MAXL: Distributed Trajectories for Modular Motion

Jake Robert Read
jakeread@mit.edu
MIT Center for Bits and Atoms
Cambridge, Massachusetts, USA

Nadya Peek
nadya@uw.edu
University of Washington
Machine Agency
Seattle, Washington, USA

Neil Gershenfeld
neil.gershenfeld@cba.mit.edu
MIT Center for Bits and Atoms
Cambridge, Massachusetts, USA

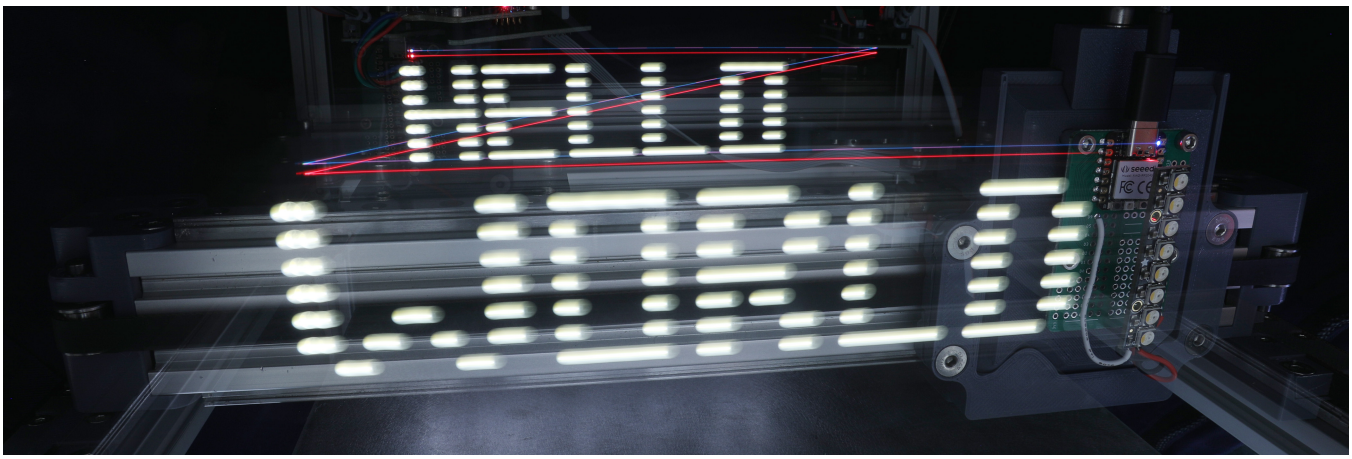

Figure 1: MAXL (for Modular Acceleration eXecution Library) is a distributed software system that allows multiple microcontrollers to work together to execute time-synchronized motion trajectories. MAXL exposes generalized APIs to high- and low-level software participants that allows integration of many different processes. Shown here is a long-exposure photograph of light-painting motion trajectory made with MAXL.

ABSTRACT

Computational fabrication relies on time-synchronized operation of various machine components. Designing machines for novel workflows is of interest to the computational fabrication community, but designing control systems for these machines, especially with diverse actuators and sensors, remains challenging. We present MAXL, a modular, extensible machine control architecture that enables synchronous control of heterogeneous components. We contribute (1) a design pattern for a distributed trajectory object with one author and multiple readers, (2) high- and low-level APIs for interfacing this trajectory object to modular hardware and to digital fab applications (3) a simple time-synchronization algorithm and queuing scheme for distributing the trajectory object, and (4) an extensible hardware implementation of MAXL. We demonstrate MAXL's utility in developing new computational fabrication applications by integrating it into two motion control applications; one for time-synchronized data output (light-painting), and the other for time-synchronized data retrieval (from an accelerometer). Finally, we discuss how MAXL can be extended for use in future machine applications.

CCS CONCEPTS

• **Computer systems organization** → **External interfaces for robotics**; **Reconfigurable computing**; • **Software and its engineering** → *Object oriented development*; • **Networks** → **Programming interfaces**; • **Human-centered computing** → **Systems and tools for interaction design**.

KEYWORDS

Machine Control, Networked Systems, Modularity, Fabrication

ACM Reference Format:
Jake Robert Read, Nadya Peek, and Neil Gershenfeld. 2023. MAXL: Distributed Trajectories for Modular Motion. In *Symposium on Computational Fabrication (SCF '23), October 8–10, 2023, New York City, NY, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3623263.3623362

SCF '23, October 8–10, 2023, New York City, NY, USA

ACM ISBN 979-8-4007-0319-5/23/10.
https://doi.org/10.1145/3623263.3623362

1 INTRODUCTION

Computational fabrication relies on time-coordinated actions. CNC milling relies on coordination between X, Y, and Z axes while controlling the rotational speed of a spindle. Laser cutting relies on motion coordination while modulating laser power and firing rate. Extrusion-based 3D printing relies on motion coordination while heating a build plate and melting filament for extrusion.

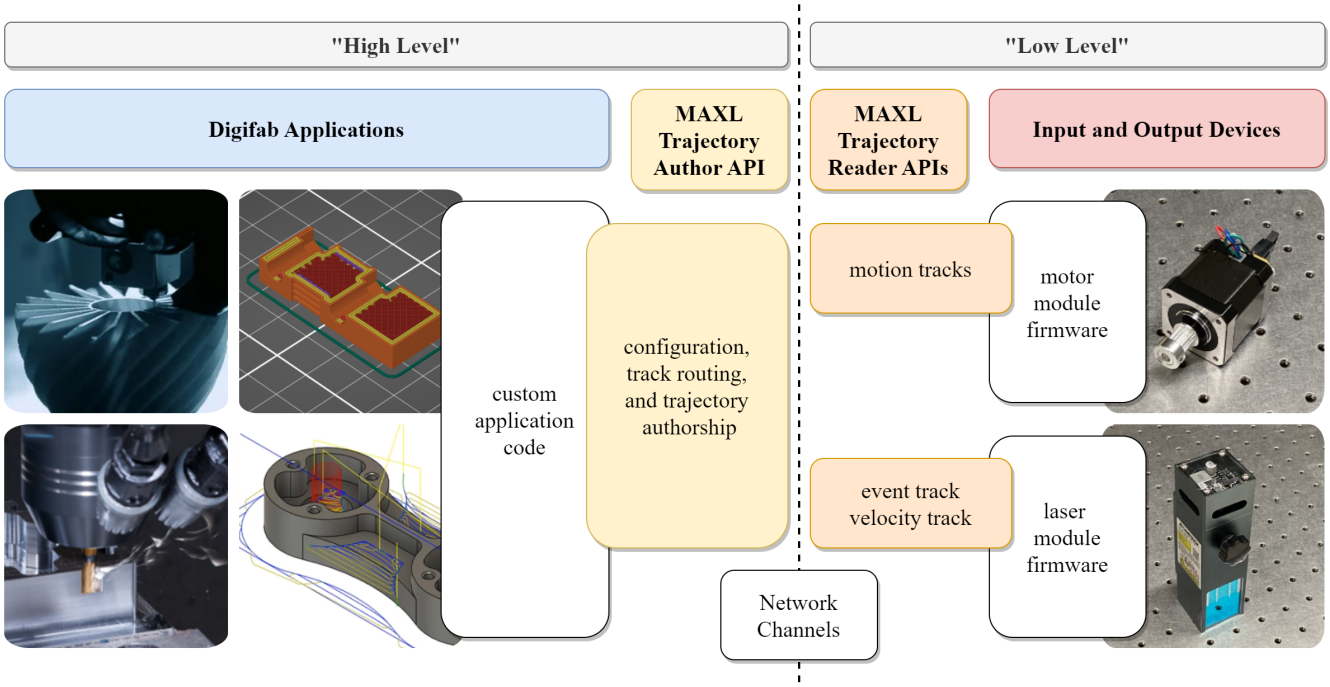

Figure 2: MAXL is a distributed application that is directly embedded into system authors' code. At the high level, MAXL provides APIs for configuring machines and for interfacing with a distributed trajectory object that we discuss in Section 3.1 (i.e. writing new motion segments into a queue). In low-level components (hardware modules), MAXL presents APIs for time-synchronized reading of trajectory information. The system is a pure software solution to motion control; rather than requiring that users acquire particular hardware components, it instead offers useful APIs that systems authors can use to integrate motion control into their own custom hardware, or to extend existing systems with new hardware.

