POLITECNICO
MILANO 1863

Design and Implementation of a Computer Music System: Polisonic Creations

Matteo Orlandin, Patrick Niec, Claudio Costantini, Emanuele Turbanti, Gabriele Leonardi

Computer Music - Languages and System

27/05/2024

Introduction

In the dynamic realm of music production, innovation perpetually drives progress. Our project, the Voice-Modulated Synthesizer Plugin, represents a pioneering endeavor that seamlessly integrates vocal recordings with traditional synthesizer parameters. Within this context, envision a sonic landscape where the human voice transcends its conventional role, becoming the catalyst for shaping electronic soundscapes. Imagine creating ethereal soundscapes where your voice becomes the driving force behind filter cutoffs, detuning, distortion, chorus, ecc…

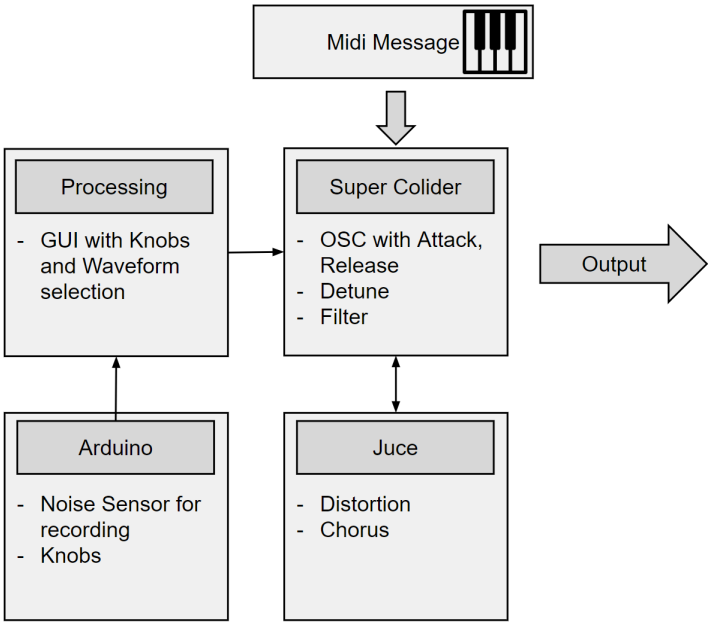
The Goal

As music producers, we always grapple with a challenge: the temporal dissonance between creative inspiration and parameter adjustment. When crafting intricate soundscapes, the rhythm that initially ignited our imagination often dissipates during the meticulous process of parameter tuning. The delicate balance between artistic vision and technical implementation can be elusive.
Our plugin emerges as a solution to this common conundrum. By capturing vocal envelopes we bridge the chasm between rhythmic intuition and parameter manipulation. Now, whispered cadences, soulful hums, or impassioned exclamations transcend mere audio artifacts; they become conduits for shaping the very fabric of synthesized music.
Our Synthesizer invites artists to reimagine their creative process, where the human voice, once ephemeral, now resonates as a tangible force. As we delve into the technical intricacies, we unveil a canvas where rhythm and creativity harmonize—a canvas painted with the hues of innovation.

Design and Topology

The topology shown below describes the integration and interaction of the components in the project. The synthesizer output is generated through SuperCollider, which processes OSC messages for parameters such as attack, release, detune, and filter. The system involves a MIDI device, Processing for GUI, an Arduino with a noise sensor and knobs, and Juce for adding effects like distortion and chorus.

The MIDI device functions by playing notes and sending MIDI data, which includes several information regarding the user’s performance recording, including note on/off and velocity signals. It outputs these MIDI messages to SuperCollider.
Processing serves as the graphical user interface (GUI) for the system, featuring knobs for parameter adjustments and waveform selection.



The GUI knobs allow users to adjust envelope parameters and effects, while the waveform selection lets users choose different waveforms. Processing receives serial messages from Arduino and the user input from the GUI, and transforms this data into OSC messages that are sent to control the parameters of SuperCollider's Synths.

