AED Controller

Team members: Peter Ju, Aline Yue, Zuocheng (Joe) Wang

Project Overview

For our project, we created an Arduino MIDI controller for virtual instruments on laptops. As three musicians (Joe and I, Peter, were in a band), we recognized a common challenge. Professional MIDI controllers are expensive, often costing $200-500+, creating a barrier for student musicians/producers and emerging artists who want to experiment with electronic music production. We designed and built a custom MIDI controller using Arduino Leonardo, five slide potentiometers, and a button, creating an affordable (~$40-50) alternative that provides essential music production controls. Our controller interfaces directly with Logic Pro or any other Digital Audio Workstations (DAWs), giving users control over virtual instruments.

Typical user personas for our product are:

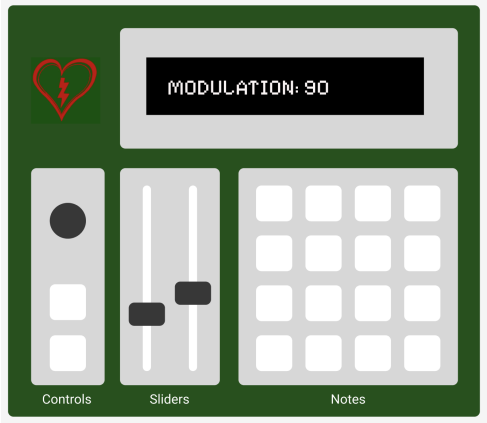
- Band members who want to add synthesizer textures to live performances without buying expensive keyboards
- Indie producers creating music on a budget
- Music students learning about MIDI, synthesis, and electronic music production
- Jam session musicians who want quick, hands-on control without complex setups

Our design philosophy was simple and quite intuitive. When the user presses a button, a note plays. If they move a slider, the corresponding parameter changes in real-time. When they release the button, the note ends. A complete controller includes a button and 5 sliding potentiometers. They send standard MIDI messages, making it compatible with any virtual instruments that accept MIDI input. So the goal here really is to create a functional and low-cost MIDI controller with a reproducible design and demonstrate a practical application for Arduino boards.

The project is a product of teamwork. I am mainly in charge of the programming aspect. Joe is responsible for the wiring and integration (testing) phase of the project, and Alina helped with the appearance design of the controller as well as general project management.

Design Process

Our initial design incorporated the Adafruit NeoTrellis (4x4 button grid with RGB LEDs) that we envisioned as a launchpad for inputting notes. The concept was to have 16 pressure-sensitive buttons that could trigger different notes, with visual feedback through color-coded LEDs. Our initial design (shown in the screenshot below) features 16 different MIDI notes and 2-3 sliders.



Initial Figma Design Screenshot

We began prototyping with Arduino Uno (our provided hardware), but quickly discovered a critical limitation: the Uno's chip doesn't support the MIDIUSB library which is crucial for sending over MIDI data to the connected device. This is why we switched to use Arduino Leonardo, a board with MIDI support.

We attempted to integrate Adafruit NeoTrellis with Arduino Leonardo. However, it seems that the buttons on the launchpad were responding to pressing for a short amount of time, then it would completely die without responding to any further touches. We've tried to fix this issue with less LED intensity, plugging Leonardo board into a 9V outlet for its power source, and debugging with code, but none of it seems to have fundamentally fixed the problem. After spending too much time troubleshooting, we pivoted to a slider-based controller with a single button trigger.

