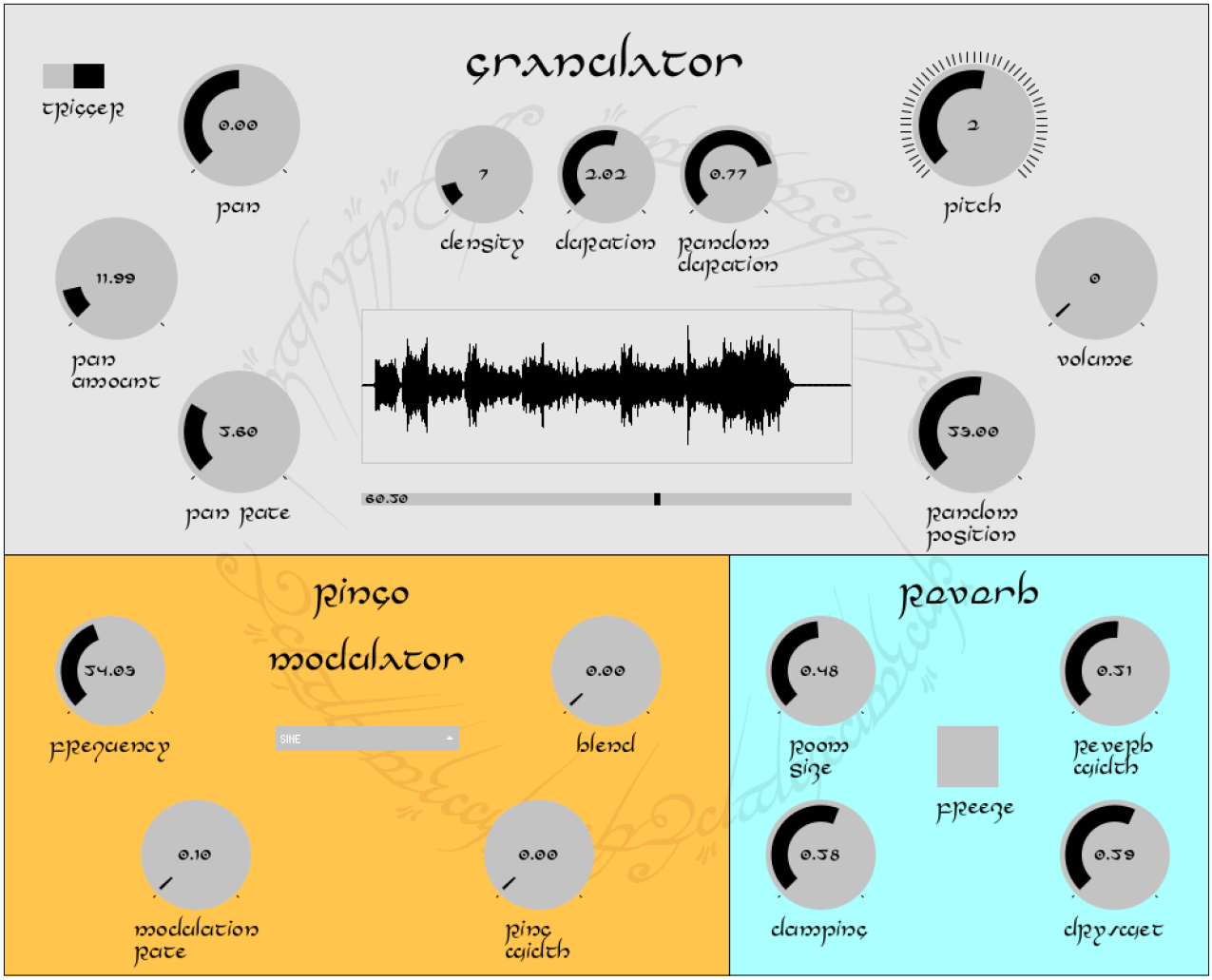


La Kunularo de l'Ringo

Chiara Auriemma, Veronica Coccoli, Anna Fusari, Mattia Martelli, Filippo Marri

27/05/24

Computer Music - Languages and Systems

GENERAL DESCRIPTION

Goal

The aim of our project is to create an expressive granular synthesiser whose output is sent to a set of effects composed by a simple reverb and a ring modulator, as the name suggests.