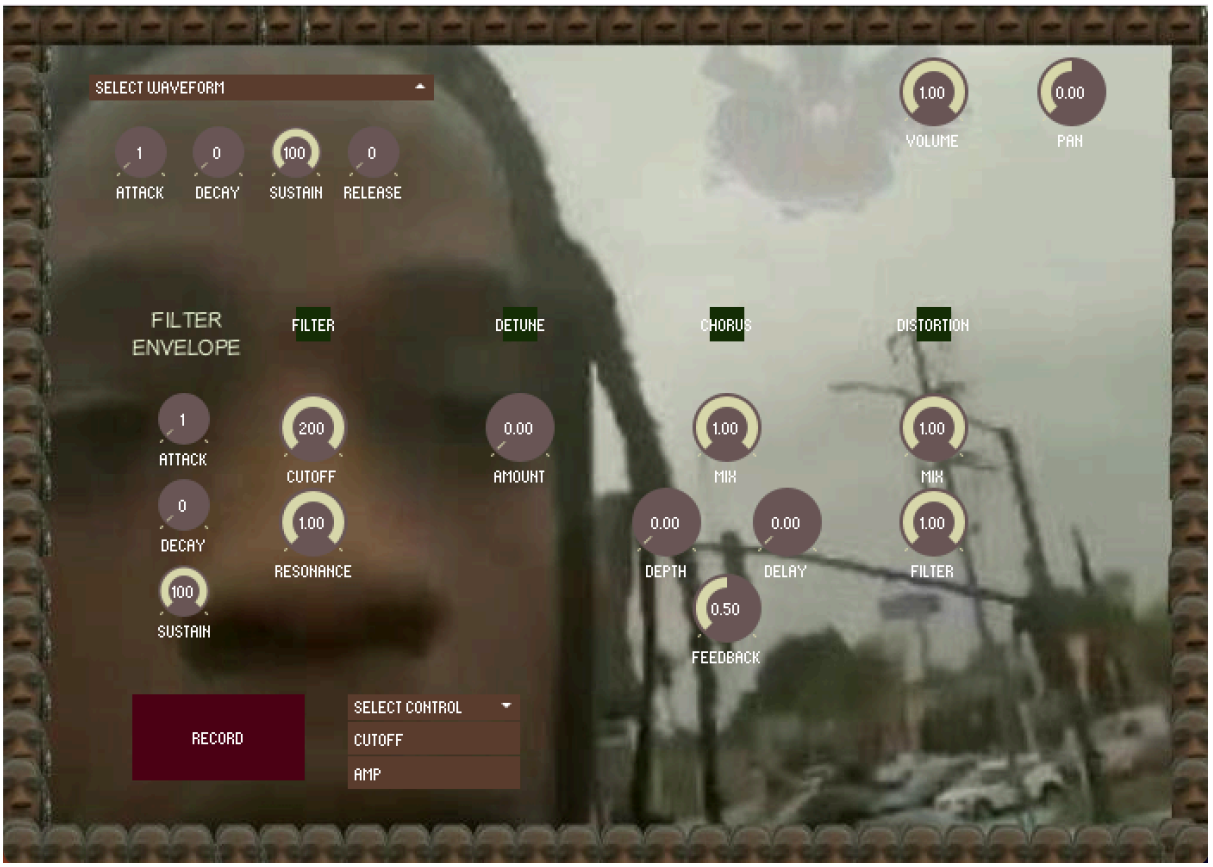
The Arduino captures environmental noise using a dedicated noise sensor and provides additional hardware knobs for real-time control. The noise sensor records environmental sounds for modulation or processing, and the knobs offer manual control over parameters such as modulation and effects. The Arduino sends sensor data and knob adjustments to Processing as a serial output.

Juce adds audio effects such as distortion and chorus to the synthesized sound. The distortion component adds harmonic content and warmth to the sound, while the chorus component adds richness and depth by creating slight variations in pitch and timing.
The effects chain is implemented in SuperCollider as a Synth, whose input corresponds to the signal coming from the oscillator (after the filter), and the output is routed to the main bus. This means, basically, that Juce receives audio input from SuperCollider, processes the audio with the added effects, and then outputs the final audio signal.

SuperCollider is the core sound synthesis engine that processes OSC messages and generates audio output. It handles messages for parameters like attack, release, detune, and filter settings, and generates and modulates sound based on these parameters.
SuperCollider receives MIDI data from the MIDI device and OSC messages from Processing, then outputs the audio to Juce for further processing.
The final processed audio signal is output from Super Collider, ready for playback or further routing.

