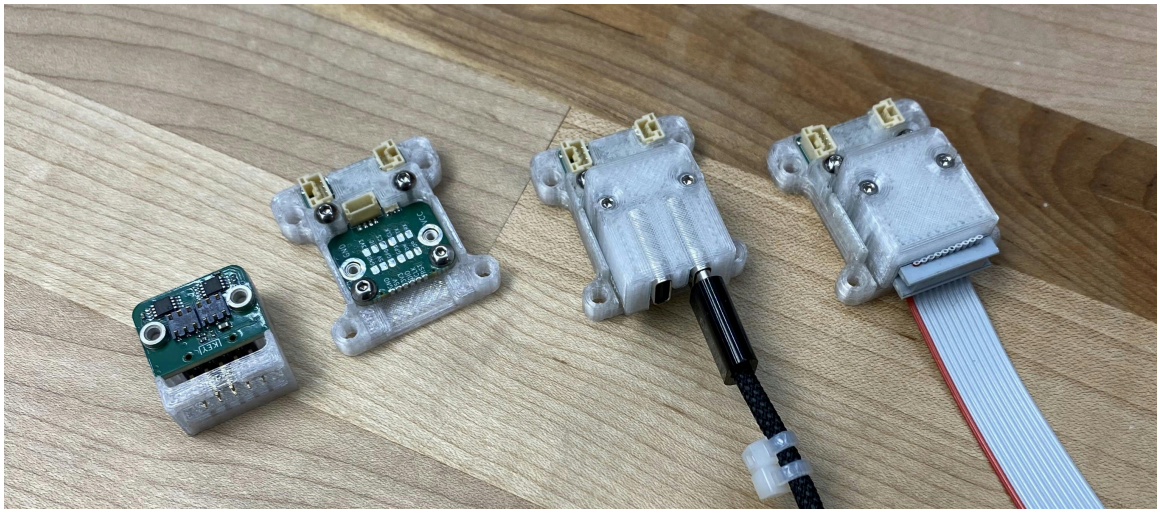


Figure 38: Here I show an interface that this set of circuits uses, that I call a *backpack*. This is a GPIO interface that allows circuits to connect to a broad set of physical communications layers. This means that I can build (a) a set of modular circuits and (b) a set of modular PHYs, and rapidly combine them depending on the context rather than filling out the AxB matrix of unique circuit designs.