Each of these well-established processes are served by custom-built hardware. However, computational fabrication researchers are interested in developing new workflows that accommodate experimental materials and machines, e.g., [Bhooshan et al. 2020; Rivera et al. 2023; Vasquez et al. 2020; Wang et al. 2016]. Building custom hardware controllers to explore this design space is challenging. A common approach is to hack existing machines, adding functionality by overloading existing practice, e.g., by modifying off-the-shelf 3D printers [Rivera and Hudson 2019; Roumen et al. 2016]. However, this approach doesn't generalize well and requires expertise—adding additional end effectors requires low-level access for timing, configuration, and safety, which is difficult to achieve [Landwehr Sydow et al. 2022].

To address the challenges associated with designing custom machine controllers, we seek to contribute a modular motion and end-effector control system. A main design goal is for the system to be extensible by plugging in additional boards, rather than needing to implement new hardware. Furthermore, the system should be straightforward to use by people who are prototyping workflows, for example computational fabrication researchers developing new machines and materials. Finally, it should be able to accommodate many different types of processes with a single set of core features.

In this short paper, we present MAXL, a Modular Acceleration eXecution Library, an extensible and modular motion control system. MAXL's core design attribute is considering the trajectories of multiple devices as one distributed object, and providing clear software APIs to interface with that object. A MAXL "author" can create an object with trajectories for multiple MAXL "readers". Here an example author might be computational fabrication software, such as a 3D printing slicer, and example readers would be 3D printer components such as stepper motors, extruders, and heated beds. We illustrate where MAXL sits in a computational fabrication workflow in Figure 2. MAXL creates a new type of connection between digital fabrication applications (e.g., Computer-Aided Design (CAD) programs such as Fusion360 or Rhino3D or Computer-Aided Manufacturing (CAM) programs such as 3D printing slicers or toolpath planning programs) and digital fabrication machines (e.g., 3D printers, CNC mills, and more experimental machines).

The rest of this short paper is structured as follows. First, we describe how MAXL relates to other research efforts in Section 2. We then provide details of MAXL's implementation, including its time synchronization methods and segment buffering in Section 3. We evaluate MAXL by showing key features of the system in demonstrations in Section 4. Finally, we close with a discussion and future work.

2 RELATED WORK

Our goal of supporting the development of experimental computational fabrication machines is widely shared. In this section, we describe related research efforts and how our approach draws from and is distinct from this prior work.

2.1 Exploratory Digital Fabrication

There is increasing interest from the computational fabrication community to create alternative methods for interfacing with digital fabrication machines. This ranges from being able to control the machine in real time, enabling interactive [Fossdal et al. 2021; Kim et al. 2018; Tian et al. 2019, 2018; Willis et al. 2010] and exploratory [Tran O'Leary et al. 2023, 2022] fabrication, to creating modes of interaction that prioritize material exploration in a computational fabrication process [Tokac et al. 2022], to allowing for style transfer with different robotic toolpaths [Ma et al. 2020], to allowing the use of found objects such as tree branches in the fabrication process [Larsson et al. 2019]. This research is in addition to the development of new material affordances based on computational fabrication methods, such as programmed deformation [Forman et al. 2020; Ion et al. 2016; Tricard et al. 2020] and 4D printing [An et al. 2018; Wang et al. 2018]. We are inspired by this research and seek to advance it by developing a motion system that can support similar lines of inquiry. MAXL makes real-time control of computational fabrication systems straightforward by exposing each element of a fabrication machine to the control software. This enables applications that require material tuning and iteration such as listed above, and also supports applications that seek to move away from mesh-based toolpath planning methods [Keeter 2013; Nandi et al. 2018].

2.2 Digital Fabrication Machine Building

Building custom machines for digital fabrication has also been explored through prior research. However, this has mainly focused on the mechanical components of the machines such as the mechanical motion [Fossdal et al. 2020; Peek et al. 2017] or toolchangers [Vasquez et al. 2020]. MAXL complements this prior research by focusing on motion- and end-effector control.

Commercial and off-the-shelf solutions for machine control such as GRBL [GRBL 2023] or Replicape [Replicape 2023] are difficult to extend with additional controllers or modules. Especially users who wish to add time-sensitive control of additional components, such as modulating the temperature of a hot-end based on its dwell time on styrofoam, will find extending GCode based solutions constraining. The most closely aligned approach is that of Klipper [Klipper3D 2023], 3D printer firmware which also prioritizes extensibility. However, Klipper is also challenging to configure, especially for tasks that are outside of its intended 3D printing applications, and uses a rigid trajectory representation that makes integrating new firmwares within the system difficult. MAXL is designed for reconfigurability and applications that go beyond 3D printing, thereby expanding the landscape of machine control options and enabling new applications.

3 SYSTEM IMPLEMENTATION

MAXL introduces a design pattern that allows modular computing devices to work together on the execution of a time-synchronized trajectory by re-casting the trajectory as a distributed object with one author (written in a high level programming language) and multiple readers (written in embedded C++ and deployed using the Arduino framework).

In a higher level view, MAXL is situated between path generators (i.e. 3D Printing slicers and CNC Milling CAM tools) and machine hardware (i.e. stepper motors, hotends, etc) as a distributed software object, as diagrammed in Figure 2. It is primarily concerned with orchestrating the execution of tool-paths (that are generated in computer programs) in the real-world, on hardware that must respect networking and physical constraints like acceleration limits.

MAXL takes a software-first approach, allowing system developers to interface their trajectory with custom hardware using an Arduino [Arduino LLC 2023] library, and with their path generation application using a JavaScript API. It is intended to be embedded directly within digital fab applications in around the same place as most existing applications would export GCode, i.e., just after path generation and before path execution.

```
// Configuring a MAXL Trajectory Author Object

let maxl = createMAXL({

  // assigning names to positional DOF:
  motionAxes: ["x", "y"],

  // and defining event channels:
  eventChannels: ["powerOut"],

  // managing subscriptions:
  subscriptions: [{
      device: "motorOne",  // the device to route the track to
      track: "x",          // the output to route
      reader: "stepper"    // the callback, within the device
    },{
      device: "motorTwo",
      track: "y",
      reader: "stepper",
    },{
      // the laser module to event and velocity tracks,
      device: "laserModule",
      track: "velocity",
      reader: "velocityReader",
    },{
      // we can route multiple tracks to the same device !
      device: "laserModule",
      track: "powerOut",
      reader: "laserPower",
    },

  ]
});
```

Listing 1: This is an example of a MAXL configuration object from a laser cutter that has X and Y axes of motion, and that routes one event track and the velocity track to a laser output module.

```
for(let p = 0; p < path.length; p ++){
  await maxl.addSegmentToQueue({
    endPosition: maxl.testPath[p],
    maxVelocity: 250
  });
}
```

Listing 2: We use JavaScript's 'async' semantics to control program flows.

