POLITECNICO
MILANO 1863

CMLS - Homework - 2023/24

ANSAL Three Men Band Project

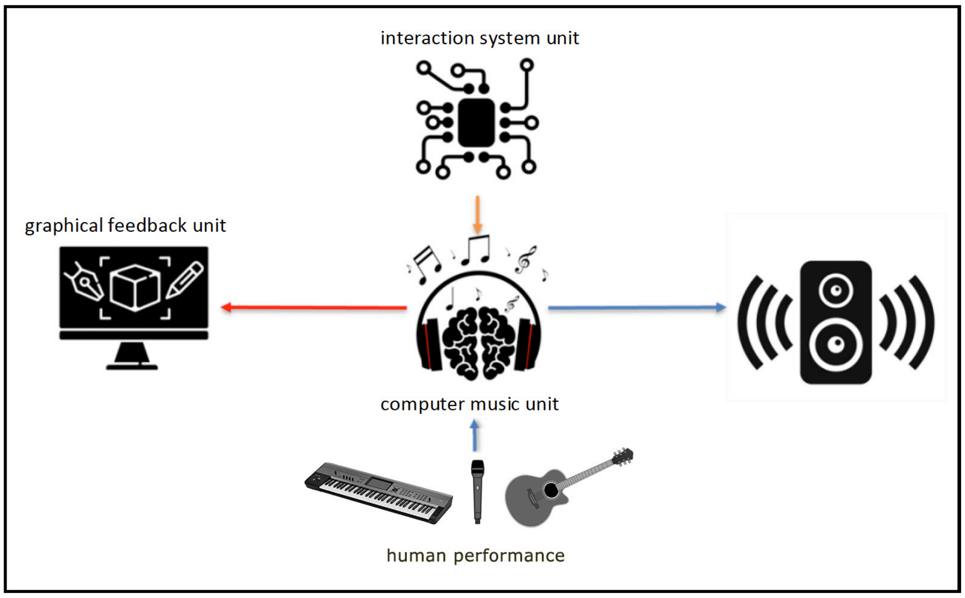
AN - Angelo Antona - 10665838
SA - Salvatore Benvenuto Zocco - 11035358
AL - Alessandro Manattini - 11006826

Table of Contents

"$e^{ix} + 1 = 0$ (...) is the cornerstone of a major mathematical edifice that represent musical signals, among other things, in a crisp and penetrating way.[1]"

1. The Main Goal

Having had a clear assignment, we had to decide the main purpose and to derive the content targets of our work. At ANSAL, we are all musicians, singers/songwriter, producers and sound engineers, and it was clear to us, from the very beginning, that we wanted to build a system that could help us play, in real time, live, or in Studio, enhancing our expressiveness.



1.1. Production Objective and Sound Design

Our different skills include composing, singing, playing keyboards and guitar, sequencing rhythm and drum tracks. Therefore the choice of three modules: one for the voices, a second for keyboards, a third for guitar. Rhythm and drum tracks had to be sequenced. The bass track had to be assigned to keyboard or guitar.

Sounds had to be tailored for contemporary pop/dance productions, with a choice of pre-set tailored to specific demo songs to be decided and presented as a simplified demos.

1.12 Module Assignment

We decided to write a Vocoder, to create computer-sounding main and choir voice lines. For such a task, JUCE seemed us to be the most logical choice. Being SuperCollider a server/client powerful ambient, with SClang being a higher level language, it seemed to us more apt for writing synthetic keyboard sounds. Regarding guitar, we were undecided at first, but pretty soon SC revealed to be geared for the purpose, with a "SoundIn" function that turns the incoming audio signal into a regular Ugen.

1 Gareth Loy - Musimathics, volume 2 - p. 67.

2. Guitar Synth

This module has been deployed in SuperCollider. We've called it "Guitar" because the way we used it but, technically, it could be used with any audio input signal. Care has been put in the choice of the interface - the MMA-A - , to ensure a stable conversion, and the mic - 4099 - a precise supercardioid, both from DPA Microphones.

2.1 The Algorithms

A good starting point was the study of the SoundIn.ar UGen. For our purpose, we decided to always have a mono input and a stereo (or at least stereo centered) output.