Most of the synthesisers or, more generally, most of the keyboard instruments do not offer the possibility to control the sound produced once the key has been pressed. Furthermore, in the context of digital instruments, the player cannot feel a real mechanical key response. These two conditions can lead to difficulties in expression and interpretation making the instrument play mass-produced sounds with no deepness or personality. To overcome this issue, we decided not to try to simulate something that wood and strings can surely do better, but to look for another way to make the player able to put down in music his own ideas. Drawing our inspiration from the "melodica" instrument, we came up with a mouthpiece in which to blow: as it happens when we sing, the velocity of air will control the amplitude of the output sound wave giving the possibility to modulate the intensity according to the musical thought.

Target

The synthesiser is thought for live performing. In light of that, the GUI is optimised to be simple and clear. Furthermore, the knob and the sensor control two crucial parameters as randomness and the type of trigger are, offering the possibility to the interpret to roam among different sound environments by means of a very accessible interface.

Operating Principle

Setup

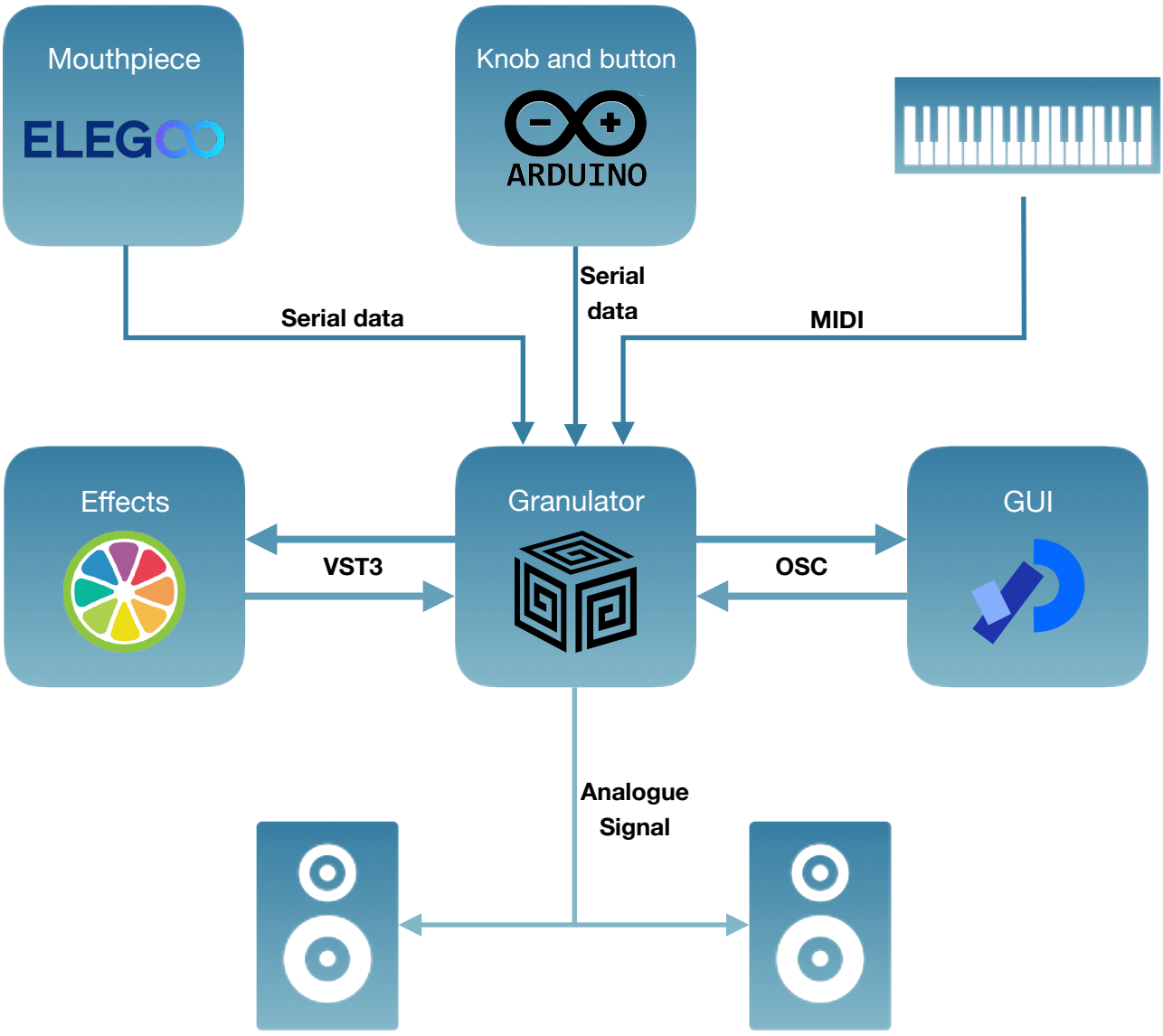
1. Connect the Arduino (optional) and Elegoo devices as it is reported at page 6;
2. Send the codes to Arduino (optional) and Elegoo;
3. Build the effect in JUCE;
4. Drag the VST3 file into the operating system's VST3 folder;
5. Open the SC File;
6. Run the sequence of instructions written to reduce the latency, run the server, read the audio file, initialize the busses and run the plug-in section.
7. Run the SynthDef function and, now, we are ready to run the granulatorSynth. Run everything that is contained into the MIDI section (except for the instructions that disconnect the device), everything inside the Processing section and the lines contained in the Arduino section (once again, except for the instructions that disconnect them).
8. Open the processing file and open the GUI.