Graphic User Interface Design (GUI)

We tried to keep the GUI very simple and intuitive to reduce as much as possible the time spent between the inspiration process and the implementations.



Waveform Selection Area (Top Left): At the upper left quadrant of the interface, users are greeted with a waveform selection module. This feature empowers users to sculpt the foundational character of their sound by toggling between classic waveforms: the pure tones of a Sine Wave, the harmonic richness of a Square Wave, and the bold, buzzy texture of a Sawtooth Wave. Each selection dynamically alters the sonic palette, offering a versatile starting point for sound design.

Sound Modulation Controls (Below Waveform Selection): Directly beneath the waveform selector lies the heart of dynamic expression: the ADSR envelope controls. These knobs and sliders are meticulously laid out to provide intuitive manipulation of the Attack, Decay, Sustain, and Release parameters. Adjusting these will shape the temporal evolution of the sound, from the initial transient to the lingering fade, granting the user complete control over the note's articulation.

Effects Central Hub (Middle Section): Dominating the central area of the GUI is the effects hub. Here, users can delve into the Filter section to sculpt frequencies, employ Detune for subtle pitch variations, add Chorus for a lush, wide soundscape, or introduce Distortion for gritty texture. Each effect comes with dedicated controls for fine-tuning, alongside an advanced parameter matrix for seasoned users seeking granular customization. For all the effects we dedicated Knobs and Buttons in Arduino; users can adjust the intensity of each

effect using dedicated knobs, instantly switch between clean and processed sounds with enable/disable buttons, and receive visual feedback through LED indicators.
Recording and Rhythm Application Module (Bottom Section): The lower section of the GUI is dedicated to the plugin's innovative Record and Apply feature. The Record Button invites users to capture their vocal rhythms or beatboxing directly within the plugin. With a simple tap, the user's live input is recorded, and with a double-click, the recording session ends. The captured rhythm can then be seamlessly applied to modulate the Filter Cutoff or Amplitude of the sound in real-time, infusing the user's personal touch into the synthesis process. This intuitive design ensures that even complex modulation techniques are accessible and engaging.
Loop and Playback Functionality: Once the recording is captured, the user can activate the loop function to continuously play back their rhythmic creation. This allows for hands-free operation, enabling the user to focus on further tweaking the sound or layering additional elements. The looped rhythm becomes an integral part of the sound, moving the Filter Cutoff or Amplitude in sync with the user's original performance, bringing a dynamic and organic feel to the synthesized sound. To reset the recording and erase the memorized one the user can simply press the reset button on Arduino and start again with a new rhythm.

Technological Challenges and Solutions

During the development of our plugin, designed to transform vocal expressions into dynamic parameters for synthesizers, we encountered several technological challenges. These challenges span communication protocols, software integration, and hardware control, each requiring careful consideration and problem-solving.

One of the primary challenges has been ensuring seamless communication from Arduino through Processing to SuperCollider, particularly in handling Open Sound Control (OSC). Effective real-time data transmission and accurate parameter manipulation are crucial for maintaining the integrity of the creative process. Setting up OSC communication involves defining the network address for message delivery, managing the reception and processing of these messages, and ensuring that the correct parameters are updated in response to the incoming data. Adding MIDI communication introduces another layer of complexity, requiring the integration of MIDI messages with OSC protocols to facilitate comprehensive control and flexibility.

Working with SuperCollider also presents its own set of challenges. The programming language and syntax can be prone to errors, making the development and debugging processes more difficult. Frequent issues with script execution and system reboots can interrupt the workflow, while establishing reliable bus connections and communication channels within SuperCollider is often problematic. Additionally, integrating imported JUCE effects has led to further complications, with bugs affecting performance and stability.

Another significant challenge has been achieving precise control of GUI elements via Arduino, while maintaining synchronization between hardware and software controls. For example, when the cutoff filter is active, it is essential to manage the chosen cutoff value through the GUI carefully to avoid conflicts with dropdown list selections. Ensuring that

hardware controls on the Arduino and GUI interactions do not interfere with each other requires robust logic to handle state changes and input priorities effectively.