2.1.1 Bass Synth

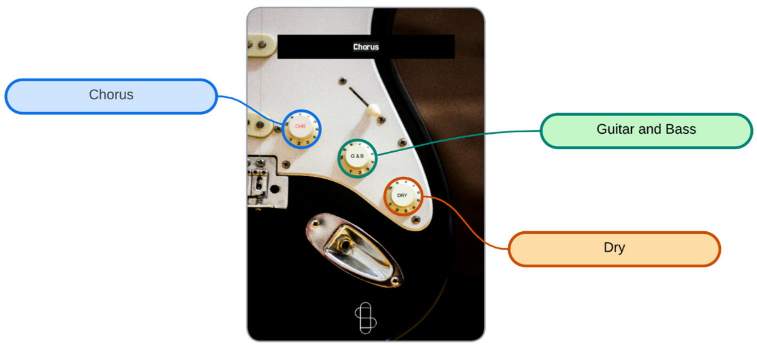
The use of the Pitch.kr and Amplitude.ar UGens, along with SoundIn, allows for a stable tracking of the incoming signal that, assigned as a frequency parameter to an UGen like SinOsc.ar, in its turn generates a stable, controllable output. Initially, and playing single notes, we tuned it straight (1.0) and per octaves (0.5 lower, 2.0 higher), but we pretty soon discovered few interesting effects: the coefficients used needed to be fine tuned to compensate for the initial detuning of the strings in play and the slight delay of the output, and accidentally playing more than one note, the resulting pitch was attributed to the note of higher volume in execution, thus allowing a polyphonic input. The latter gave us the idea to redesign the algorithm to play a synthetic bass line along with clean chords, or arpeggios.

The straight tuning was fine tuned to a more effective 0.994, with octave up at 1.985 and down at 0.497. Formant.ar was exploited instead SinOsc for a more synthetic result. To perfect the bass result, we decided to add a release time in the Amplitude.ar and to add a low pass filter after the SoundIn for the treated result, to be add in the output and balanced with chords and arpeggios from the input.

2.1.2 Guitar Stereo Effects

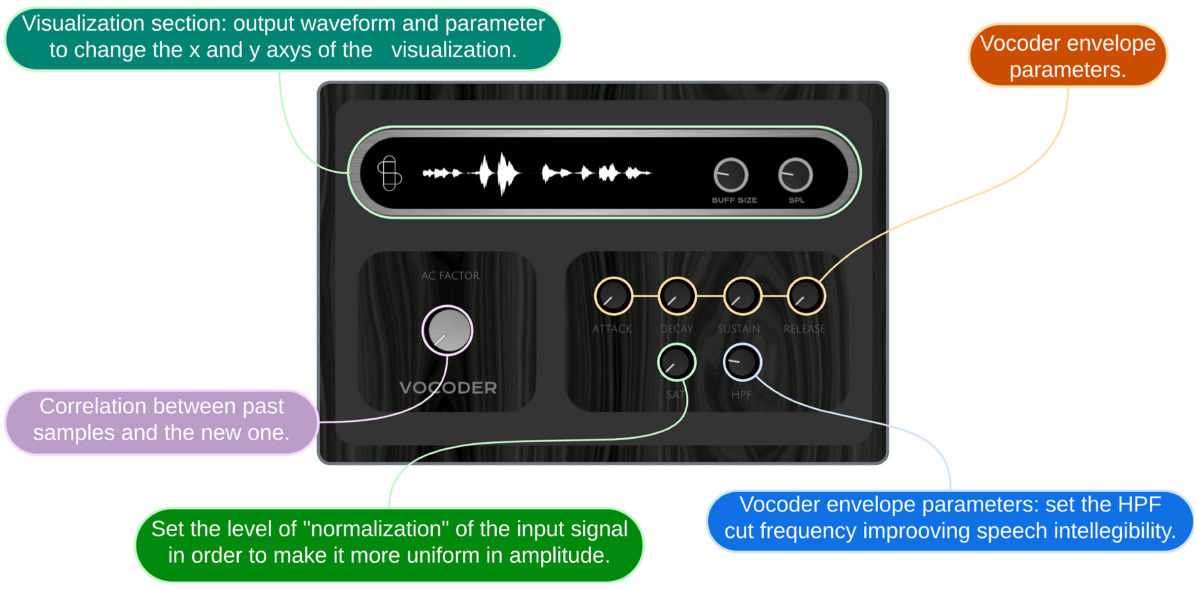
Pitch effects on the market (Choruses, Flangers, Phasers), invariably treat the whole guitar signal, adding muddiness to the lower frequencies. Having added the LP on the input, we reversed its result, to present the untouched bass registers in the center of the output, and we added an HP to gather the signal (mid and high registers), to be detuned cyclically L-R. LP and HP were offset in an almost Linkwitz-Rayleigh configuration, not to overlap the signal bands. L-R detunings obtained fine tuning two delayed SinOsc with two arguments "rate @~0.1" and "depth @~6", as in this example:

```
DelayN.ar(highSig, 0.5, SinOsc.kr((rate + 0.3)).exprange(depth*0.012, rate*0.030)*0.1);
```