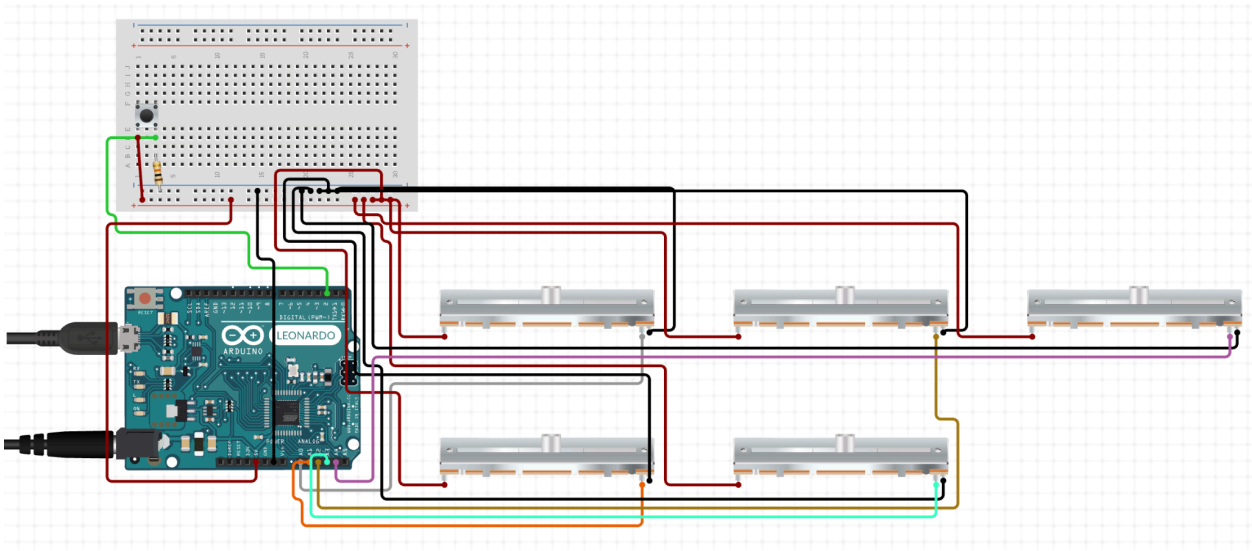
A working 5-slider controller is definitely more valuable than a broken 16-button controller. Also, sliders perform great when it comes to continuous parameter control (such as cutoff-frequency, resonance, etc.). Also it would draw less power so we don't have to worry about providing an external power supply to the board besides the laptop it's connected to. The 5-sliders are responsible for volume, pitch, cutoff-frequency, resonance, and vibrato.

By sending MIDI signals to different channels, we are able to communicate with virtual instruments in Logic Pro. For pitch, we thought we could use pitch bend. However, it has a relatively limited bound (plus or minus 2 semitones) which we then switched to using note number control. While some MIDI CC messages (like volume) work automatically, others such as resonance and cutoff require manual MIDI Learn mapping in Logic Pro. Luckily, this was relatively easy to do in modern DAWs.

Hardware Documentation

Bill of Materials

Component	Quantity	Specifications
Arduino Leonardo	1	ATmega32u4
Linear Slide Potentiometer	5	10kΩ
Push Button	1	SPST, Normally Open
Resistor	1	10kΩ
Breadboard	1	400 Point
Jumper Wires	Several	
USB Cable	1	USB-A to Micro-USB



A graphical visualization of the wiring (made with www.circuito.io)

For the sliding potentiometers, their GND and VCC goes to Arduino's GND and 5V pins. Their signal pins are wired to Arduino's analog pins A0 - A4. The push button will be connected to the 5V voltage supply and ground. A third wire will go from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a 10kΩ resistor to ground. The other leg will connect to the power supply.

The USB Cable will be used to connect the Arduino Leonardo board with the laptop. The laptop will supply power to the board and the board will send over MIDI signals to the laptop. (Use an adaptor if needed).

Software Documentation

The code uploaded to Arduino Leonard will be uploaded separately to Canvas.

The software file depends (includes) the MIDIUSB Library. In the Arduino IDE, you download by going to Sketch -> Include Library -> Manage Libraries and searching for “MIDIUSB.” This library is essential as it provides USB MIDI functionality for native USB Arduino boards.

MIDI Mapping Table

Analog Pin	**MIDI Channel**	**Parameter**
A0	CC 7	Volume
A1	CC 1	Vibrato
A2	CC 74	Cutoff Frequency
A3	CC 71	Resonance
A4	Pitch	Note number 36-96

Button will be mapped to NoteOn/NoteOff events.

Important Functions Explained

```
void noteOn(byte channel, byte pitch, byte velocity) {
        midiEventPacket_t noteOn = {0x09, 0x90 | channel, pitch, velocity};
        MidiUSB.sendMIDI(noteOn);
        MidiUSB.flush();
}
```

This function sends a MIDI "Note On" message to start playing a note. MIDI was designed so multiple instruments can share one cable. Channels let you control different instruments independently. Each instrument only responds to messages on its channel and ignores others. In our case, we only have one controller (Logic Pro instrument), so we use Channel 0.