Problem 1: OSC and MIDI Protocol Implementation

Communication from Processing to SuperCollider through OSC protocol:
In this section we handle Open Sound Control (OSC) messages that are sent from the GUI to SuperCollider:
1. OSC Address Initialization: Initially we set up the OSC communication by defining the network address (‘n = NetAddr("127.0.0.1")’) to which OSC messages will be sent.
2. OSC Message Handling: The ‘OSCFunc’ function is used to define what happens when an OSC message is received. The OSC message is expected to have the address ‘/address’ and to be sent to the network address ‘n’ on port ‘57120’.
3. Value Update: The first argument of the OSC message (‘msg[1]’) is used to update the value.

The MIDI communication is set up as follows:
1. MIDI Channel and Notes Initialization: We need to first set up the MIDI channel (‘~midiChannel’) to 0 and initialize an array (‘~notes’) of 128 elements, representing the 128 possible MIDI notes in a keyboard.
2. MIDI Note On Handling: The ‘MIDIdef.noteOn’ function is used to define what happens when a MIDI noteOn message is received. If the channel of the incoming MIDI message matches ‘~midiChannel’, a new Synth (‘~notes[note]’) is created based on the ‘dynamicLPF’ SynthDef defined earlier. The parameters of the Synth are set based on various variables.
3. MIDI Note Off Handling: The ‘MIDIdef.noteOff’ function is used to define what happens when a MIDI note off message is received. If the channel of the incoming MIDI message matches ‘~midiChannel’, the gate of the corresponding Synth is set to 0, which should trigger the release stage of the envelope. The Synth is then freed and the corresponding element in the ‘~notes’ array is set to nil.

Problem 2: Controlling the Knobs of the Gui with arduino and meanwhile in the GUI

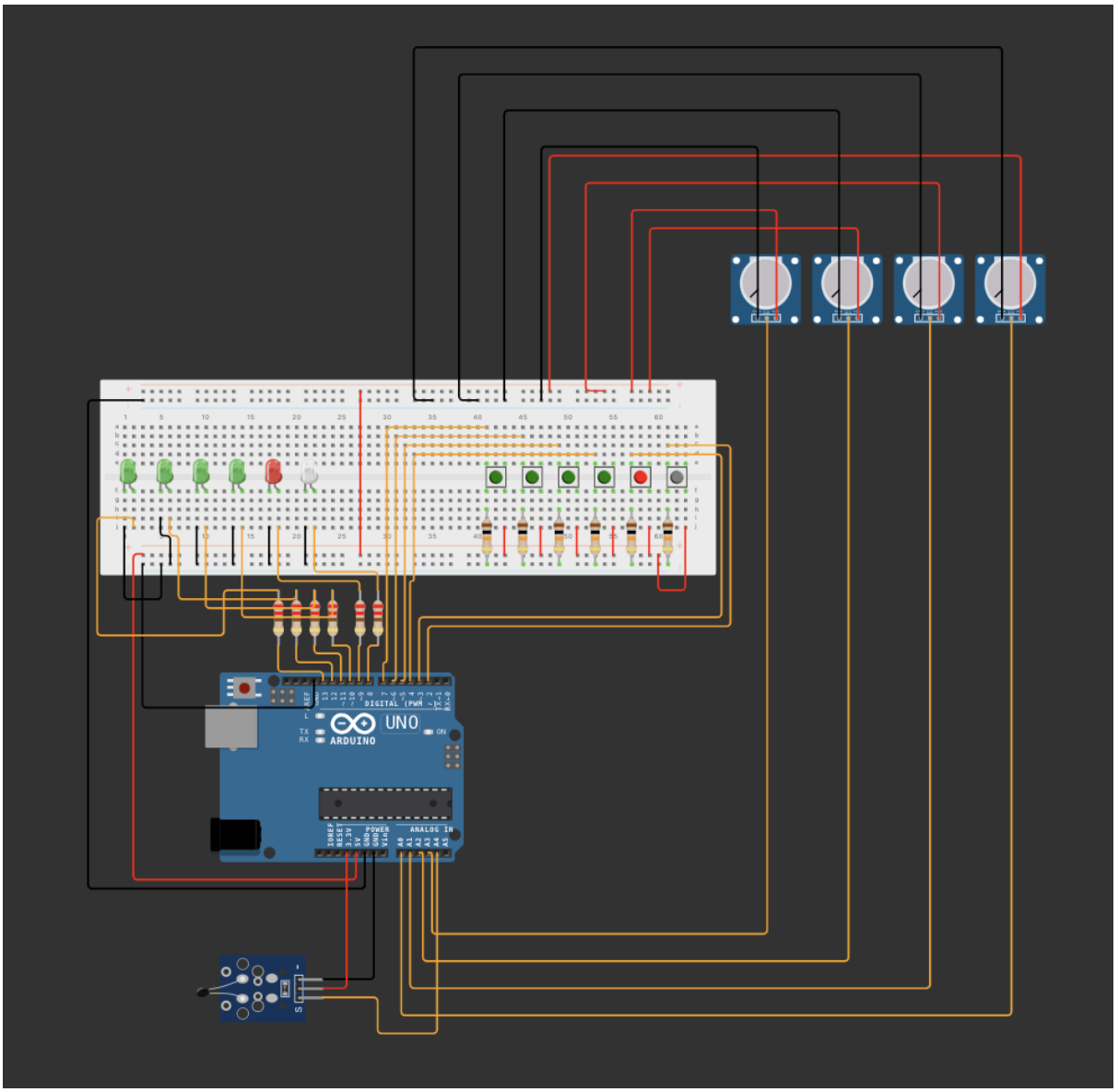
A challenging issue we had to face was controlling the knobs with the GUI while they could also be controlled through hardware interface.
This could be solved for some of the parameters via the following statement:

```
// Filter update                                          // If the filter cutoff button is on (so if the filter's state variable is true), we enter this loop
if(cutoffButtonState){
   filterButton.setValue(1);
   cutoffButtonState = !cutoffButtonState;
   filterButton.setColorBackground(color(40, 150, 90));
   sendOscMessage("/cutoffState", 1);                     // We're telling SC that the filter is active. We will modify the "lpfOn" parameter in SC
   if(d2.getValue() != 0){                                // If the second dropdown list isn't set on "Filter Cutoff", then we receive values from the knob
     cutOffKnob.setValue(cutOff);                         // We set the cutoff via GUI, so the controller can be manipulated through the GUI when it isn't chosen in the dropdown list
     sendOscMessage("/cutOff", cutOff);                   // Send the cutoff value to SC
   }
   }
```