Final result is a stereo Chorus (almost a Flanger), with a firm and clean bass register in the center of the image. Finally, for our demo, we implemented a control interface were we assigned the three sounds needed: Clean, Chorus, and (Clean with a) Bass.

3. Vocoder



Visualization section: output waveform and parameter to change the x and y axys of the visualization.
Vocoder envelope parameters.
BUFF SIZE
SPL
AC FACTOR
ATTACK
DECAY
SUSTAIN
RELEASE
SAT
HPF
VOCODER
Correlation between past samples and the new one.
Set the level of "normalization" of the input signal in order to make it more uniform in amplitude.
Vocoder envelope parameters: set the HPF cut frequency improoving speech intelligibility.

The polyphonic vocoder implemented in this project is a K-voice polyphonic vocoder (where the K parameter can be easily modified as a private variable in PluginProcessor), controlled via MIDI. The operation is as follows:

1) Audio Input - The audio enters through the microphone input into the PluginProcessor. Here, it is normalized (via the SimpleCompressor class) to ensure a stable input signal level.

2) Audio and MIDI Processing - The processBlock reads the incoming audio buffer and MIDI input. For each MIDI note reading, it activates one of the K voices and assigns it to process the audio buffer at the specific MIDI note frequency.

3) Voice Processing - Each vocoder voice (PhaseVoc class) processes the audio buffer with the following "leaky autocorrelation" formula, to balance past autocorrelation values with new data:

$$R_{l,\,n} = (1 - k)\, R_{l,\,n-1} + k\, x_n\, x_{n-l}$$

Where:

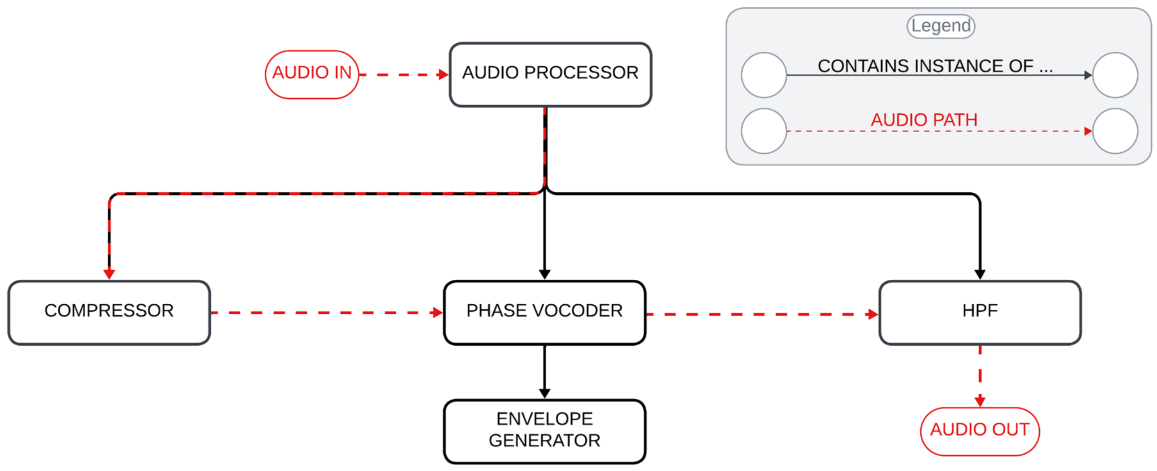
- $R_{l,\,n}$: Leaky autocorrelation at lag *l* and time *n*.
- $R_{l,\,n-1}$: Leaky autocorrelation at lag *l* and previous time step *n-1*.
- x_n : Input signal at time *n*.
- x_{n-l} : Input signal *l* steps before *n*.
- k : Leakiness constant, typically around 0.001.