Play

Once the system is set up, we are ready to play! By pressing a key on the keyboard we set the pitch of the grain whose intensity is modulated by the speed of the breath into the mouthpiece. The position in which to extract the grain from the song can be set by the slider directly below the sound wave. The knob set the randomness of the grain's extraction position and the button the type of trigger used to drive the granulator.

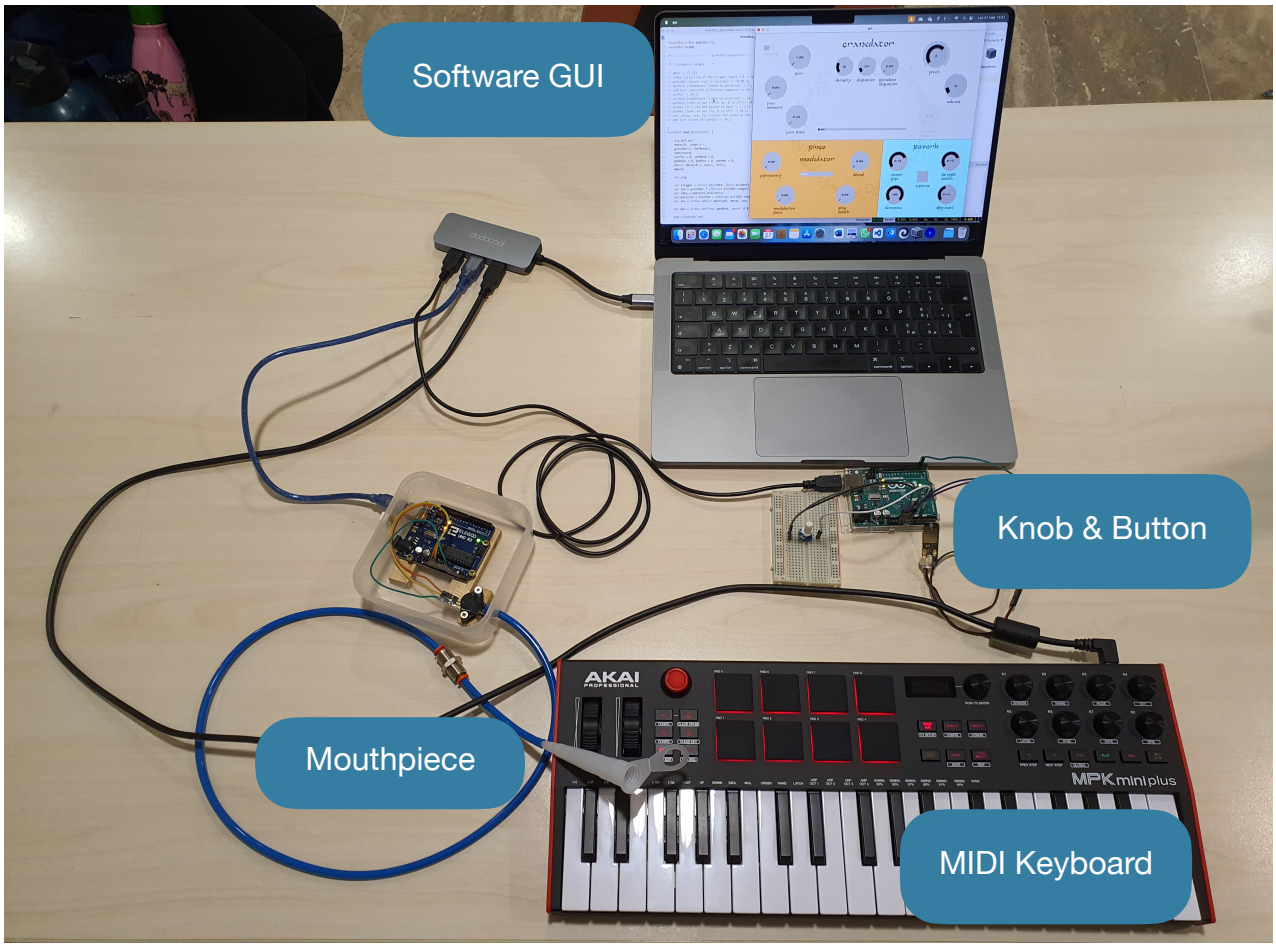
At the end, we run the `granulator.free` instruction and the functions contained in the "stop connection sections" to stop the application.

Block Scheme



MUSIC ENGINEERING - POLITECNICO DI MILANO

Picture



CODE OVERVIEW

SuperCollider

The core of the code is written in SuperCollider, therefore all the modules refer to the SuperCollider's script that behaves as a dispatcher.

The main part of the script is obviously the SynthDef, called `\granulator`. It is written recalling the structure of a class: firstly the arguments, and then the actual functions. Let us now go through them.
The arguments we choose are nothing but the classic parameters that a granulator needs: the density of the grains (grains per seconds), an `index` to choose how to generate the grains, two duration parameters, a variable to control the pitch, two parameters to control the position from which the grain is extracted from the track, three parameters to handle the pan and a set of parameters to control the ADSR. According to their role, they are properly used to define some variables given as arguments to the UGen `GrainBuf`.
Since the synthDef is based on this function, we dwell for a while on its arguments to see how the synthesiser is characterised.