Every MIDI message starts with a status byte that encodes both the message type and the channel. The 4 least significant bits in the status byte are used for encoding channel information, while the 4 most significant are for message type. By using a bitwise OR operation, we can combine the two information to send over to the laptop.

Pitches in MIDI are denoted by an integer within the range of 0-127. In our application, we are mapping this between 36 (Low C) and 96 (High C). Velocity denotes how hard a note is played

(ranging from 1 to 127 inclusively), which affects volume and the timbre of the sound. Here by default the velocity notes are played at 100.

At the end of the function, we are calling the MidiUSB.flush() function. This forces immediate transmission to prevent the message getting buffered or delayed.

Debouncing
Physical buttons don't always make clean electrical contact. When pressed/released, they could bounce, rapidly connecting/disconnecting, causing inconsistency between the note played and the user's behavior. This emerges after testing our program out as sometimes signals will be sent to the laptop even if the button is not pressed down. We even had duplicated notes playing at the same time which led to absolute chaos (that might be great for a horror movie soundtrack).

Referring to the Arduino Documentation on Debouncing (https://docs.arduino.cc/built-in-examples/digital/Debounce/), we start a timer and wait for 50 milliseconds before accepting the change as valid when the button state changes. If the button state changes again during this waiting period (indicating a bounce), we reset the timer and start waiting again. Only after the button has maintained a consistent state for the full 50 milliseconds do we actually process the press or release and send the corresponding MIDI message.

Analog Threshold Filtering
The Arduino's readings of potentiometers might naturally fluctuate even if the sliders themselves are completely stationary. This noise could come from several sources such as the electrical interference on the breadboard and the minor changes in the power supply voltage. Without filtering, these fluctuations would be converted to MIDI control change messages and sent continuously, causing jittery parameter changes (which happened while we were testing!). Our solution filters by tracking the previously sent value for each slider and only transmitting a new MIDI message when the current reading differs from the previous value by more than a defined threshold (1).

Setup

Once your Arduino is programmed and connected via USB, Logic Pro needs to recognize it as a MIDI input device. The Arduino Leonardo will appear as "Arduino Leonardo" in the system MIDI devices list.

To enable the Arduino as a MIDI device, open Logic Pro -> Settings -> MIDI and look for "Arduino Leonardo" in the device list under "Inputs." Make sure the checkbox next to "Arduino Leonardo" is checked/enabled.

In Logic Pro, create a new Software Instrument track and choose a synthesizer instrument (Retro Synth recommended). Make sure the track is selected (highlighted). Press your Arduino button and you should hear a note play immediately.
The Arduino sends standard MIDI messages, but some need to be manually mapped to synthesizer parameters. Volume (CC 7) and Modulation (CC 1) usually work automatically, but Cutoff (CC 74) and Resonance (CC 71) typically require manual assignment. MIDI Learn is the fastest way to map your sliders to synth controls.

Open your synthesizer's interface (click on it in the channel strip). Locate the Cutoff (Resonance) knob on the synth. Control-click on the knob and select "Learn MIDI CC Assignment" or "MIDI Learn." Move the designated slider (A2 for cutoff, A3 for resonance) on your Arduino controller. Logic Pro will save the MIDI mappings with your project.

Testing and Results

Throughout the development process, we conducted testing to verify functionality and identify issues. Testing occurred in three phases: component-level testing (individual sliders and button), integration testing (Arduino code with MIDI communication), and performance testing (real-world use in Logic Pro with synthesizers).

During our in-class demonstration on December 9th (today), we successfully showed the controller's functionality. The demonstration video (recorded by instructors) shows real-time MIDI communication between Arduino and Logic Pro, button triggering notes on/off with clean response, five sliders controlling their assigned parameters, and live performance technique: combining button presses with simultaneous slider movements. The demonstration proves that our controller will work reliably in a live performance context and responds quickly enough for musical expression.

What Works Well

Button Response: The note on/off triggering is highly responsive and reliable. After implementing software debouncing, we achieved zero false triggers during testing. The 50ms debounce delay is imperceptible to the user while completely eliminating button bounce artifacts. Musicians can press the button rhythmically or hold it down for sustained notes.