This allows the vocoder to adapt to changes in the speech signal over time.

4) Envelope Application - To ensure the notes have a pleasant envelope, PhaseVoc applies methods from the EnvelopeGenerator class to the processed audio. Once this is done, it returns the buffer containing the processed result to PluginProcessor.

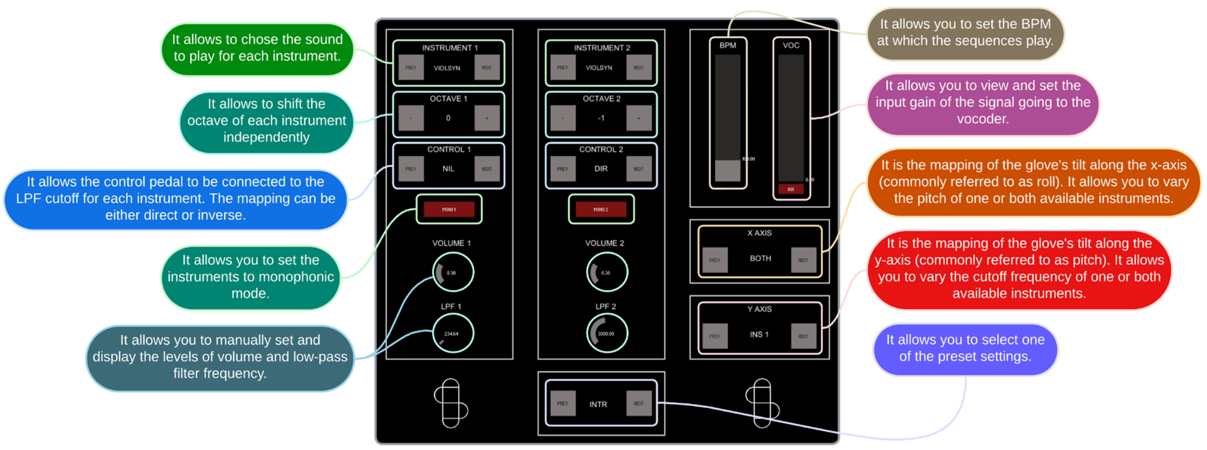
5) High-Pass Filtering - Before outputting the audio, PluginProcessor precesses it with a High-Pass Filter (HighPassFilter class), to remove the lowest frequencies and enhance audio intelligibility, where needed.

The following steps and the class hierarchy related to audio processing are easily deducible from the diagram below:



4. Synth and System Integration

The synth module includes the various functions available, all controllable via both the graphical interface and MIDI. The Vocoder[2] is integrated within the Synth module. This setup ensures that the MIDI notes used to play the synthesizer are also forwarded to the Vocoder, allowing it to modulate the voice with the same harmonies.


It allows to chose the sound to play for each instrument.

It allows to shift the octave of each instrument independently

It allows the control pedal to be connected to the LPF cutoff for each instrument. The mapping can be either direct or inverse.

It allows you to set the instruments to monophonic mode.

It allows you to manually set and display the levels of volume and low-pass filter frequency.

It allows you to set the BPM at which the sequences play.

It allows you to view and set the input gain of the signal going to the vocoder.

It is the mapping of the glove's tilt along the x-axis (commonly referred to as roll). It allows you to vary the pitch of one or both available instruments.

It is the mapping of the glove's tilt along the y-axis (commonly referred to as pitch). It allows you to vary the cutoff frequency of one or both available instruments.

It allows you to select one of the preset settings.

2 In the previous section.

4.1 Hardware Configuration

The hardware setup for the synth module is as follows:

4.1.1 MIDI Input Devices

System parameters could be modified and controlled using various MIDI controllers (details on how this is achieved in subsequent sections). The devices include:

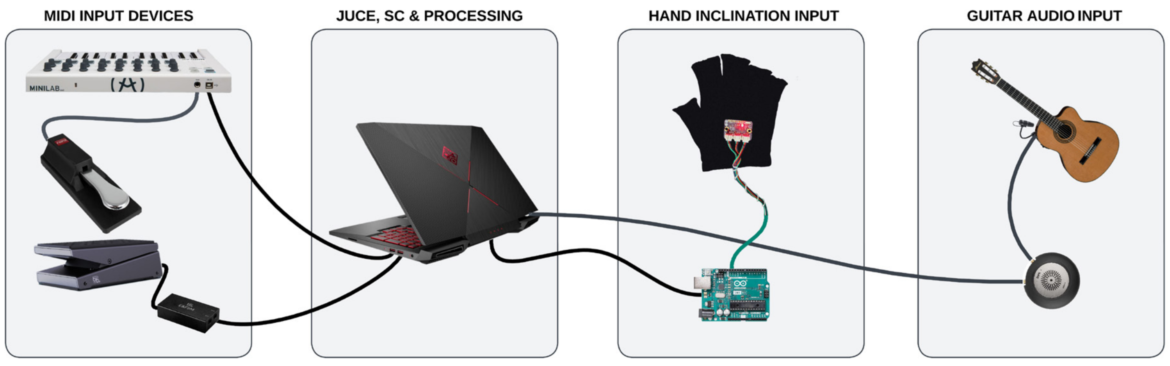
- MIDI keyboard (Arturia Minilab, fully configured within a specific preset).
- Sustain pedal.
- Volume control pedal (used for MIDI CC control).
- Analog control pedal to MIDI signal adapter.

4.1.2 Hand Inclination Input

The system takes also input from the hand movements of the keyboardist. The reason and method for using this system control will be explained later. Hardware devices are:

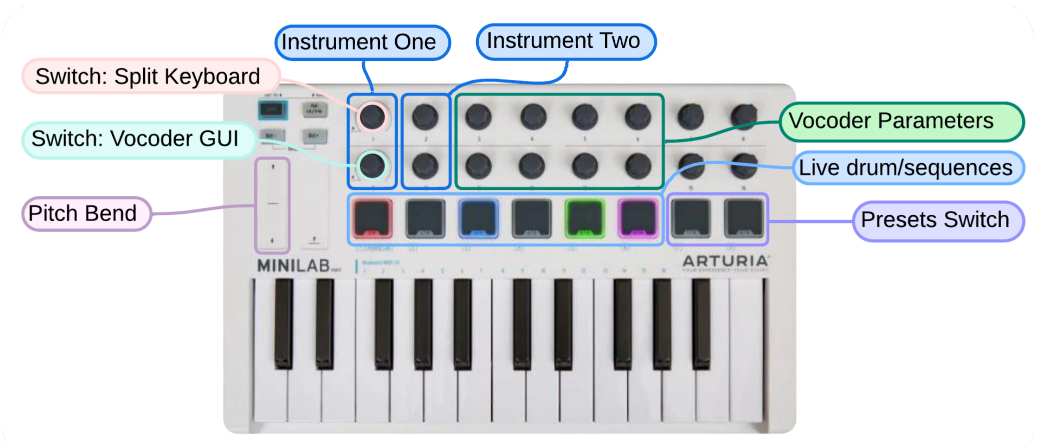
- Accelerometer: Mounted on a glove, to be worn while playing.
- Arduino Uno: Required to derive angle data from the accelerometer coordinates and to interface the accelerometer with the system.

The system components are connected as illustrated in the figure below.



4.2 Synthesizer Features

The synthesizer offers extensive configuration and parameter customization options. The list of functionalities is detailed in the GUI picture at the beginning of this chapter and in the MIDI mapping diagram below. For brevity, we will not re-list them here.



4.2.1 Sound Patches Details

To enhance sound customization, various types of synths have been designed in SuperCollider, with passion and care to their finest details. Below are their families and names[3]:

- **Rhodes**: RHODES1 and RHODES2.
- **Bass Synths**: BASSYN1, BASSYN2, BASSYN3, and BASSIMP.
- **Lead and Wave Synths**: LEADSCR, TRIWAVE, and SAWWAVE.

4.2.2 Presets Details

- **Mono Bass, Drum, and Synth**: By activating keyboard split, applying a bass (octave shift -1) on the left section and a different synth on the right section, you can simultaneously play a bass and a synth. The mono setting on the left side allows you to not worry about the sustain pedal release, enabling the player to focus on coordinating the pedal with only the right hand. A very interesting effect could be created adding a drum sequence and mapping the synth's right-hand cut-off frequency with glove or pedal.

