

# L-BOW: GESTURE-DRIVEN DIGITAL AUDIO EFFECTS FOR AUGMENTED VIOLIN IN A UNIFIED CSOUND ENVIRONMENT

Jinlan Zhao and Richard Boulanger

Berklee College of Music  
Boston, MA, USA

jzhao9@berklee.edu, rboulanger@berklee.edu

## ABSTRACT

Live performance leaves little room for a sensor pipeline that misfires; when a gesture fails to map correctly to an intended effect, the error is immediately audible. This paper presents L-Bow, a wrist-worn six-degree-of-freedom (6-DoF) inertial measurement unit (IMU) controller for augmented violin performance. By removing intermediate software layers, L-Bow integrates gesture sensing, six performance modes, and a shared digital effects chain within a single, self-contained Csound file. This is achieved using Csound's native `arduinoRead` opcode for direct serial communication rather than an external Python–OSC bridge. The paper discusses the architecture of this system and its implications for designing dependable, low-maintenance interactive digital audio effects.

## 1. INTRODUCTION

Arduino is a staple platform for sensor-based musical interfaces. In typical hybrid systems, the microcontroller serves as a data-acquisition tool while real-time synthesis and digital effects processing are offloaded to a dedicated sound engine like Csound. A common communication pipeline utilizes a Python serial parser to decode the data and forward it to the sound engine via OSC over UDP [1]. Though flexible, this workflow introduces external runtimes, dependencies, and multiple points of failure during live performance.

Csound's `arduinoRead` opcode offers an integrated alternative, mapping Arduino sensor values directly to k-rate control signals [2]. This paper explores the implementation of `arduinoRead` within L-Bow, a custom wrist-mounted performance controller. As a classically trained violinist, the author sought to expand the sonic possibilities of traditional violin playing through human–machine interaction—making the relationship between live gesture and electronic response perceptible to audiences while preserving unmodified technique and portability. L-Bow consolidates sensor data reception, mode-dependent gesture mapping, six performance modes, and a shared effects (FX) chain within a single `.csd` file, providing a unified environment for interactive digital effects.

## 2. BACKGROUND AND RELATED WORK

Connecting physical sensors to Csound originally required managing low-level byte reading, data decoding, and potential thread-blocking within the orchestra code. Dedicated Arduino opcodes [2] bypassed these issues via an internal listener thread that synchronizes and decodes the incoming serial stream into directly usable k-rate variables.

Prior augmented string systems have successfully mapped complex violin gestures to sound: the Hyperbow [3] captured bow position, acceleration, and lateral strain; the K-Bow [4] provided a wireless bow measuring grip pressure, hair tension, and rotation; and the Hypercello [5] pioneered bow-dynamics sensing via electric field proximity between bow stick and bridge.

Wrist-mounted sensing for bowed strings was explored by Kimura et al. [6], who extracted expressive bow-hand gesture for interactive composition, and by Thorn's Alto.Glove [7], a nine-axis IMU wrist glove designed to avoid interfering with bow technique. These systems typically rely on external middleware or proprietary protocols to bridge hardware and DSP software. L-Bow similarly places sensing at the wrist but pairs this wearable design with a native Csound pipeline, requiring no physical modifications to the instrument or bow.

## 3. L-BOW SYSTEM OVERVIEW

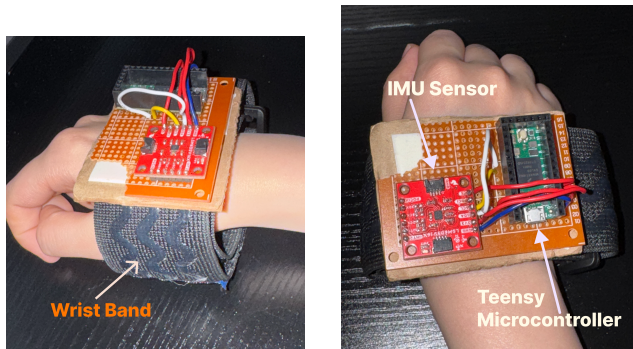
The L-Bow hardware is built around an LSM6DSV16X 6-DoF IMU embedded in an elastic wristband and connected via I<sup>2</sup>C to a Teensy 4.0 microcontroller (Figure 1). The Teensy transmits three accelerometer and three gyroscope axes to the performance computer over USB serial at 9600 baud. Signed physical-unit readings are mapped to 10-bit values on the microcontroller before transmission, making them directly readable via `arduinoRead`.

A serial-to-OSC bridge [1] requires a Python runtime, `pyserial` and `python-osc` packages, and OSC port configuration. Using `arduinoRead` eliminates all three, retaining only Arduino firmware, a compatible Csound build, and the correct serial device path—directly reducing the failure points between gesture and sound.