For example, suppose a machine builder constructed a pen plotting machine and they wanted to build an application where user pen strokes are transmitted to a machine in real-time. To do so, they might start by building a sketching tool that ingests input on a tablet as polylines. Using a MAXL machine, they could add

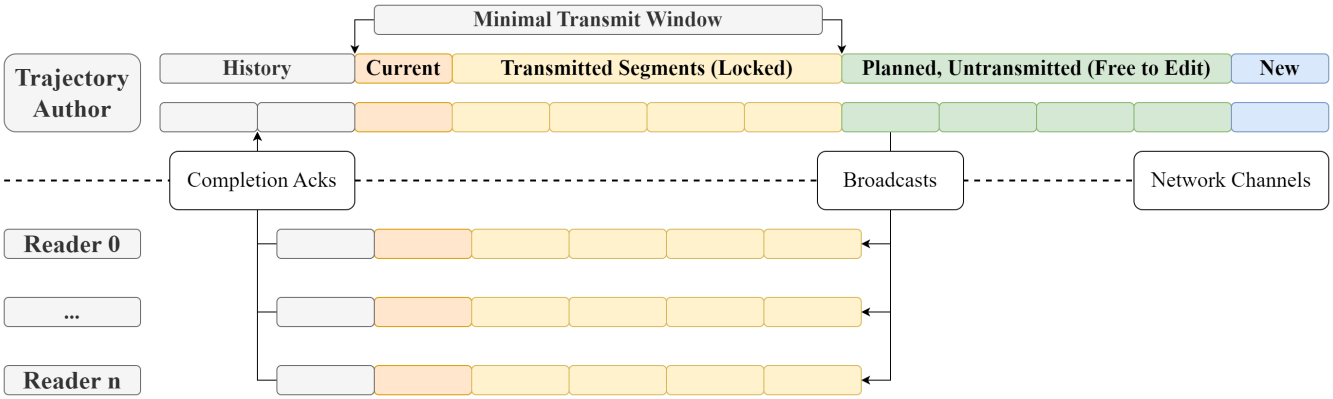

Figure 3: A distributed trajectory object is key to MAXL's operation. In this figure, we see the trajectory as a queue of segments that transition through four states: unplanned, planned, transmitted, and historical. Segments that have been transmitted are locked to prevent further editing by the author, but any others in the queue can be modified. MAXL contributes a design pattern for minimal buffering (Section 3.2) that defines the size of this window.

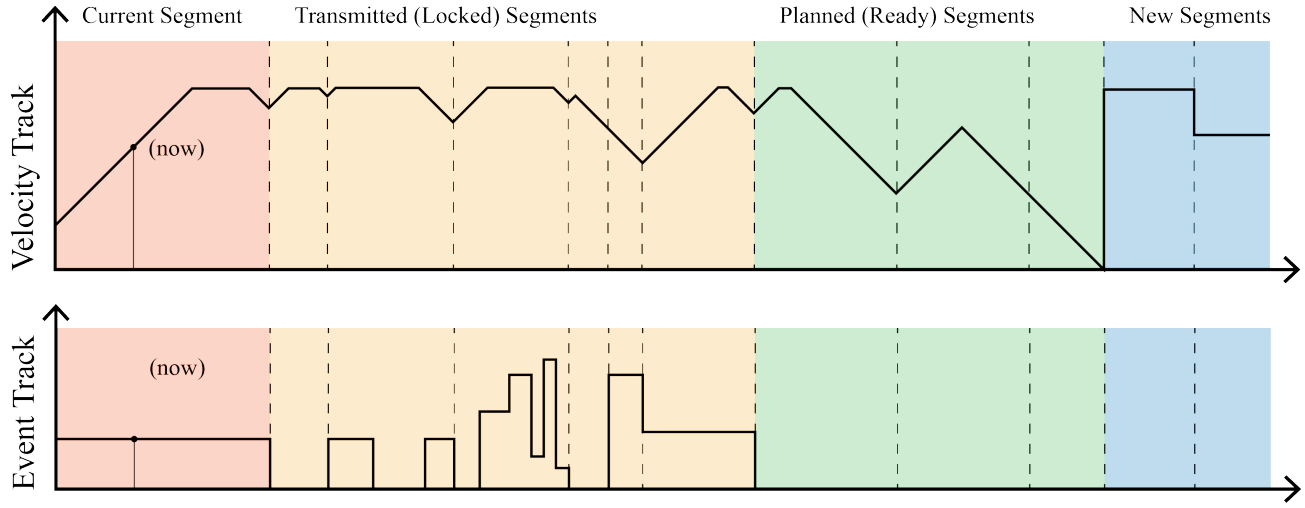

Figure 4: MAXL trajectories can be further decomposed into *tracks,* each of which is a single time-series function. This figure shows an example of what two tracks - one velocity track and one event track - may look like in practice. Velocity tracks, generated as a result of MAXL's motion optimization routine, encode constant acceleration with linked velocities at segment junctions. Event channels are simple time-stamped step functions, and provide the utility of turning remote devices on and off (or changing their levels) in time-synchronized manner. Typical uses may include setting laser power levels during engraving jobs, pulsing inkjet heads at precise intervals, or triggering sensor readings. In our light-painting demonstration from Section 4.1, we use an event channel to write LED states at precise locations. Section 3.4 explains in more detail how event tracks are generated.

these trajectory objects directly to the distributed queue (which we explain in Section 3.1) via an API like the one shown in Listing 2, while monitoring MAXL's internal copy of the trajectory state to track, in real-time, the machine's progress.

To build their machine, they could use MAXL as a library for motion control on a custom board, or simply deploy it on existing hardware of their choosing; in the latter case, they would only be responsible for authoring the code that interfaces from MAXL's trajectory object to the board's hardware API, two examples of which are shown in Listings 3 and 4.

3.1 The Distributed Trajectory

The trajectory object, diagrammed in Figure 3, is made up of discrete segments of motion. These segments each encode a linear move between two points in the machines' defined coordinate space. MAXL's basic task is to ingest these segments from a computational

design application, apply acceleration constraints to them, and coordinate their execution across modular machine hardware.

Segments are ingested in an *unplanned* state (rendered in blue in Figures 3 and 4), meaning they arrive without precise speed profiles. For example, a typical GCode instruction `"G0 X90 F100"` encodes only a maximum cruise velocity of $100 units/sec$ for the given move, but does not specify entry and exit velocities. Were these requests to be executed verbatim, acceleration constraints would likely be violated, meaning that machines would be asked to exert torques that their designs are incapable of. To prevent this, MAXL optimizes entry, exit and cruise speeds within each segment according to a junction-deviation-based lookahead scheme that is common amongst many motion controllers [GRBL 2023; Replicape 2023; Smoothieware 2023]. An example of resulting speed profiles is drawn in Figure 4.

