






Plaquette: An Object-Oriented Framework for Embedded Signal Processing in Interactive Media

Sofian Audry ^{1*} and Thomas Ouellet Fredericks ^{2*}

1 Université du Québec à Montréal, Canada  2 Collège Montmorency, Canada  * These authors contributed equally.

DOI: [10.21105/joss.09144](https://doi.org/10.21105/joss.09144)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Daniel S. Katz](#) 

Reviewers:

- [@WarmCyan](#)
- [@timonmerk](#)

Submitted: 30 September 2025

Published: 10 November 2025

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Plaquette is an object-oriented C++ framework for interactive media on embedded systems, supporting a wide range of platforms including AVR, ARM, ESP32, SAMD, and STM32. It provides a signal-centric architecture and a suite of modular abstractions (oscillators, filters, units, and scheduling engines) that simplify the design of time-based behaviors. Its expressive syntax allows fast prototyping with multiple sensors, actuators, and real-time processes, enabling researchers as well as creative practitioners to experiment with and design complex physical computing systems.

Beyond its technical contributions, *Plaquette* serves as a bridge between scientific research and creative practice. Its application to interdisciplinary projects involving affective biofeedback and robotic behaviors demonstrates how the framework's flexible and robust infrastructure supports creativity and experimentation across interactive media.

Statement of Need

Plaquette is designed for research—specifically, practice-based research ([Candy & Edmonds, 2018](#)) and research-creation ([Loveless, 2019](#))—in embedded interactive media, with applications ranging from affective computing and digital lutherie to robotic art, interactive installation, connected objects, and performance. The [Arduino](#) ([Banzi & Shiloh, 2022](#)) open-source platform for physical computing anchors a large and active ecosystem; it remains the go-to environment for artists, makers, and researchers working with embedded interactive media. Its accessibility, community, libraries, and hardware options have made it the most popular microcontroller platform worldwide. Yet, Arduino's core library is not optimized for real-time signal processing: it lacks an object-oriented design, provides limited abstractions for managing concurrent events, and often requires manipulating raw numerical values, making interaction design unintuitive and hindering expressive experimentation.

In contrast, dataflow software popular within scientific and creative communities working with real-time media (such as Pure Data, Max, and TouchDesigner) provide powerful models for composing with signals, but are too memory- and CPU-intensive to run on constrained hardware. Similarly, scientific tools such as Python's NumPy/SciPy, Matlab, or R, offer rich signal analysis tools, but are not designed for real-time processing on embedded devices.

Several Arduino libraries specialize in particular aspects of real-time signal processing such as filtering (e.g., [arduinoFFT](#), [DataTome](#), [FIR-filter](#)), generating waveforms (e.g., [FunctionGenerator](#), [SyncWaveformsLib](#)), or timing (e.g., [arduino-timer](#), [Chrono](#), [SoftTimer](#)). While these tools are effective within their domains, they address isolated functionalities and do not share a common programming model. Finally, Arduino's hardware timers provide precise low-level control but require complex configuration of prescalers, registers, and interrupt handlers. These

limitations of existing Arduino libraries and hardware timers hamper creative expression and experimentation.

Plaquette addresses these gaps by bringing the expressive power of dataflow signal-based programming into a lightweight object-oriented framework optimized for microcontrollers (i.e., requiring minimal CPU and memory usage). It enables intuitive handling of signals, providing efficient implementations of core data processing functions such as peak detection, normalization, scaling, and smoothing (low-pass), all under a unified framework. This enables researchers in art and science to focus on experimentation and expressivity while ensuring accurate and reliable real-time performance on resource-constrained platforms. Additional functionalities that expand on *Plaquette*'s core, such as advanced data filtering and event management, can be implemented as external *Plaquette* libraries.

The framework provides a strong foundation for workshop-based research-creation projects, where participants often have diverse levels of technical skills. Its accessibility ensures that beginners can quickly grasp and apply core concepts, while its efficient, expressive, and extensible architecture supports the needs of advanced users. This makes it particularly well-suited for collaborative prototyping in media arts, design, and human-computer interaction, where embodied and situated practices require adaptable tools.

Plaquette has already supported a number of public research projects. It was used to improve real-time physiological signal processing as part of the [BioData](#) library for affective biofeedback, supporting creative applications in music and performance ([Gee, 2023](#)), and in studies of electrodermal activity ([Hagler et al., 2022](#)). It was integrated at the core of the [MisBKit](#), a robotic kit enabling research on object behaviors ([Bianchini et al., 2015](#)). It was also employed for signal processing and robotic expression in *Morphosis*, an installation featuring three spheroid robots that learn in real-time using reinforcement learning ([Audry et al., 2020](#); [Audry, 2023](#)). These examples illustrate the framework's role not only as a technical tool but also as a catalyst for interdisciplinary research.

Functionality and Design Overview

The core of *Plaquette* is organized around two interdependent abstractions called *units* and *engines* that provide a coherent structure for building complex, real-time interactive systems on microcontrollers. *Units* are modular building blocks that encapsulate behaviors such as sensing, generating, filtering, or actuating. *Engines* operate as conductors, managing initialization and timing of units so that they execute consistently without blocking or interruptions.