## 4. SYSTEM ARCHITECTURE AND DIGITAL AUDIO EFFECTS DESIGN

### 4.1. Global Architecture

L-Bow is organized as a self-contained Csound performance environment within a single `.csd` file providing six performance modes. Instrument 1 receives the six IMU channels through `arduinoRead`, smooths the values with `portk` (half-time



**Figure 1:** L-Bow wristband prototype with the LSM6DSV16X IMU.

0.05 s), scales them to approximate physical ranges, and stores the results in global k-rate variables `gkAX`–`gkGZ` available to all modes; the acquisition structure is shown in Listing 1. Instrument 2 routes live violin audio to the processing buses. Instrument 5 dispatches CsoundQt GUI controls and mode-dependent trigger events. Audio from the active mode is routed through shared `gaChainL/R` buses; Instruments 70–80 form a cascaded global effects chain; Instrument 90 applies a look-ahead peak limiter and soft-clip stage before stereo output; Instrument 99 clears all buses at the end of each control cycle. A SoftStep 3 MIDI foot controller sends note messages to Instrument 3, which deactivates idle instruments, resets mode-specific state, and recalls the corresponding effects preset. The complete signal flow is illustrated in Figure 2.

```
giport arduinoStart "/dev/cu.usbmodem", 9600
```

```
instr 1
  kAX arduinoRead giport, 1
  kGX arduinoRead giport, 4
  gkAX portk kAX, 0.05
  gkGX portk kGX, 0.05
  gkAX scale gkAX, 16, -16, 1023, 0
  gkGX scale gkGX, 2000, -2000, 1023, 0
endin
```

*Listing 1: Csound acquisition path (two of six axes shown). The device path is OS-specific; the same `arduinoRead-portk-scale` pattern repeats for all six axes.*

#### 4.2. Interactive Gesture-Driven Effects

Continuous and thresholded IMU data shape synthesis and digital audio effects across six performance modes, summarized in Table 1. The design follows the view that the mapping layer defines the expressive character of an electronic instrument [8]; continuous sensor values shape synthesis, filtering, spatialization, and spectral processing, while thresholded features generate discrete events.

**Call & Response:** Gyroscope magnitude triggers a discrete FM-synthesized bell-like resonant tone when a gesture exceeds a set threshold, while individual rotational axes shape timbral filtering.

**PVS Freeze:** Captures and sustains a spectral snapshot of the violin audio, deriving its stereo position from the wrist angle (AY) at the moment of creation.

**Granulator:** Maps gestures to real-time granular variables: buffer scan speed (GX), positional spray (AX), pan spread (GY), grain duration (AY), and pitch spread (AZ). Listing 2 shows the GX-to-scan-rate mapping; the GX stream is clamped at an empirical threshold of 450 (raw 10-bit channel value; range 0–1023), normalizing the practical bow-gesture range to [0, 1] without per-session calibration.

**Pitch Follower / Choir:** Tracks the live violin pitch and layers harmonized additions, using wrist gestures to alter harmonic brightness (GX), vibrato/chorus depth (GZ), and vowel crossfading (AY).

**Rhythm Gate:** Translates motion energy and spatial coordinates into gate sharpness, rhythmic subdivision rates, and sweeping stereo motion.