This conditional statement is accessed when the cutoffButtonState is "true", so when the filter is active. Here we send the cutoff value, chosen through the GUI, only if the cutoff isn't selected in the ‘d2’ dropdown list.

Summing up we could implement this feature for only two parameters: the filter cutoff and the amplitude. This is due to the fact that we control these parameters through the dropdown list, selecting when a parameter is actually manipulated manually or through the Arduino’s sensor.

Problem 3: Arduino Setup and adjustment



Looking at the Arduino setup, it may seem quite complex at a first glance.
We have 4 buttons assigned to the button on/off parameters respectively for the filter, detune, chorus and distortion. The other 2 buttons are used to record/loop the incoming signal and delete the loop (this last option basically resets the circuit).
The 4 knobs control the filter cutoff, the detune, and the two effect mixes.

This complexity resulted in a particularly challenging setup every time a test was needed. Eventually the whole circuit was soldered in a 3D printed casing in order to facilitate the transportation and testing.

Conclusion and Future Developments:

The synthesizer in its current state is a solid starting point, showing its worth as a practical tool for music creation. It's not quite ready to be sold yet, but it's a strong base for an idea that could use some more work. This plugin could change the way we use automation in music, making it easier and quicker to produce sounds. The synthesizer has all the basics needed for today's music and opens up new, often faster ways to make music. With some more development, it could become a very useful tool for producers.
One of the most crucial improvements would be to modify the code to integrate the use of the microphone as the principal recording hardware.
Other future developments and upgrades will focus on enhancing the stability and functionality of the plugin. Improvements in communication protocols, particularly in refining OSC and MIDI message handling, will further streamline real-time data transmission and parameter manipulation. Continued work on SuperCollider integration will aim to resolve existing bugs and improve the reliability of script execution and system reboots. Additionally, an important feature that is missing is the possibility to modify and edit the recorded vocal envelope to have a greater control on it and ensure the best sound. Furthermore, the system would benefit from the capability to adjust and refine the recorded vocal envelope. This enhancement would provide more precise control over the sound quality, ensuring optimal audio output.
These future enhancements will not only address current limitations but also open up new possibilities for creative exploration. By continually refining and expanding our plugin's capabilities, we aim to provide artists with even more powerful tools for bringing their musical visions to life.