- The first parameter is set to choose the number of channels of our synthesiser: in our specific case is set to `2` in order to implement a stereo system.
- `trigger`: by selecting the value of `index`, the user can choose whether to generate the grains as a series of periodic impulses or chaotically. `trigger` is basically one of the two functions according to the selection made.
- `len`: length of the grains evaluated as the result of a deterministic component and a random one. The former is given by two parameters that the user can choose (`grainDur` and `durRand`) while the latter is given by a random function. The length will be given by the product between `grainDur` and a random value between $\frac{1}{durRand}$ and `durRand` itself. Formally, it is a sort of variance controller. This means that the higher the `durRand` value is, the higher is the randomness.
- `buf`: buffer that contains the song from which we extract the parameters.
- `rate`: to set the pitch.
- `position`: where we pick the grain from the song. This time again, a component of randomness is introduced. By increasing the parameter `varRand`, we increase the uncertainty area around the position expressed by the parameter `varRand` where to pick the frame.
- The next parameter chooses the type of interpolations to use, we opted for the standard one.
- `pan`: the pan is implemented as a sine-wave controlled, in turn, by three parameters: `panRate` that sets the frequency, `panAmt` that sets the amplitude and `panPos` to set the amplitude offset.
- The last four parameters are, respectively, the envelope related to every single grain (Gaussian), the maximum number of grains that can play together (512) and two neutral parameters for the multiplication and sum functions.