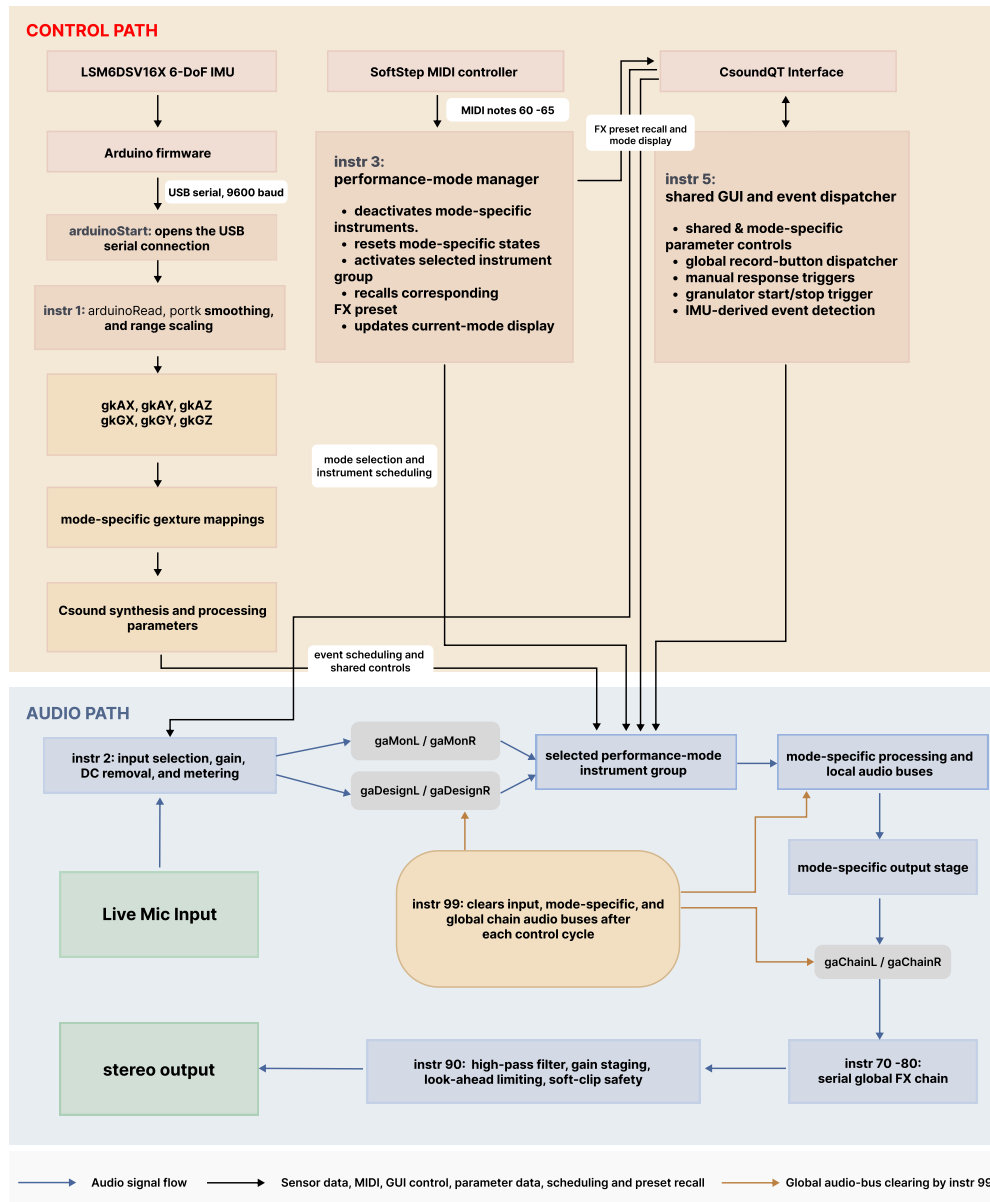
**Glitch:** Modulates an interactive loop buffer via continuous IMU streams, controlling loop density (GZ), chopping segment sizes (AZ), buffer pointer location (GX), and loop durations (AY).

```
; GX (wrist rotation) -> granular scan speed
; clamp at 450: bow-gesture ceiling
kSrc = limit(abs(gkGX), 0, 450)
kGXN = sqrt(kSrc / 450)
kGXN = limit(kGXN, 0, 1)
kScanRate = 0.002 + (kGXN * 0.08)
kScanRate portk kScanRate, 0.15
```

*Listing 2: GX-to-scan-rate mapping in Granulator mode. Square-root compression favours slow gestures; maximum scan speed is capped at 0.082.*

**Table 1:** IMU gesture mappings across the six performance modes.

| Mode            | IMU Input                                             | Musical Function                              |
|-----------------|-------------------------------------------------------|-----------------------------------------------|
| Call & Response | GX/GY/GZ magnitude → trigger; GX → freq.; GZ → filter | Timbral shaping; discrete sound layer trigger |
| PVS Freeze      | AY → pan at capture                                   | Stereo position from wrist angle              |
| Granulator      | GX scan; AX spray; GY pan; AY duration; AZ pitch      | Full granular parameter control               |
| Pitch Follower  | GX brightness; GZ vibrato; AY vowel                   | Harmonic and timbral layering                 |
| Rhythm Gate     | GX subdiv.; GZ energy; AY pan; accel. magnitude burst | Gate sharpness; stereo motion                 |
| Glitch          | GZ density; AZ chop; GX pointer; AY loop              | Loop buffer modulation                        |



**Figure 2:** L-Bow signal flow: sensor acquisition, mode dispatch, cascaded effects chain, and mastering output. Solid lines indicate audio; dashed lines indicate control and scheduling.

### 4.3. Global FX Chain and Preset Management

The cascaded digital effects chain features ten discrete processors assigned across Instruments 70–80: Distortion, EQ, Compressor, Wah, Flanger, Ring Modulation, Echo, Harmonizer, Shimmer, and Reverb. Audio from the active performance mode arrives on the shared `gaChainL/R` buses and passes through each active processor in series.