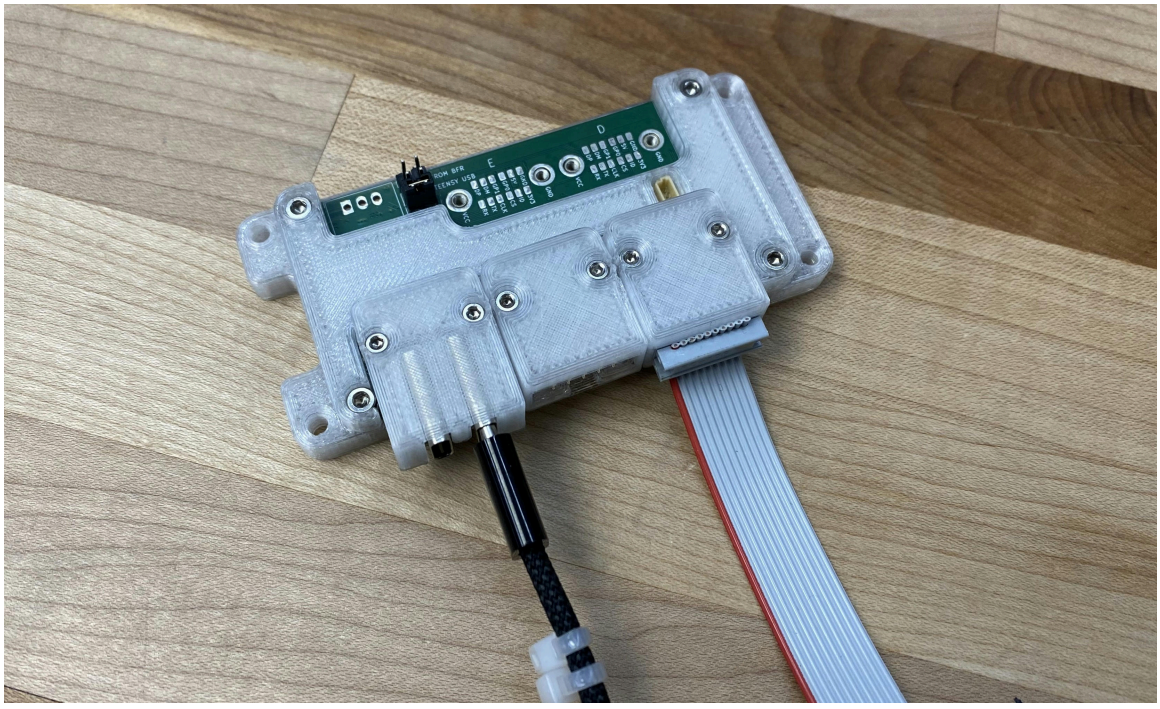


Figure 39: A hub circuit that lets us combine multiple PHYs in the same system, and readily pass messages between any given device. Because of OSAP's flexibility, the hub is not only a message passing device, we can impelement some components of our controllers on it.

5.4.d A Systems Development Environment

From Figure 1 and other discussion, it is clear that the systems deployed in this thesis are (1) always distributed and (2) sometimes messy. Structurally, they are all graphs, but I do not have a tool to visualize them as such. I would like to build a tool to do so.