The signal is then multiplied by two an ASDR envelope and to an amplitude to regulate the intensity.

Now, the MIDI section: we highlight that we intentionally decided not to define any `~noteOff` function as we do not want to implement a MIDI keyboard but simply to give indication to the system about pitch. We notice in fact that `~noteOn` simply choose the pitch by setting the `\semitons` parameter.

After that the Processing section where we standardly modify the parameters.

Finally the Arduino and Elegoo connection. Inhere, we connect the doors (`~port1` and `~port2`) to the twos devices. The former for Arduino, the latter for Elegoo. Then we read the value from `~port1` by using a routine where the data is priorly ascii-converted and then stored in a variable, and a second one that retrieves the `~breath` data, the velocity of the flux of air that flows into the mouthpiece. A third routine, set the granulator's parameter according to the values read by the two devices. At the end, we disconnect the ports.

Juce

The Juce section is delegated to handle the effect section. The main body of the effects is written in the `pluginProcessor` script. There we start by defining the parameters we need:

Ring Modulator:

- `NAME_RATE` the frequency of the LFO;
- `NAME_FREQ` the frequency of the carrier sinusoid;
- `NAME_WF` the LFO waveform used to modulate the carrier sinusoid (`Sine`, `Step squared`, `Squared`, `Random`, `Envelope Follower`);
- `NAME_WT` modulation intensity;
- `NAME_DW` dry-wet.

Reverb:

- `NAME_RS` reverb's room size;
- `NAME_DM` reverb's damping;
- `NAME_RDW` dry-wet. This parameter merges the dry and wet parameter of the class we use;
- `NAME_WR` reverberation intensity;
- `NAME_FR` freeze mode. This last parameter, when its value is more than 0.5, put the reverb into a continuous feedback loop.

Jumping over the default and the standard functions, we go directly to the `processBlock` to see how the effects are applied. In this function, we simply store in the variable `dryWet` the samples of the signal we want to process, we prepare the samples of the two oscillators and we call the `applyRingModulation` in order to apply the ring modulation to the input buffer. Obviously, there is no ring of diodes, the effect is implemented digitally according to the following equation:

$$output[n] = input[n] \cdot carrier[n]$$

In the code, this is done by calling the function `multiply` for every channel.
Once the ring modulation is completed, we merge the modulated (wet) signal to the original one (dry) in a percentage that is blended by the parameter `NAME_DW`. At the end, we apply the reverberation calling the namesake function `ApplyReverberation`. As we said before, inside this function we make the two parameters dry and wet become one by inversely connecting them. This allows the user to control the effect more easily without getting confused by too much pernickety features.

Processing

As we want our interface to give a sensation of organic unity and compactness, we decided to realise it completely in Processing. We opted for a simple and minimal design except for the font: hinted reference to the Lord of The Rings. The interface is divided in three parts: the two on the bottom are devoted to the effects while the top one to the granulator. All the parameters are adjustable by means of knobs. Furthermore, two toggles are implemented: one to select the type of trigger and the other one to switch either on or off the reverb's freeze mode.
To visualise where the grain is picked from the file, a wave-form is plotted. It is placed in the middle of the granuliser section, directly atop the slider we use to choose the position in which to extract the grain.
The implementation is quite straight-forward: after having imported all the libraries, we define the variables and the functions we need.