Pitch Control (Slider A4): The pitch slider provides smooth note control across a wide range (MIDI notes 36-96). Unlike our initial pitch bend approach, the direct note control implementation allows for unlimited (but within a reasonable) pitch range and creates a glissando effect when the slider is moved while holding the button.

Volume Control (Slider A0): Volume response is pretty smooth and predictable. MIDI CC 7 is universally recognized by all synthesizers and DAWs, so this control required no mapping and worked immediately.

Filter Controls (Sliders A2 & A3): Cutoff and resonance sliders dramatically affect sound character, especially with subtractive synthesizers like Retro Synth. The cutoff sweep (from dark/muffled to bright/full) is the most distinguished sonic transformation available on the controller. Resonance adds emphasis at the cutoff frequency, creating "wah" effects.

Vibrato/Modulation (Slider A1): CC 1 (modulation wheel) is widely supported and often pre-mapped to LFO modulation in most synthesizers. This slider successfully adds expressive vibrato effects. While the degree of modulation varies by synthesizer preset, the control itself functions perfectly.

Physical Layout: The breadboard layout with five sliders and the button positioned separately provides intuitive control. Users can easily manipulate multiple sliders simultaneously or focus on an individual parameter. The controller is supported by cardboard glued with sturdy supportive structures. Its surface layer is decorated with a silver color scheme that glitters under

light and hides away the wires/cables so the musician can fully devote themselves to their music. A laser cutted logo of our band (thanks to the support of Open Lab @ Pitt) is glued to the surface layer as well showing the product proudly made by music enthusiasts.

Limitations

Single Note Polyphony: The controller can only trigger one note at a time due to the single-button design. Unlike keyboard controllers that allow chords and multiple simultaneous notes, our design is monophonic, which limits harmonic possibilities.

Fixed Slider Assignment: Slider mappings are hardcoded in the Arduino sketch. Users cannot reassign sliders without re-uploading code with modified CC numbers. Professional controllers typically offer on-device configuration or software editors for flexible mapping.

Limited Visual Feedback: The controller provides limited indication of current parameter values (no clearly indicated levels). Users must rely on audio feedback and Logic Pro's visual interface to know slider positions. Adding an LCD display or LEDs would provide better parameter monitoring.

Manual Mapping Required: CC 74 (cutoff) and CC 71 (resonance) must be manually mapped in most synthesizers using MIDI Learn. While this is standard practice for advanced MIDI controllers, it creates an initial setup barrier.

Synthesizer Dependency: Filter controls have little to no effect on sampled instruments (e.g.: pianos, drums, orchestral sounds). The controller is optimized for subtractive synthesizers with resonant filters. This limits the range of compatible instruments but is inherent to the CC choices we made.

No Velocity Sensitivity: Note velocity is fixed at 100 (out of 127). Players cannot vary attack dynamics as they would with keyboard velocity sensitivity or pressure-sensitive pads.

Possible Improvements

- Visual feedback (LCD Display that shows parameter names and values)
- Implement velocity control
- Add octave shift buttons (up/down an octave, which is 12 semitones)
- Add multiple buttons for chords
- Better material for the appearance of controller

Conclusion

Our Arduino MIDI controller successfully shows core music production control capabilities without spending commercial controller costs. Despite limitations in build quality, polyphony, and configuration flexibility, the device performs its core functions (note triggering, pitch control, and parameter modulation) reliably and responsively.

The showcase validated our design decisions of simplicity over complexity and direct MIDI control over elaborate processing. While the controller won't necessarily replace professional equipment, it serves its intended purpose as an accessible, educational, and functional introduction to DIY music technology.

Most importantly, the controller is genuine and musical. It produces sounds we enjoy playing with (that’s why we made our custom jam session backing tracks), enables expressive performance techniques, and integrates into a professional DAW workflow. For a first Arduino project, I think it is a successful fusion of hardware engineering, software development, design, and musical application.

References

https://mschoeffler.com/2020/11/14/arduino-tutorial-slide-potentiometer-slide-pot-controls-led-strip-ws2812b/
https://docs.arduino.cc/built-in-examples/digital/Debounce/
https://docs.arduino.cc/built-in-examples/digital/Button/
https://docs.arduino.cc/libraries/midiusb/
https://docs.arduino.cc/hardware/leonardo/