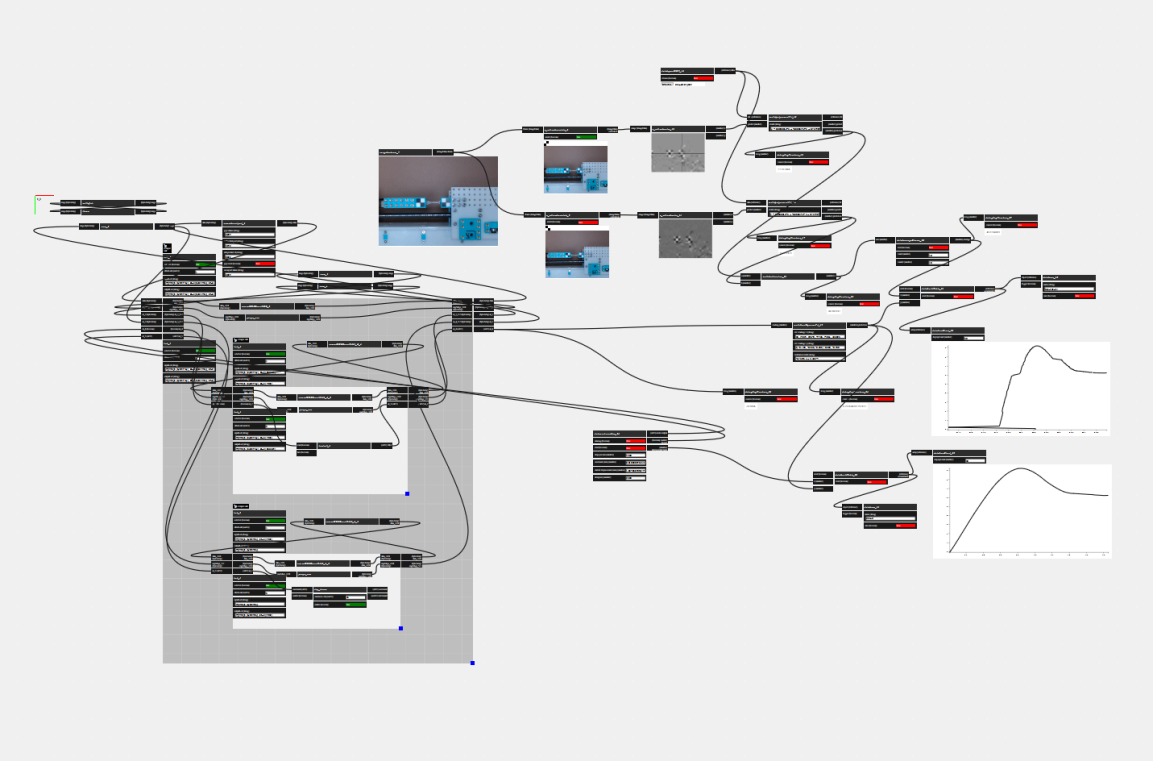


Figure 40: This is a screenshot from the development environment UI that I implemented during my masters thesis [92]. The system was functionally similar to work presented in this thesis, albeit of lower quality systems design. The graph visualizer was nonetheless a promising tool, allowing users to co-ordinate data-flows across all layers in the system.

A graph visualizer and editor would let systems developers quickly debug which hardware modules are connected, inspect their APIs, and build low-level data streams between devices. I have built a similar system in the past, but made the mistake of *over burdening* the graph representation: programs there had to be described entirely as graph entities. In an updated version, I would like to be able to interchangeably use scripting and graphs. I suspect that graph representations will be useful for low-level configurations, but that high level orchestration will take place using scripts.

I plan also to include all of the graph editing API as script elements, meaning that a machine will be able to configure its own low-level systems; I anticipate that this may be useful for machines that need to alter their configurations during runtime, such as tool-changing systems.

6 Timeline

I have just over six months to complete all of the work outlined in this proposal. I believe that most of the hard work is already behind me - this includes developing the core optimization framework, the networks used to deploy it, and much of the hardware that I will use for the final evaluation. The printer prints, and MAXL controls machines. However, I have still not closed the big outer loop to show that MAXL can iteratively develop and then deploy models as it works. There is also some polish required for MAXL before I think it will be ready for the proposed *plotter comp.*

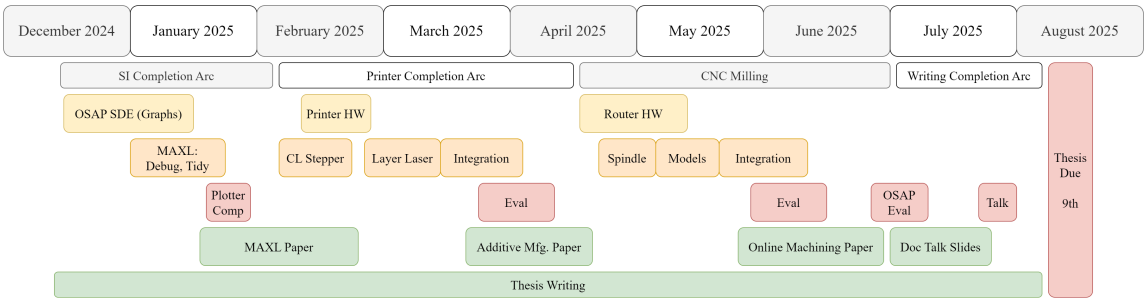


Figure 41: A gantt chart outlining my plans from now until my target graduation date, on the Fall degree list in 2025.

I am proposing to organize my time in three sprints: one to finish MAXL and deploy it for the plotter comp, another to finish the printer and write it up for a journal like Additive Manfuacturing, and a final sprint to map the same strategies to CNC Milling. The latter of these three is where the most questions are; I am not sure yet how best to capture important milling phenomenology.

In the final stretch, I will also write-up and formalize my evaluations of system components within OSAP. Writing has been a good organizing activity, and I hope that I can continue to write the thesis as time progresses - this should also keep me away from side quests. I think that this proposals' ~ fourteen thousand words should provide a good head-start for the framing sections of that document, and the papers I plan to write along the way detailed dives into each subsection.

7 Resources Required

The CBA is exeptionally well resourced, and I have everything I need to complete the tasks I have described in this proposal. This includes our own digital fabrication equipment that I use to build machines, the funding to have new circuits manufactured (although I have already finished most of these), and to purchase other parts like sensors, motors, etc.

8 Conclusion

I am proposing to show that machine systems can be developed using modern, modular approaches to control, in contrast to the pervasive use of an antiquated format. In the course of which, I aim to show that we can use *models* instead of *parameters* to operate these machines, and that these models reduce the brainpower required to successfully operate and build them - as well as help them to out-perform older approaches.