Arduino and Elegoo

To manage the hardware part, we wrote two different scripts for each device. We decided to use two different devices as the Elegoo one was recycled from another project where some of the pieces were glued on a shell in which the device was contained. In order not to break everything, we decided to use them separately.
The two codes are extremely simple: they communicate with the sensors and returns the value relative to the sensors. Let us see how these values are used:

Knob potentiometer

The potentiometer returns one value from 0 to 1023 that is mapped into `\varRand`, the variable that handles the randomness of the position in the track from which the grain is picked around a pre-set reference point.

Touch button

The touch button simply switch from one type of trigger to the other modifying the variable `\index`.

Mouthpiece

The mouthpiece evaluates the speed of the air that is blown into it by mapping it on a scale from 0 to 100. This value will control the `\amp` of the output sound wave.

RELEVANT PORTIONS

We want now to dwell on particular aspects of our script and hardware interesting to see in more details.

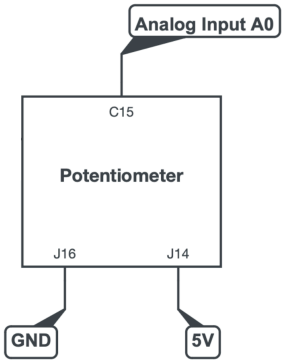
Processing

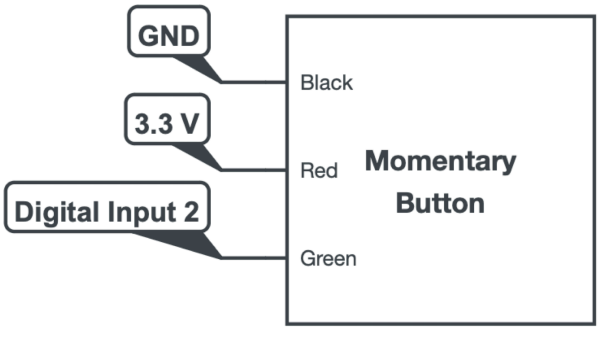
Having the possibility to change some parameters by means of both the GUI and the Arduino system we bump into the following issue: if we modify a parameter from the physical knob, we can visualise the change on the GUI by simply varying the position of the element that represents the knob on the GUI. On the other hand, if we change a parameter on the GUI, there is no way to make the physical knob change accordingly. There were several solution that could be implemented: we decided to deactivate the GUI's input for those elements as soon as we connect the Arduino boards. This solution was the lightest and the simplest for a user to understand.

Arduino and Elegoo

One particular feature of our project is the mouthpiece that modulates the intensity, so the energy, the amplitude, of our sound-wave. Let us see how the measuring system is implemented. A tube is connected to a pressure sensor, in our case the MPX5010 by NXP Semiconductors. We highlight that, in order to make the system sense the atmospheric pressure, we drill a small hole in the pipe. The pin 1, 2 and 3 of the sensor are respectively the output, the ground and the 5V power supply. Hence, the last two pins are connected to the Elegoo's power pin while the first one to the Elegoo's analog input. The output is then mapped into a range from 0 to 100.

For what it concerns the Arduino device, the connection of the sensors has been done accordingly to the scheme below.





FURTHER POSSIBLE IMPLEMENTATIONS

In the following section, we list some of the ideas we could add to enrich the project.

SuperCollider

- A first very interesting feature to implement could be the possibility to make the pan casual for every grain. This will extremely enrich the spatiality of the granulator generating an immersive and enticing sound, perfect for noisy-industrial music or, also, to look for an overwhelming sensation like the wall of sound is.
- Then, as the set up is a bit intricate, it could be useful to define a simple function to boot the entire system without doing it manually.

Processing

- In order to enrich the timbre expression possibilities of the grains, we could add a little graph on the GUI where to "draw" their envelope.

Arduino

- Another interesting hardware device to add to our project could be an XY touchpad to control the two most characterising reverb's parameter: on one axis the dry-wet, on the other one the room size.
- To have a completely physical interface, we could add more knobs to control each parameter. Furthermore, with the same aim, we could exploit the knobs on the MIDI-keyboard.