- **Theremin-like Setup**: Unique results obtained activating the glove pitch bend mapping for both instruments, then connecting the glove's y-control to the cut-off frequency of the higher synth and the control pedal to the cut-off frequency of the lower one (-1 octave shift).

- **Synth and Drum**: This type of effect could be achieved with settings similar to the first preset on the list, but removing the keyboard split.

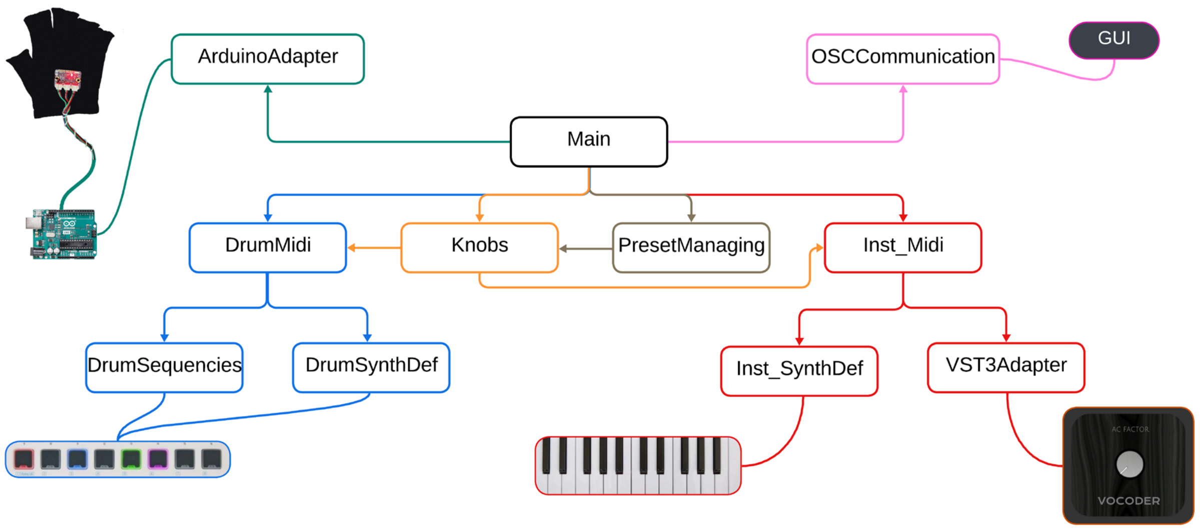
Note: The ***Keyboard Split*** can be activated/deactivated as follows:

- By clicking on knob 1 (top left on the Arturia). The system waits for a note pressure. Once done, that note becomes the split limit between the left and right sections of the keyboard.

- To remove the keyboard split, knob 1 should be pressed again. Now, both instruments will play simultaneously across the entire keyboard range.

4.3 Implementation Details

Implementing the system, we aimed at separating, as much as possible, functionalities into distinct modules and files. This, as shown in the diagram below, to ensure independent operation, improving maintainability and code reusability:



3 More detailed explanations of the single Patches are available on the project repository on GitHub

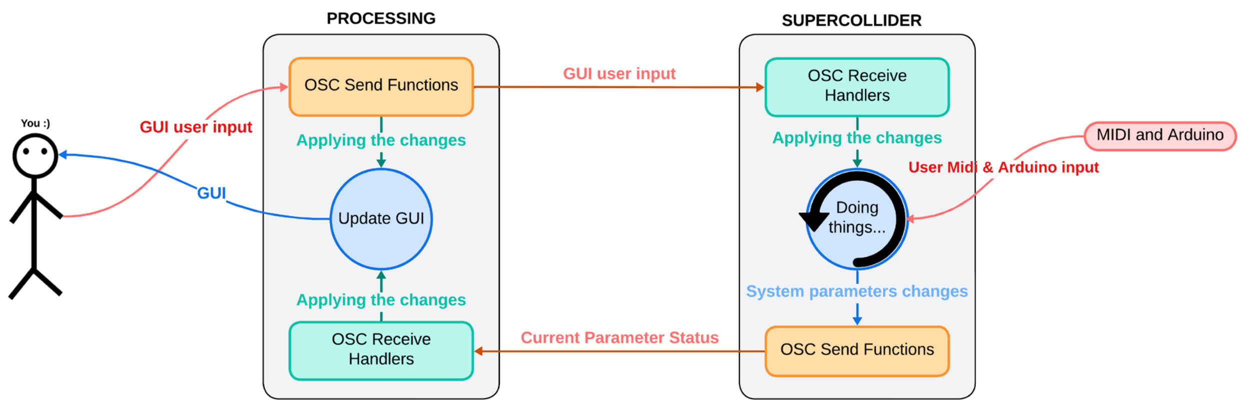
- **Blue Modules**: Manage the rhythmic section, handling MIDI inputs, defining the sounds to be played, and setting up drums sequences.
- **Red Modules**: Handle MIDI note management, including processing MIDI note inputs, defining the synth sounds, and forwarding notes to the vocoder.
- **Orange and Gray Sections**: For the management of knobs and presets. When a user selects a new preset, the knobs module updates all "knob values" (system parameters), which then propagate to all other modules. Additionally, the knobs module handles MIDI CC inputs.
- **Green Module**: ArduinoAdapter module, which receives hand inclination values and applies them to the system.
- **Pink Module**: OSCCommunication module, responsible for receiving user mouse inputs and refreshing the GUI to display the system status on the screen.