References

Bibliography

[1] P. Blikstein, “Digital Fabrication and Making in Education: The Democratization of Invention,” 2013.

[2] J. Jacobs and N. Peek, “Learning remotely, making locally: Remote digital fabrication instruction during a pandemic,” *ACM [Online]*, 2020.

[3] J. Rutter and N. Gershenfeld, “Haystack Labs.” [Online]. Available: https://www.haystack-mtn.org/haystack-labs

[4] B. Subbaraman, O. de Lange, S. Ferguson, and N. Peek, “The Duckbot: A system for automated imaging and manipulation of duckweed,” *Plos one*, vol. 19, no. 1, p. e296717, 2024.

[5] M. Politi *et al.*, “A high-throughput workflow for the synthesis of CdSe nanocrystals using a sonochemical materials acceleration platform,” *Digital Discovery*, vol. 2, no. 4, pp. 1042–1057, 2023.

[6] J. L. Dávila, B. M. Manzini, M. A. d'Ávila, and J. V. L. da Silva, “Open-source syringe extrusion head for shear-thinning materials 3D printing,” *Rapid Prototyping Journal*, vol. 28, no. 8, pp. 1452–1461, Aug. 2022, doi: 10.1108/RPJ-09-2021-0245.

[7] N. A. Mendez and G. Corthey, [Online]. Available: https://gitlab.com/pipettin-bot/pipettin-bot

[8] P. Dettinger *et al.*, “Open-source personal pipetting robots with live-cell incubation and microscopy compatibility,” *Nature Communications*, vol. 13, no. 1, p. 2999–3000, May 2022, doi: 10.1038/s41467-022-30643-7.

[9] D. Florian, “OTTO Liquid Handling Robot.” [Online]. Available: https://openliquidhandler.com/

[10] N. Gershenfeld, “How to make almost anything: The digital fabrication revolution,” *Foreign Aff.*, vol. 91, p. 43–44, 2012.

[11] N. Gershenfeld, “Rapid-Prototyping of Rapid-Prototyping Machines: How to Make Something that Makes (almost) Anything.” [Online]. Available: https://fab.cba.mit.edu/classes/865.24/index.html

[12] N. Gershenfeld, “How to Make Almost Anything.” [Online]. Available: https://fab.cba.mit.edu/classes/863.24/

[13] D. L. Parnas, “On the Criteria To Be Used in Decomposing Systems into Modules,” vol. 15, no. 12, p. 6–7, 1972.

[14] D. F. Noble, *Forces of Production: A Social History of Industrial Automation*. Routledge, 1984.

[15] T. P. P. A. (PyPA), “pip: The Python Package Installer.” [Online]. Available: https://pip.pypa.io/

[16] P. Research, “PrusaSlicer: Open-Source 3D Printer Slicer Software.” [Online]. Available: https://github.com/prusa3d/PrusaSlicer

[17] M. E. Mackay, "The importance of rheological behavior in the additive manufacturing technique material extrusion," *Journal of Rheology*, vol. 62, no. 6, pp. 1549–1561, 2018.

[18] T. J. Coogan and D. O. Kazmer, "Prediction of interlayer strength in material extrusion additive manufacturing," *Additive Manufacturing*, vol. 35, p. 101368–101369, 2020.

[19] I. Autodesk, "Autodesk Fusion 360: Cloud-Based 3D CAD, CAM, CAE & PCB Software." [Online]. Available: https://www.autodesk.com/products/fusion-360

[20] P. Research, "Linear Advance in FFF Printing." [Online]. Available: https://help.prusa3d.com/article/linear-advance_2252

[21] J. Go, S. N. Schiffres, A. G. Stevens, and A. J. Hart, "Rate limits of additive manufacturing by fused filament fabrication and guidelines for high-throughput system design," *Additive Manufacturing*, vol. 16, pp. 1–11, 2017.