All units implement a unified interface consisting of a single input and a single output function. This design makes it possible to chain units together in a dataflow-like manner using a special operator (`>>`), where the output of one unit is sent as input to another. This signal-centric approach allows developers to work with flows of information rather than low-level procedural code.

The framework includes a set of core unit types:

- **Base units:** basic analog and binary inputs and outputs
- **Generators:** generative source signals such as square, triangle, and sine waves, as well as ramps
- **Timing units:** scheduling and temporal control units such as timers and metronomes
- **Filters:** real-time signal transformations such as min-max scaling, normalizing, and detecting peaks
- **Fields:** spatial functions sampled at fractional positions to plot, shape, or transform signals across space

Engines and units have a low memory footprint, with static allocation at compile time that prevents dynamic allocation and fragmentation. In particular, signal-processing units such

as min-max scaling and normalization use exponential moving averages rather than circular buffers, ensuring low and predictable memory usage.

Examples

The following program chains an analog input through a min-max scaling filter to bring it to full range, then uses the input value to influence the period of oscillation of an LED using a sine wave.

```
#include <Plaquette.h>

AnalogIn input{A0};           // analog input on pin A0
AnalogOut led{9};             // PWM-controlled LED on pin 9
MinMaxScaler scaler{};        // min-max scaler
Wave oscillator{SINE};         // sine wave

void begin() {}

void step() {
  input >> scaler;             // rescale input to full range [0, 1]
  oscillator.period(scaler.mapTo(1, 10)); // set period from 1 to 10 seconds
  oscillator >> led;           // send oscillator value to LED
}
```

This program reacts to peaks in the incoming signal by triggering a sudden movement (ramp) in a servo motor. The peak detector triggers in response to outliers after signal normalization, using an event callback to start the ramp. The normalization is calibrated over a sliding time window, smoothly re-calibrating itself in response to changes in the input signal over time.

```
#include <Plaquette.h>

AnalogIn input{A0};           // analog input on pin A0
ServoOut servo{9};            // servomotor connected on pin 9
Normalizer normalizer{0, 1};   // normalizes to N(0, 1)
PeakDetector peak{1.5};        // detects outliers at 1.5 times stddev
Ramp ramp{2.0};                // ramp with 2 seconds duration

void begin() {
  normalizer.timeWindow(60);    // 60 seconds calibration sliding time window
  peak.onBang([](){ ramp.start(); }); // on peak detection: restart ramp
}

void step() {
  input >> normalizer >> peak; // chain-process input signal
  ramp >> servo;               // send ramp value to servo motor
}
```

Acknowledgements

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada, the Social Sciences and Humanities Research Council of Canada, the Fonds de Recherche du Québec — Société et Culture, the Canada Council for the Arts, MITACS, and the Society for Arts and Technology. We thank our collaborators and colleagues for supporting the project, in particular Luana Belinsky, Marianne Fournier, Erin Gee, Matthew Loewen, and Chris Salter.

References

- Audry, S. (2023). Choreomata. In R. A. Trillo & M. Poliks, *Choreomata* (1st ed., pp. 283–307). Chapman and Hall/CRC. <https://doi.org/10.1201/9781003312338-16>
- Audry, S., Dumont-Gagné, R., & Scurto, H. (2020, December). Behaviour aesthetics of reinforcement learning in a robotic art installation. *4th NeurIPS Workshop on Machine Learning for Creativity and Design*. <https://hal.archives-ouvertes.fr/hal-03100907>
- Banzi, M., & Shiloh, M. (2022). *Getting started with Arduino: The open source electronics prototyping platform*. Make Community, LLC. ISBN: 978-1-68045-693-6
- Bianchini, S., Bourganel, R., Quinz, E., Levillain, F., & Zibetti, E. (2015). MisBehavioral objects. In D. Bihanic (Ed.), *Empowering users through design: Interdisciplinary studies and combined approaches for technological products and services* (pp. 129–152). Springer International Publishing. https://doi.org/10.1007/978-3-319-13018-7_8
- Candy, L., & Edmonds, E. (2018). Practice-based research in the creative arts: Foundations and futures from the front line. *Leonardo*, 51(1, 1), 63–69. https://doi.org/10.1162/LEON_a_01471
- Gee, E. M. (2023). *The BioSynth—an affective biofeedback device grounded in feminist thought*. 479–485. <https://doi.org/10.5281/zenodo.11189254>
- Hagler, J., Kim, C., Kateb, P., Yeu, J., Gagnon-Lafrenais, N., Gee, E., Audry, S., & Cicoira, F. (2022). Flexible and stretchable printed conducting polymer devices for electrodermal activity measurements. *Flexible and Printed Electronics*, 7(1), 014008. <https://doi.org/10.1088/2058-8585/ac4d0f>
- Loveless, N. (2019). *How to make art at the end of the world: A manifesto for research-creation*. Duke University Press. <https://doi.org/10.1215/9781478004646>