4.4 Interaction with Accelerometer and Arduino

Regarding the implementation of the glove, the input management is almost entirely handled in SuperCollider. However, the data received in SuperCollider is not the raw 3-axis accelerometer data. The accelerometer detects acceleration along the three Cartesian axes, whereas SuperCollider receives acceleration relative to the hand's orientation. To perform this conversion, we used formulas typically employed for managing drone orientation in flight.

4.5 Communication between SuperCollider and Processing

Processing creates a graphical user interface (GUI) that allows users to control musical parameters, which are sent to SuperCollider via the Open Sound Control (OSC) protocol. SuperCollider processes these inputs to produce audio and can send updates back to Processing for dynamic GUI adjustments. This interaction is illustrated in the diagram below:



4.5.1 Processing

Processing generates the GUI, including buttons, sliders, and knobs for controlling parameters such as volume, low-pass filters (LPF), instrument selection, octaves, control pedals, and presets. Excluding initialization and support functions, the code can be grouped into the following main sections:

• **Communication Management with SuperCollider**:

- Sending OSC Messages: The code handles sending OSC messages to SuperCollider to communicate various parameters. Functions send the names of selected instrument patches, control parameters such as mono status, volume, low-pass filter (LPF) frequency, BPM, vocoder volume, and the state of the vocoder GUI. Additionally, they send also selected octaves and accelerometer values.

- Receiving OSC Messages: When OSC messages are received from SuperCollider, the graphical interface updates accordingly. This includes updating knobs, sliders, and labels with received values and control statuses, such as the mono and GUI buttons.

• **User Input Management**: The graphical interface allows users to interact with various controls. Buttons enable changing instruments, octaves, control pedal settings, accelerometers, and presets. Each button has a listener that changes its background color and sends an OSC message to SuperCollider. Sliders and knobs allow adjusting parameters such as BPM, vocoder volume, and filter frequencies, sending the selected values to SuperCollider when modified.

• **GUI Update**: The code graphically updates the user interface in each frame.

4.5.2 SuperCollider

In SuperCollider, we can also divide communication management into main sections:

• **Receiving OSC Messages**: The code handles receiving OSC messages from Processing via OSCdef, which defines various handlers to process the received messages. These handlers update corresponding variables based on the messages received. They manage different aspects such as instrument selection, mono control, volume, low-pass filter, drum sequence tempo in BPM, octave selection, vocoder management, control pedal, glove mapping, and preset selection.

• **Sending OSC Messages**: The ~updateGUI function sends OSC messages to Processing, to keep the GUI updated. This function sends information about instrument volumes, low-pass filter cut-off frequencies, selected preset, selected instruments, mono status, drum sequence tempo in BPM, current glove status, vocoder volume, vocoder editor status, control pedal mapping, and selected octaves.

• **GUI Update Routine**: The ~guiRoutine ensures that the GUI is always synchronized with the current parameter states by executing the ~updateGUI function every 0.1 seconds.

5. Final Considerations

With the version presented here, we feel we obtained the goals we had in mind and, for some aspects, we have even exceeded our expectations: We can play our instruments and sing together, and we are very happy with the results. But, of course, we suspect we can be happier and we have already identified few areas where we will make further improvements. Therefore, for us, this Project is now just at "Version 1".