[22] S. Lavernhe, C. Tournier, and C. Lartigue, "Optimization of 5-axis high-speed machining using a surface based approach," *Computer-Aided Design*, vol. 40, no. 10–11, pp. 1015–1023, 2008.

[23] I. Moyer, "CoreXY: A Mechanism for Motion Control in 3D Printers and CNC Machines." [Online]. Available: https://corexy.com/

[24] G. Research, "JAX: Autograd and XLA for High-Performance Machine Learning Research." [Online]. Available: https://github.com/google/jax

[25] DeepMind, "Optax: Composable Gradient Processing and Optimization Library for JAX." [Online]. Available: https://github.com/deepmind/optax

[26] S. L. Brunton and J. N. Kutz, "Chapter 10: Data Driven Control," *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, Cambridge, United Kingdom, pp. 389–408, 2022.

[27] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2018, pp. 1–9.

[28] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, "Data-driven MPC for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, 2021.

[29] S. Kuindersma *et al.*, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous robots*, vol. 40, pp. 429–455, 2016.

[30] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019, doi: 10.1007/s12532-018-0139-4.

[31] R. Verschueren *et al.*, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, 2021.

[32] S. L. Brunton and J. N. Kutz, "Chapter 11: Reinforcement Learning," *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, Cambridge, United Kingdom, pp. 419–444, 2022.

[33] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[34] Y. Song, S. Kim, and D. Scaramuzza, "Learning Quadruped Locomotion Using Differentiable Simulation." [Online]. Available: https://arxiv.org/abs/2403.14864

[35] B. N. Turner, R. Strong, and S. A. Gold, "A review of melt extrusion additive manufacturing processes: I. Process design and modeling," *Rapid prototyping journal*, vol. 20, no. 3, pp. 192–204, 2014.

[36] T. J. Coogan and D. O. Kazmer, "In-line rheological monitoring of fused deposition modeling," *Journal of Rheology*, vol. 63, no. 1, pp. 141–155, 2019.

[37] T. Sanladerer, "InFiDEL Filament Sensor." [Online]. Available: https://www.youtube.com/watch?v=RYgdLPe_T0c

[38] D. O. Kazmer, A. R. Colon, A. M. Peterson, and S. K. Kim, "Concurrent characterization of compressibility and viscosity in extrusion-based additive manufacturing of acrylonitrile butadiene styrene with fault diagnoses," *Additive Manufacturing*, vol. 46, p. 102106–102107, Oct. 2021, doi: 10.1016/j.addma.2021.102106.

[39] J. R. Read, J. E. Seppala, F. Tourlomousis, J. A. Warren, N. Bakker, and N. Gershenfeld, "Online Measurement for Parameter Discovery in Fused Filament Fabrication," *Integrating Materials and Manufacturing Innovation*, vol. 13, no. 2, pp. 541–554, 2024.

[40] M. E. Mackay, Z. R. Swain, C. R. Banbury, D. D. Phan, and D. A. Edwards, "The performance of the hot end in a plasticating 3D printer," *Journal of Rheology*, vol. 61, no. 2, pp. 229–236, 2017.

[41] J. A. Afonso, J. L. Alves, G. Caldas, B. P. Gouveia, L. Santana, and J. Belinha, "Influence of 3D printing process parameters on the mechanical properties and mass of PLA parts and predictive models," *Rapid Prototyping Journal*, vol. 27, no. 3, pp. 487–495, 2021.

[42] S. Deshwal, A. Kumar, and D. Chhabra, "Exercising hybrid statistical tools GA-RSM, GA-ANN and GA-ANFIS to optimize FDM process parameters for tensile strength improvement," *CIRP Journal of Manufacturing Science and Technology*, vol. 31, pp. 189–199, 2020.

[43] A. Qattawi, B. Alrawi, A. Guzman, and others, "Experimental optimization of fused deposition modelling processing parameters: a design-for-manufacturing approach," *Procedia Manufacturing*, vol. 10, pp. 791–803, 2017.

[44] O. Luzanin, D. Movrin, V. Stathopoulos, P. Pandis, T. Radusin, and V. Guduric, "Impact of processing parameters on tensile strength, in-process crystallinity and mesostructure in

FDM-fabricated PLA specimens," *Rapid Prototyping Journal*, vol. 25, no. 8, pp. 1398–1410, 2019.