MAXL makes *planned* segments available to trajectory readers throughout the system by maintaining remote queues of segments across a collection of networked devices, and time synchronizing those devices such that they each know where, exactly, within the queue they are meant to be at any given time. The trajectory author also maintains access to planned, unplanned, current and historical segments, as well as a real-time estimate of the trajectory's current state.

```
// supposing we have a hardware API that lets us issue steps
void stepperStep(boolean dir);

// and we can define the conversion between world-units
// and stepper motion,
float stepsPerUnit = 100.0F;
float unitsPerStep = 1.0F / stepsPerUnit;
float stepModulo = 0.0F;

// we can write a callback that receives position updates
// as well as position deltas (since the last call)
// from the track that this motor is subscribed to

void onPositionUpdate(float position, float delta){
  // this simply checks if we have crossed a step threshold,
  // and issues a step if so
  stepModulo += delta;
  if(stepModulo > unitsPerStep){
    stepperStep(true);
    stepModulo -= unitsPerStep;
  }
  if (stepModulo < -unitsPerStep){
    stepperStep(false);
    stepModulo += unitsPerStep;
  }
}

// we enable this track by declaring the object,
// passing along our callback:

MAXL_TrackPositionLinear stepperTrack(
   "stepper",        // the reader's name
   onPositionUpdate  // the reader's callback
);
```

Listing 3: This is a code snippet from our stepper motor module, where a positional track is defined and the motor is programmed to step when position deltas exceed the size of one step.

3.2 Minimal Buffering

Once a segment has been transmitted to readers, it becomes locked from further edits by user programs or by the speed optimization routine. Because locking portions of the trajectory is not often desirable (it is best to be able to stop or adjust machine paths on the fly, for example), and because trajectory readers have limited buffer depths, MAXL makes an effort to minimize the number of transmitted segments. To do so, it deploys a queuing scheme where a few criteria are maintained, which we list in Table 1.

The *Minimal Buffer Time* criteria can conflict with the *Maximum Buffer Size* criteria in cases where trajectories are composed of very many small segments, each of which is traversed quickly. Unsurprisingly, this means that faster underlying networks (smaller RTTs) will allow MAXL to handle more detailed trajectories. At the moment, this condition simply results in the reporting of an error condition and a systems halt, but we mention other solutions to this trouble in our discussion on future work (Section 5). It has not yet posed a serious performance constraint.

```
// supposing we have a laser API that lets us write
// power values from 0-100.0F
void writeLaserPower(float pwr){};

// we'll keep track of the currently requested relative-power:
float unscaledPower = 0.0F;

// and we can update that with an event track,
// that the trajectory author can use to set
// a laser power level:

void onLaserOnOffEvents(float val){
  unscaledPower = val;
}

MAXL_TrackEvent laserPowerTrack(
  "laserPower",
  onLaserOnOffEvents
);

// then we can additionally scale it by speed, since we need
// to match power per mm, rather than power per second
// using a second track listener, this time for velocity

void onSpeedUpdate(float rate){
  float scaledPower = unscaledPower * rate;
  writeLaserPower(scaledPower);
}

MAXL_TrackVelocity velocityTrack(
  "velocityReader",
  onSpeedUpdate
);
```

Listing 4: This is an example from a laser power module, where an event track sets laser power per millimeter of travel, and a speed track is used to continuously update the power per second as speeds fluctuate. The point here is that MAXL does not interface with or include hardware implementations directly, rather it serves as a software-first interface between custom hardware and custom motion control applications, allowing systems developers the freedom to write their own glue code..

3.3 Time Synchronization

Each trajectory segment is precisely time-stamped with a start and end time (in microseconds), and data within trajectory segments are all delineated on the basis of time; each segment is essentially just some function that describes what the machine's state should be at a given time, $x = f(t)$. By synchronizing trajectory readers' clocks with the trajectory author's clock, we can guarantee that, at any given microsecond, every participant in the system knows what they are responsible for doing.

However, MAXL does not specify network architectures. It uses abstracted networking links implemented in the Modular Things framework [Read et al. 2023] in order to communicate with remote devices. We think that this aspect is valuable, as it means that the system will remain extensible even if machine builders choose to deploy devices on network architectures that are unknown to us at the moment. But, it means that we (MAXL's authors) do not know

Table 1: MAXL's minimal buffering scheme tries to maintain a buffer size that minimizes the number of locked segments while maintaining that remote buffers are not starved.

Criteria	Value	Comments
Minimal Buffer Size	2	Unless the trajectory is coming to an end, we maintain at least two segments in remote buffers: the currently operating segment, plus one (for grace).
Minimal Buffer Time (s)	$2 * RTT$	MAXL measures the average packet round trip time (RTT) to each reader when it starts up; we can use this measurement to ensure that remote buffers are long enough such that when we transmit a new segment, it will arrive before the previous segments have completed.
Maximum Buffer Size	64	Trajectory readers, being typically deployed on micro-controllers with limited memory, have limited buffer depths; we have set this limit at 64 although it is easily re-configurable.

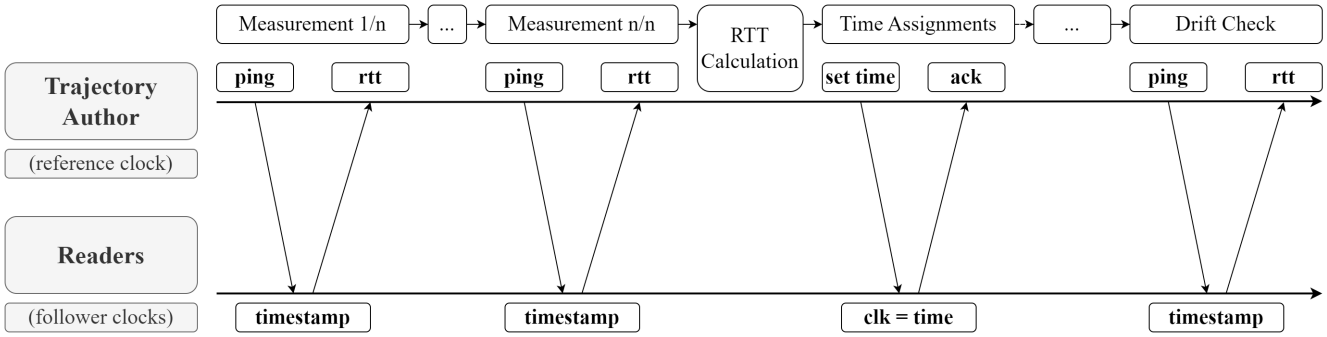

Figure 5: In order to synchronize trajectory readers' clocks to the author's clock, MAXL deploys a simplified type of network time protocol. MAXL starts up by measuring a number of round-trip-time (RTT) samples from remote clocks and calculating the average half-round-trip time. It then formulates one time assignment message per device that sets the device's clock to one half-round-trip time in the future, such that when the packet arrives at the remote device, the message is accurately stamped with the trajectory author's current time. MAXL can then occasionally check remote clock time-stamps in order to monitor for drift. Average RTT times are additionally used to inform MAXL's minimal buffering algorithm as discussed in Section 3.2

exactly the timing properties of the networks that MAXL will be deployed on.

In order to synchronize clocks on unknown networks, we implemented a simple network time protocol that is diagrammed in Figure 5. At startup, MAXL makes a series of measurements using a ping packet, estimating the average round-trip-time to each trajectory reader. It uses that information to align trajectory reader clocks with the trajectory author. One key difference between our network time protocol and *the* Network Time Protocol [Mills 1991] is that ours is asymmetric: the trajectory author is the sole clock source.