Preset management is handled through a global function table (`giPresets`) storing six rows of eleven binary (on/off) values; the columns correspond to the ten active processors plus one reserved slot. When the performer switches modes via the foot controller, Instrument 3 reads the corresponding row and writes the

values directly to the CsoundQt widgets, updating the entire effects state instantaneously with a single SoftStep 3 action. This design allows the performer-designer to define distinct sonic contexts per mode—for example, the Granulator preset activates Shimmer and Reverb but bypasses Distortion, while the Glitch preset enables Ring Modulation and Echo—without any manual widget interaction on stage.

## 5. DISCUSSION AND DESIGN REFLECTIONS

L-Bow demonstrates that Csound’s native `arduinoRead` infrastructure is fully capable of driving simultaneous multi-axis IMU streams, intricate gesture mappings, and a complex digital effects

chain without relying on middleware. A controlled trigger-latency test (20 trials, button input) showed zero false retriggers with `arduinoRead` versus three with a Python–OSC bridge, and a mean latency 22.5 ms lower for the native path. As a performer-designer who developed both the hardware and instrument architecture, the author fine-tuned mappings—smoothing coefficients, scaling factors, and threshold values—within the same `.csd` file, a workflow that proved especially productive during iterative design. Csound [9] was the natural choice: its opcode library spans granular synthesis, spectral processing, physical modeling, real-time sensor input, and effects processing, making it possible to realize six distinct performance modes, a cascaded effects chain, and automated preset management all within a single environment. Informal feedback from violinist colleagues confirmed that the system’s latency did not interfere with performance.

**Live Demonstration:** At DAFx26, L-Bow is presented as a live augmented violin performance. The author performs each of the six modes sequentially via the SoftStep 3 foot controller, allowing attendees to observe the direct coupling between bowing gesture and digital effect in real time. Required equipment: violin with pickup or close microphone, laptop running Csound, L-Bow wristband hardware, and a SoftStep 3 MIDI foot controller.

**Future Perspectives:** While the 10-bit quantization maps broadly to smoothed effects parameters, finer control applications may notice resolution limits. Future revisions will explore the `arduinoReadF` opcode for native floating-point streaming and will increase the baud rate from 9600 to 115200—reducing frame transmission time from 13.5 ms to 1.1 ms. Adapting this mapping structure to CsoundUnity or Cabbage will extend these architectures into VR and plugin-based performance contexts.

## 6. REFERENCES

- [1] M. Wright and A. Freed, “Open SoundControl: A new protocol for communicating with sound synthesizers,” in *Proc. Int. Computer Music Conf. (ICMC)*, Thessaloniki, Greece, 1997, pp. 101–104.
- [2] J. ffitch and R. Boulanger, “New Arduino opcodes to simplify the streaming of sensor and controller data to Csound,” in *Proc. 6th Int. Csound Conf.*, Athlone, Ireland, 2022.
- [3] D. Young, “The Hyperbow controller: Real-time dynamics measurement of violin performance,” in *Proc. Int. Conf. New Interfaces for Musical Expression (NIME)*, Dublin, Ireland, 2002, pp. 201–206.
- [4] K. A. McMillen, “Stage-worthy sensor bows for stringed instruments,” in *Proc. Int. Conf. New Interfaces for Musical Expression (NIME)*, Genoa, Italy, 2008, pp. 347–348.
- [5] J. A. Paradiso and N. Gershenfeld, “Musical applications of electric field sensing,” *Computer Music Journal*, vol. 21, no. 2, pp. 69–89, 1997.
- [6] M. Kimura, N. Rasamimanana, F. Bevilacqua, B. Zamborlin, N. Schnell, and E. Fléty, “Extracting human expression for interactive composition with the augmented violin,” in *Proc. Int. Conf. New Interfaces for Musical Expression (NIME)*, Ann Arbor, Michigan, USA, May 21–23, 2012.
- [7] S. D. Thorn, “Alto.Glove: New techniques for augmented violin,” in *Proc. Int. Conf. New Interfaces for Musical Expression (NIME)*, Blacksburg, Virginia, 2018, pp. 334–339.
- [8] A. Hunt, M. M. Wanderley, and M. Paradis, “The importance of parameter mapping in electronic instrument design,” *Journal of New Music Research*, vol. 32, no. 4, pp. 429–440, 2003.
- [9] V. Lazzarini, S. Yi, J. ffitch, J. Heintz, Ø. Brandtsegg, and I. McCurdy, *Csound: A Sound and Music Computing System*. Springer, 2016.