[45] P. Ferretti *et al.*, "Relationship between FDM 3D printing parameters study: parameter optimization for lower defects," *Polymers*, vol. 13, no. 13, p. 2190–2191, 2021.

[46] C. S. Davis, K. E. Hillgartner, S. H. Han, and J. E. Seppala, "Mechanical strength of welding zones produced by polymer extrusion additive manufacturing," *Additive Manufacturing*, vol. 16, pp. 162–166, Aug. 2017, doi: 10.1016/j.addma.2017.06.006.

[47] P. Wu, "Modeling and Feedforward Deposition Control in Fused Filament Fabrication," 2024.

[48] K. Dunwoody, "Automated identification of cutting force coefficients and tool dynamics on CNC machines," 2010.

[49] C. Sharma and others, "Automatic modeling of machining processes," 2021.

[50] Y. Altintaş and E. Budak, "Analytical prediction of stability lobes in milling," *CIRP annals*, vol. 44, no. 1, pp. 357–362, 1995.

[51] D. Aslan and Y. Altintas, "On-line chatter detection in milling using drive motor current commands extracted from CNC," *International Journal of Machine Tools and Manufacture*, vol. 132, pp. 64–80, 2018.

[52] C. M. G. Bort, M. Leonesio, and P. Bosetti, "A model-based adaptive controller for chatter mitigation and productivity enhancement in CNC milling machines," *Robotics and Computer-Integrated Manufacturing*, vol. 40, pp. 34–43, 2016.

[53] R. Ward *et al.*, "Machining Digital Twin using real-time model-based simulations and lookahead function for closed loop machining control," *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 11, pp. 3615–3629, 2021.

[54] T. L. Schmitz and K. S. Smith, "Machining dynamics," *Springer*, p. 303–304, 2009.

[55] N. Peek, "Making machines that make: object-oriented hardware meets object-oriented software," 2016.

[56] I. E. Moyer, "A gestalt framework for virtual machine control of automated tools," 2013.

[57] N. Gershenfeld, R. Krikorian, and D. Cohen, "The internet of things," *Scientific American*, vol. 291, no. 4, pp. 76–81, 2004.

[58] M. M. Smith, "Recursive Robotic Assemblers," 2023.

[59] A. Abdel-Rahman, C. Cameron, B. Jenett, M. Smith, and N. Gershenfeld, "Self-replicating hierarchical modular robotic swarms," *Communications Engineering*, vol. 1, no. 1, p. 35–36, 2022.

[60] J. Devine *et al.*, "Plug-and-play physical computing with Jacdac," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 3, pp. 1–30, 2022.

[61] T. Ball, P. de Halleux, J. Devine, S. Hodges, and M. Moskal, "Jacdac: Service-based prototyping of embedded systems," *Proceedings of the ACM on Programming Languages*, vol. 8, no. PLDI, pp. 692–715, 2024.

[62] P. Blikstein, "Gears of our childhood: constructionist toolkits, robotics, and physical computing, past and future," in *Proceedings of the 12th international conference on interaction design and children*, 2013, pp. 173–182.

[63] S. A. Papert, *Mindstorms: Children, computers, and powerful ideas*. Basic books, 2020.

[64] L. Valk and D. Lechner, "PyBricks: robotics made easy." [Online]. Available: https://pybricks.com/

[65] J. R. Read, L. Mcelroy, Q. Bolsee, B. Smith, and N. Gershenfeld, "Modular-Things: Plug-and-Play with Virtualized Hardware," in *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–6.

[66] J. R. Read, N. Peek, and N. Gershenfeld, "MAXL: Distributed Trajectories for Modular Motion," in *Proceedings of the 7th Annual ACM Symposium on Computational Fabrication*, 2023.

[67] F. Fossdal, R. Heldal, and N. Peek, "Interactive digital fabrication machine control directly within a CAD environment," in *Proceedings of the 6th Annual ACM Symposium on Computational Fabrication*, 2021, pp. 1–15.