3.4 Tracks, APIs, Transforms and Configurations

In order to organize trajectories, we further decompose them into a series of *tracks*. For example, in a laser cutting machine, the machine's trajectory is encoded into five tracks: x, y, and z positions, one velocity track, and one event track that encodes laser on/off requests.

Positional and velocity tracks are generated by MAXL by default and are available regardless of configuration. To name them, users can provide a list of positional track names (string identifiers) when they configure the system. Machine builders can also provide transform functions that map machine positions to actuator positions, for machine configurations like CoreXY [Moyer 2012]. In the cases where transforms are provided, actuator- and cartesian-space positional tracks are both made available to trajectory readers. These configurations are handed to MAXL when it is instantiated as a software object. In Figure 1 we show one example configuration.

Event tracks are time-series step functions: they encode as a list of time stamps with a value at each stamp. An example of this type of track is figured alongside an example of the velocity track in Figure 4, and we show an example of how they are defined, using evaluators, in Figure 5. They are calculated after a segment's motion profile has been optimized, but before the segment is transmitted. Users provide a callback for the segment that, given the current machine state, returns a desired value for the event track at that time. In many cases event tracks are likely to be very simple: for example just switching a device "on" or "off"—but others (like our light-painting example) are more complex.

Listing 1 also shows a typical *subscriptions* data structure, which is the semantic that MAXL uses to denote which device will be responsible for carrying out each track in the trajectory. Each object in the subscription data structure names a device (string identifiers

are akin to network addresses in the Modular Things framework), a track that it should be subscribed to, and a callback function that it should use to read that track (the "reader").

Embedded devices are pre-configured to read particular types of tracks. For example, our modular stepper motor contains a firmware (a snippet of which is pictured in Listing 3, at left), that reads a positional track using a reader-callback labelled "stepper". However, it is only subscribed to a *particular* positional track (i.e. "x" or "y") when it is configured by a trajectory author. Our laser module (code snippets in the same figure, at right), defines two track readers: one for an event track that defines laser power, and another that reads the velocity track. It uses the combination of these two tracks to appropriately write a laser power output (which needs to scale along with speed).

The embedded APIs may seem like a small detail in the system, but they are core to the design pattern. In combination with Modular Things' abstracted networking layers, the generalized embedded APIs allow us to deploy MAXL on just about any embedded platform available in the Arduino ecosystem, using small pieces of interface code to bring new devices into MAXL systems.

```
await maxl.addSegmentToQueue({
  endPosition: [210, 0],
  maxVelocity: 250,
  eventChannels: [{
    name: "neopixelBitmap",
    evaluationPrecision: 10, // in milliseconds
    evaluationFunction: (states) => {
      let xpos = states.unitX * states.dist + states.p1[0];
      return evaluator(xpos, bitmapHello)
    }
  }]
})
```

Listing 5: Event channel tracks can be authored using evaluator callbacks, showing code from our light-painting demo from Figure 1. These callbacks are evaluated at variable timing precisions: callbacks are given the trajectory's full state at a given time, and return the desired value for the event channel, given those states. It is also possible for users to simply state a fixed event channel value for the entirety of the segment.

4 DEMONSTRATIONS

As an evaluation of our system, we provide demonstrations that show key attributes of our implementation [Ledo et al. 2018]. Firstly, both demonstrations were deployed on a machine where each motor, as well as output and input devices, were controlled via a modular circuit (some of which are photographed in Figure 6). This shows that MAXL is capable of coordinating motion across modular hardware. We think it is worth noting that this does not preclude MAXL from deployment on monolithic control boards like those used in most off-the-shelf 3D printers, since one device can subscribe to many tracks.

The light-painting demonstration showcases the flexibility of MAXL's event channels to coordinate the action of output devices and the use of abstract track types to interface with application-specific hardware. The second demo showcases the simple utility of being able to readily distribute a synchronized clock across modular input devices; we deployed an accelerometer on our machine and used time-synchronized readings to match planned accelerations with measured accelerations.

4.1 Light Painting

To demonstrate time-aligned output and the use of MAXL's event channels, we put together a light-painting demonstration whose output is the teaser figure of this paper, Figure 1. In this case we deployed a small strip of *neopixels*, which are individually addressable LEDs, on an end-effector. The firmware that runs the neopixels was deployed as a MAXL device that reads an 8-bit wide event track where bit values were mapped to LED states. We wrote bitmaps for "HELLO" and "WORLD" in JavaScript, and used an event track evaluator that wrote LED states to the track that corresponded to the machines' anticipated position within each word.

We think that this is a compelling demo because it shows clearly the flexibility of MAXL's event channels, and it maps cleanly onto motion control tasks that are typically difficult to orchestrate like jet-based printing and laser engraving.

4.2 Time-Aligned Data Retrieval

Our second demonstration deploys an accelerometer (a BNO055 chip) on our modular machine (Figure 6). We were curious to compare real-world accelerations (which are subject to vibrations and other machine realities like stretching belts and flexing components) with MAXL's planned trajectory. To do so, we were able to capture accelerometer data that was time-stamped at the source using MAXL's distributed clock, while also recording the acceleration trace from segments as they were planned. We combined these data to render the plot in Figure 7.

This demo serves partially to demonstrate that MAXL is able to actually execute motion on a distributed system, but also to showcase the simple utility of being able to generate time-aligned data traces from modular sensors. This demo is perhaps most compelling to machine builders who want to measure and characterize their machines. For example, some researchers have developed 3D printer hardware that can measure nozzle pressure in real-time [Coogan and Kazmer 2019], but other motion controllers do not provide a framework for aligning these measurements with the rest of the machines' state at the time of measurement: the extruders' flowrate and the machine's speed and position. MAXL provides the double utility of a distributed clock to time-stamp measurements from modular sensors, and a copy of the as-optimized trajectory to machine applications, such that trajectories and sensor data can be re-combined in data sets for offline analysis.

4.3 Plotter

Modular machine systems can be helpful in the context of machine prototyping environments, novel systems, and experimental machine designs, but modularity tends to increase overall systems cost and complexity. As a result, many machines are likely to go to market using monolithic controllers, or some mixture of monolithic main-boards with modular tool heads (for example).

Although we developed MAXL mostly for use in experimental and prototype machines, we wanted to show that it can be applied even in simple machines with monolithic controllers, and so we assembled a prototype of a *Blot* pen-plotter [HackClub 2023] and quickly deployed it as an instance of MAXL. This involved about 40 lines of new code in the firmware to glue the MAXL API to the plotter's existing hardware drivers, and some small changes to an