[68] F. H. Fossdal, V. Nguyen, R. Heldal, C. L. Cobb, and N. Peek, "Vespidae: A Programming Framework for Developing Digital Fabrication Workflows," in *Proceedings of the 2023 ACM Designing Interactive Systems Conference*, 2023, pp. 2034–2049.

[69] J. Tran O'Leary, G. Benabdallah, and N. Peek, "Imprimer: Computational Notebooks for CNC Milling," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–15.

[70] J. Vasquez, H. Twigg-Smith, J. Tran O'Leary, and N. Peek, "Jubilee: An extensible machine for multi-tool fabrication," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[71] K. Dunn, C. Feng, and N. Peek, "Jubilee: A case study of distributed manufacturing in an open source hardware project," *Journal of Open Hardware*, vol. 7, no. 1, 2023.

[72] N. Peek and L. Pozzo, "Pathways to Open-Source Hardware for Laboratory Automation." [Online]. Available: https://depts.washington.edu/machines/scienceautomation/

[73] X.-M. Zhang *et al.*, "Networked control systems: A survey of trends and techniques," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 1, pp. 1–17, 2019.

[74] L. Zhang, H. Gao, and O. Kaynak, "Network-induced constraints in networked control systems—A survey," *IEEE transactions on industrial informatics*, vol. 9, no. 1, pp. 403–416, 2012.

[75] F.-L. Lian, J. Moyne, and D. Tilbury, "Network design consideration for distributed control systems," *IEEE transactions on control systems technology*, vol. 10, no. 2, pp. 297–307, 2002.

[76] J. K. Yook, D. M. Tilbury, and N. R. Soparkar, "Trading computation for bandwidth: Reducing communication in distributed control systems using state estimators," *IEEE transactions on control systems technology*, vol. 10, no. 4, pp. 503–518, 2002.

[77] M. Di Natale, "Scheduling the CAN bus with earliest deadline techniques," in *Proceedings 21st IEEE Real-Time Systems Symposium*, 2000, pp. 259–268.

[78] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[79] J. C. Eidson, M. Fischer, and J. White, "IEEE-1588™ Standard for a precision clock synchronization protocol for networked measurement and control systems," in *Proceedings of the 34th Annual Precise Time and Time Interval Systems and Applications Meeting*, 2002, pp. 243–254.

[80] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 933–940, 1987.

[81] F. Holmer, "The Continuity of Splines." [Online]. Available: https://www.youtube.com/watch?v=jvPPXbo87ds

[82] P. Virtanen *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[83] CreativeTools, "3D Benchy."

[84] N. Eghbal, *Working in public: the making and maintenance of open source software*. Stripe Press, 2020.

[85] I. O. for Standardization, *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. in ISO/IEC 7498-1:1994. Geneva, Switzerland: ISO/IEC, 1994. [Online]. Available: https://www.iso.org/standard/20269.html

[86] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 4, pp. 277–288, 1984.

[87] E. I. S. Group, "Next Generation Protocols (NGP); An example of a non-IP network protocol architecture based on RINA design principles." [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NGP/001_099/009/01.01.01_60/gr_ngp009v010101p.pdf

[88] F.-L. Lian, J. Moyne, and D. Tilbury, "Performance evaluation of control networks: Ethernet, ControlNet, and DeviceNet," *IEEE Control Systems*, vol. 21, no. 1, pp. 66–83, Feb. 2001, doi: 10.1109/37.898793.

[89] Y. Benkler, A. Shaw, and B. M. Hill, "Peer production: A form of collective intelligence," *Handbook of collective intelligence*, vol. 175, 2015.

[90] C. Hess, E. Ostrom, and G. M. McCombs, "Understanding Knowledge as a Commons: From Theory to Practice.," *College and Research Libraries*, vol. 69, no. 1, pp. 92–94, 2008.

[91] Y. Benkler, “Coase's Penguin, or, Linux and" The Nature of the Firm",” *Yale law journal*, pp. 369–446, 2002.

[92] J. R. Read, “Distributed dataflow machine controllers,” 2020.