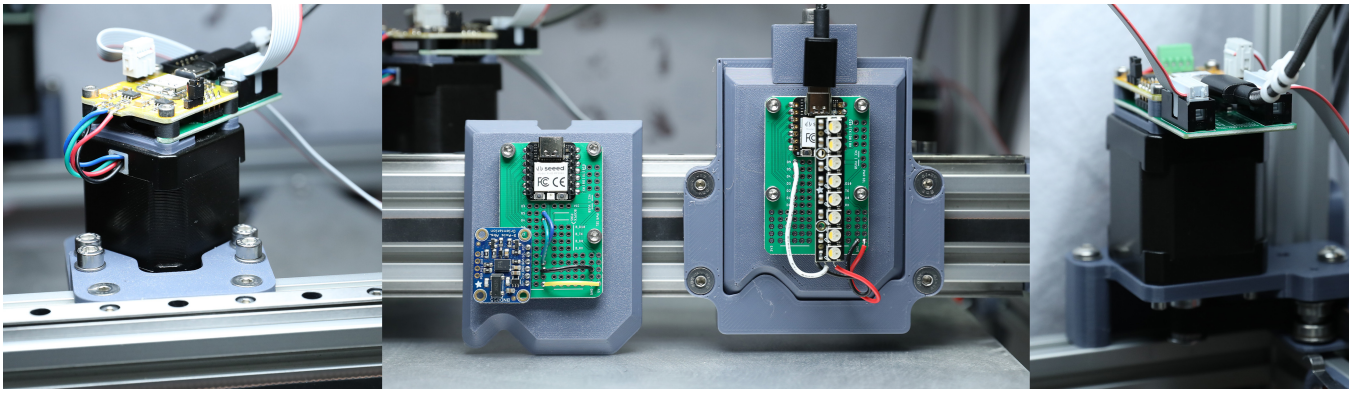

Figure 6: In order to test MAXL, we built a small modular machine using stepper-motor drivers from the Modular Things system controlling X (left, actuated using one motor) and Y (at right, which we controlled with two motors) axes, and developed two new end effectors for data retrieval (an accelerometer, center-left, shown dismounted from the machine) and data output (our light-painting device, center-right).

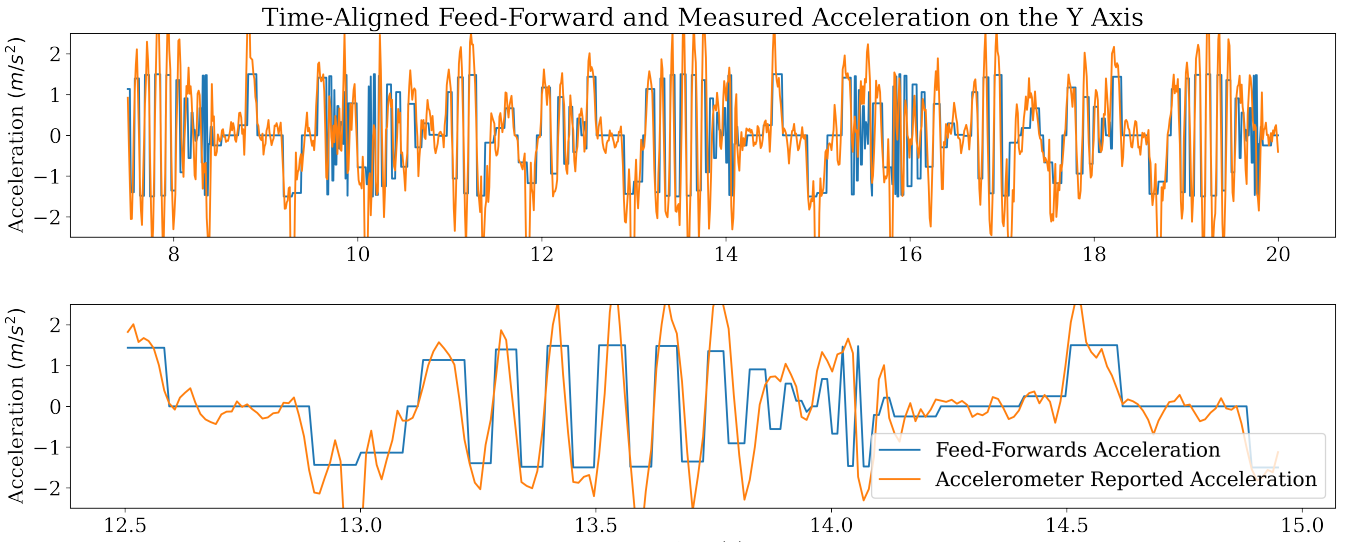

Figure 7: Here we show data traces from our time-aligned data retrieval demo that used hardware shown in Figure 6. MAXL makes post-optimization trajectories available to applications, and we used this feature to plot the reported acceleration trace (in blue). Using the distributed clock, we time-stamped accelerometer readings (in orange), and can render the two traces overlaid on one another. The point here is not to show alignment between feed-forward and sensed acceleration plots (we know that belt stretch and other vibrations contribute to error in this regard), but is meant to convey the utility of using a distributed clock to accurately time-stamp sensor readings, and compare them with as-optimized trajectories.

existing machine interface code that we use to send SVGs to the machine. In Figure 8, we show the machine plotting a test file.

5 DISCUSSION AND FUTURE WORK

In this short paper, we have explained how MAXL uses a distributed trajectory to organize modular execution of motion trajectories. We hope that the reader can see how the system could be extended to build many other computational fabrication processes, from laser cutters to FDM 3D printers, CNC mills, pick-and-place machines, and combinations thereof.

However, our ultimate goal is for members of the computational fabrication research community to extend MAXL in ways that we have not anticipated, using the generalized structures that we have deployed here. In particular, we are motivated by the notion that MAXL, or design patterns like it, could help computational fabrication researchers collaborate more productively by providing a framework within which new motor controllers, sensors, and other devices (like extruders, spindles, cameras, etc.) can be integrated into novel processes. At the moment, many such custom systems rely on outdated GCode-based interpreters that provide

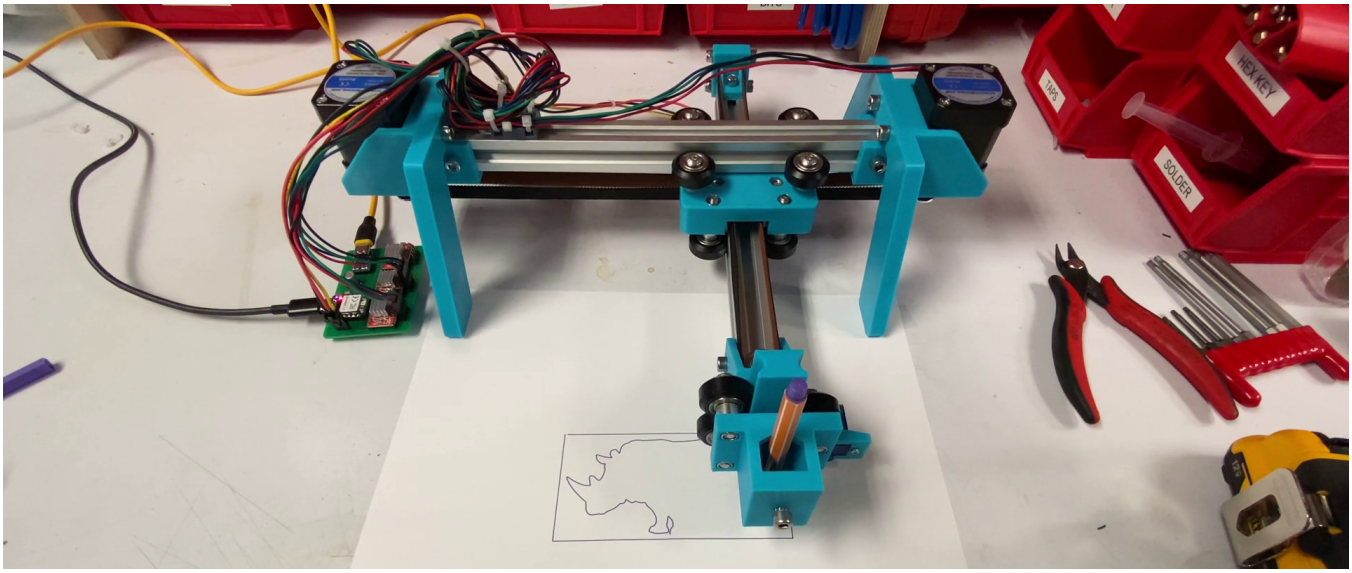

Figure 8: To demonstrate that MAXL can be deployed to modular *or* monolithic controllers, we configured an instance on this prototype *Blot* [HackClub 2023] machine.

limited access to optimized motion paths and limited surface area for modification—especially where new hardware is needed.

Furthermore, we hope that MAXL's treatment of motion control as a software interface (rather than a serialized interface like GCode) may enable researchers to more rapidly develop feedback-based, interactive computational fabrication processes (e.g., [Fossdal et al. 2021; Kim et al. 2017; Peng et al. 2018; Roumen et al. 2016; Willis et al. 2010]), or exploratory workflows ([Devendorf and Ryokai 2015; Tran O'Leary et al. 2023]). MAXL's minimal buffering routine, discussed in Section 3.2, is developed with near-real-time trajectory modification in mind, and our test system averaged 2 millisecond round-trip-time, meaning that practical minimal buffers only need to be a few milliseconds long.

With that said, MAXL is clearly a first step in this direction and not a be-all end-all solution for motion control. In future work, we anticipate following a few paths. First off, our underlying motion segments (simple linear moves) are limiting for high quality motion. We are developing an improved set of segments including arcs and bezier representations, as we think these will improve the overall quality of motion as well as help compress complex trajectories. For example, spiral and circular profiles that are currently segmented into hundreds of linear segments could be instead represented directly as arcs, greatly reducing the system's bandwidth requirement. Second, our speed optimizer is based on constant acceleration, where it is well known that constant-jerk optimizations produce higher quality motion. We are exploring the integration of this style planner into MAXL, as well as exploring the use of complete state-space dynamics representations [Rowell 2002] of machine systems in order to further optimize trajectories. We hope that this exploration will also enable us to plan for more kinematically complex systems that we know are of interest to the computational fabrication community, like robot arms and delta printers. Third, while MAXL can be configured for many different processes, we have yet to develop semantics for on-the-fly machine changes like those that arise from tool-changing CNC machines [Vasquez et al. 2020]. For example, we would like to combine FDM printing, CNC milling and laser-etching into one workflow, but have not considered how MAXL will adapt to changing underlying hardware when tools are swapped out mid-process. We do hope that the software-based nature of the system will make experimentation in this regard straightforward.

6 CONCLUSION

This paper introduces MAXL (Modular Acceleration eXecution Library): a modular and extensible motion control system for digital fabrication applications. MAXL uses a time-synchronized distributed trajectory object in order to coordinate modular devices that make up a machine. For example, a digital fabrication application such as a CAM program authors a distributed trajectory object, and a machine's hardware modules, such as a 3D printer's stepper motors and extruder, read their trajectories from the distributed object. MAXL provides a novel interface between digital fabrication applications and machines, enabling low-level and interactive control of each machine module. Authors and readers can rapidly update and read the distributed trajectory object, enabling responsive and interactive motion. We demonstrate the benefits of this approach in several example implementations. The examples show that MAXL enables high-quality time-synchronous coordination across diverse hardware modules. Ultimately, we argue that MAXL is a step towards making machine control design simpler and more extensible, enabling computational fabrication systems researchers to more readily build and re-use modular motion components as they develop novel systems and machines.

REFERENCES

Byoungkwon An, Ye Tao, Jianzhe Gu, Tingyu Cheng, Xiang 'Anthony' Chen, Xiaoxiao Zhang, Wei Zhao, Youngwook Do, Shigeo Takahashi, Hsiang-Yun Wu, Teng Zhang, and Lining Yao. 2018. Thermorph: Democratizing 4D Printing of Self-Folding Materials and Interfaces. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173834

Arduino LLC. 2023. Arduino Boards, Modules, Shields, and Kits. https://www.arduino.cc/, accessed July 2023.

Shajay Bhooshan, Tom Van Mele, and Philippe Block. 2020. Morph & Slerp: Shape Description for 3D Printing of Concrete. In *Proceedings of the 5th Annual ACM Symposium on Computational Fabrication* (Virtual Event, USA) *(SCF '20)*. Association for Computing Machinery, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/3424630.3425413

Timothy J Coogan and David O Kazmer. 2019. In-line rheological monitoring of fused deposition modeling. *Journal of Rheology* 63, 1 (2019), 141–155.

Laura Devendorf and Kimiko Ryokai. 2015. Being the Machine: Reconfiguring Agency and Control in Hybrid Fabrication. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 2477–2486. https://doi.org/10.1145/2702123.2702547

Jack Forman, Mustafa Doga Dogan, Hamilton Forsythe, and Hiroshi Ishii. 2020. DefeXtiles: 3D Printing Quasi-Woven Fabric via Under-Extrusion. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 1222–1233. https://doi.org/10.1145/3379337.3415876

Frikk Fossdal, Rogardt Heldal, and Nadya Peek. 2021. Interactive Digital Fabrication Machine Control Directly Within a CAD Environment. In *Proceedings of the 6th Annual ACM Symposium on Computational Fabrication* (Virtual Event, USA) *(SCF '21)*. Association for Computing Machinery, New York, NY, USA, Article 8, 15 pages. https://doi.org/10.1145/3485114.3485120

Frikk H. Fossdal, Jens Dyvik, Jakob Anders Nilsson, Jon Nordby, Torbjørn Nordvik Helgesen, Rogardt Heldal, and Nadya Peek. 2020. Fabricatable Machines: A Toolkit for Building Digital Fabrication Machines. In *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction* (Sydney NSW, Australia) *(TEI '20)*. Association for Computing Machinery, New York, NY, USA, 411–422. https://doi.org/10.1145/3374920.3374929

GRBL. 2023. *An open source, embedded, high performance g-code-parser and CNC milling controller written in optimized C that will run on a straight Arduino.* https://github.com/grbl/grbl

HackClub. 2023. Blog, the plotting bot from hack club. https://github.com/hackclub/blot.

Alexandra Ion, Johannes Frohnhofen, Ludwig Wall, Robert Kovacs, Mirela Alistar, Jack Lindsay, Pedro Lopes, Hsiang-Ting Chen, and Patrick Baudisch. 2016. Metamaterial Mechanisms. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 529–539. https://doi.org/10.1145/2984511.2984540

Matthew Keeter. 2013. Hierarchical Volumetric Object Representations for Digital Fabrication Workflows. In *ACM SIGGRAPH 2013 Posters* (Anaheim, California) *(SIGGRAPH '13)*. Association for Computing Machinery, New York, NY, USA, Article 84, 1 pages. https://doi.org/10.1145/2503385.2503477

Jeeeun Kim, Haruki Takahashi, Homei Miyashita, Michelle Annett, and Tom Yeh. 2017. Machines as Co-Designers: A Fiction on the Future of Human-Fabrication Machine Interaction. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI EA '17)*. Association for Computing Machinery, New York, NY, USA, 790–805. https://doi.org/10.1145/3027063.3052763

Jeeeun Kim, Clement Zheng, Haruki Takahashi, Mark D Gross, Daniel Ashbrook, and Tom Yeh. 2018. Compositional 3D Printing: Expanding & Supporting Workflows towards Continuous Fabrication. In *Proceedings of the 2nd Annual ACM Symposium on Computational Fabrication* (Cambridge, Massachusetts) *(SCF '18)*. Association for Computing Machinery, New York, NY, USA, Article 5, 10 pages. https://doi.org/10.1145/3213512.3213518

Klipper3D. 2023. *Klipper: 3d-Printer firmware.* https://www.klipper3d.org/

Sophie Landwehr Sydow, Martin Jonsson, and Jakob Tholander. 2022. Modding the Pliable Machine: Unpacking the Creative and Social Practice of Upkeep at the Makerspace. In *Proceedings of the 14th Conference on Creativity and Cognition* (Venice, Italy) *(C&C '22)*. Association for Computing Machinery, New York, NY, USA, 220–233. https://doi.org/10.1145/3527927.3532804

Maria Larsson, Hironori Yoshida, and Takeo Igarashi. 2019. Human-in-the-Loop Fabrication of 3D Surfaces with Natural Tree Branches. In *Proceedings of the 3rd Annual ACM Symposium on Computational Fabrication* (Pittsburgh, Pennsylvania) *(SCF '19)*. Association for Computing Machinery, New York, NY, USA, Article 1, 12 pages. https://doi.org/10.1145/3328939.3329000

David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3173574.3173610

Zhao Ma, Simon Duenser, Christian Schumacher, Romana Rust, Moritz Bächer, Fabio Gramazio, Matthias Kohler, and Stelian Coros. 2020. RobotSculptor: Artist-Directed Robotic Sculpting of Clay. In *Proceedings of the 5th Annual ACM Symposium on Computational Fabrication* (Virtual Event, USA) *(SCF '20)*. Association for Computing Machinery, New York, NY, USA, Article 13, 12 pages. https://doi.org/10.1145/3424630.3425415

David L Mills. 1991. Internet time synchronization: the network time protocol. *IEEE Transactions on communications* 39, 10 (1991), 1482–1493.

Ilan E Moyer. 2012. Core xy.

Chandrakana Nandi, James R. Wilcox, Pavel Panchekha, Taylor Blau, Dan Grossman, and Zachary Tatlock. 2018. Functional Programming for Compiling and Decompiling Computer-Aided Design. *Proc. ACM Program. Lang.* 2, ICFP, Article 99 (jul 2018), 31 pages. https://doi.org/10.1145/3236794

Nadya Peek, James Coleman, Ilan Moyer, and Neil Gershenfeld. 2017. Cardboard Machine Kit: Modules for the Rapid Prototyping of Rapid Prototyping Machines. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 3657–3668. https://doi.org/10.1145/3025453.3025491

Huaishu Peng, Jimmy Briggs, Cheng-Yao Wang, Kevin Guo, Joseph Kider, Stefanie Mueller, Patrick Baudisch, and François Guimbretière. 2018. RoMA: Interactive Fabrication with Augmented Reality and a Robotic 3D Printer. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3174153

Jake Robert Read, Leo Mcelroy, Quentin Bolsee, B Smith, and Neil Gershenfeld. 2023. Modular-Things: Plug-and-Play with Virtualized Hardware. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, Article 210, 6 pages. https://doi.org/10.1145/3544549.3585642

Replicape. 2023. *Replicape - a smart, silent and user friendly electronics controller board for 3D-printers and CNC machines.* https://www.thing-printer.com/product/replicape/

Michael L. Rivera, S. Sandra Bae, and Scott E. Hudson. 2023. Designing a Sustainable Material for 3D Printing with Spent Coffee Grounds. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference* (Pittsburgh, PA, USA) *(DIS '23)*. Association for Computing Machinery, New York, NY, USA, 294–311. https://doi.org/10.1145/3563657.3595983

Michael L. Rivera and Scott E. Hudson. 2019. Desktop Electrospinning: A Single Extruder 3D Printer for Producing Rigid Plastic and Electrospun Textiles. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300434

Thijs Roumen, Bastian Kruck, Tobias Dürschmid, Tobias Nack, and Patrick Baudisch. 2016. Mobile Fabrication. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/2984511.2984586

Derek Rowell. 2002. State-space representation of LTI systems. *URL: http://web. mit. edu/2.14/www/Handouts/StateSpace. pdf* (2002), 1–18.

Smoothieware. 2023. *smoothieboard.* http://smoothieware.org/smoothieboard

Rundong Tian, Vedant Saran, Mareike Kritzler, Florian Michahelles, and Eric Paulos. 2019. Turn-by-wire: Computationally mediated physical fabrication. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 713–725.

Rundong Tian, Sarah Sterman, Ethan Chiou, Jeremy Warner, and Eric Paulos. 2018. Matchsticks: Woodworking through improvisational digital fabrication. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

Iremnur Tokac, Benay Gursoy, Herman Bruyninckx, and Andrew Vande Moere. 2022. Craft-Inspired Digital Fabrication: A Study of Interactive Robotic Clay Carving. In *Proceedings of the 7th Annual ACM Symposium on Computational Fabrication* (Seattle, WA, USA) *(SCF '22)*. Association for Computing Machinery, New York, NY, USA, Article 2, 14 pages. https://doi.org/10.1145/3559400.3562003

Jasper Tran O'Leary, Gabrielle Benabdallah, and Nadya Peek. 2023. Imprimer: Computational Notebooks for CNC Milling. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 207, 15 pages. https://doi.org/10.1145/3544548.3581334

Jasper Tran O'Leary, Eunice Jun, and Nadya Peek. 2022. Improving Programming for Exploratory Digital Fabrication with Inline Machine Control and Styled Toolpath Visualizations. In *Proceedings of the 7th Annual ACM Symposium on Computational Fabrication* (Seattle, WA, USA) *(SCF '22)*. Association for Computing Machinery, New York, NY, USA, Article 8, 12 pages. https://doi.org/10.1145/3559400.3561998

Thibault Tricard, Vincent Tavernier, Cédric Zanni, Jonàs Martínez, Pierre-Alexandre Hugron, Fabrice Neyret, and Sylvain Lefebvre. 2020. Freely Orientable Microstructures for Designing Deformable 3D Prints. *ACM Trans. Graph.* 39, 6, Article 211 (nov 2020), 16 pages. https://doi.org/10.1145/3414685.3417790

Joshua Vasquez, Hannah Twigg-Smith, Jasper Tran O'Leary, and Nadya Peek. 2020. Jubilee: An Extensible Machine for Multi-Tool Fabrication. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376425

Guanyun Wang, Humphrey Yang, Zeyu Yan, Nurcan Gecer Ulu, Ye Tao, Jianzhe Gu, Levent Burak Kara, and Lining Yao. 2018. 4DMesh: 4D Printing Morphing Non-Developable Mesh Surfaces. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 623–635. https://doi.org/10.1145/3242587.3242625

Guanyun Wang, Lining Yao, Wen Wang, Jifei Ou, Chin-Yi Cheng, and Hiroshi Ishii. 2016. XPrint: A Modularized Liquid Printer for Smart Materials Deposition. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 5743–5752. https://doi.org/10.1145/2858036.2858281

Karl D.D. Willis, Cheng Xu, Kuan-Ju Wu, Golan Levin, and Mark D. Gross. 2010. Interactive Fabrication: New Interfaces for Digital Fabrication. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction* (Funchal, Portugal) *(TEI '11)*. Association for Computing Machinery, New York, NY, USA, 69–72. https://doi.org/10.1145/1935701.1935716