POLITECNICO
DI MILANO

Harmonizer effect

CMLS Homework 1 - Assignment No. 3

Manuel Alejandro Jaramillo Rodríguez, 991288
Marcello Grati, 102707
Maria Gracia Fernandez, 990109
Silvia Pasin, 101090
Natasa Popovic, 991034

Milan, 2022

1. Introduction

The goal of the assignment consisted in the creation and design of a vocal harmonizer with an interface which controls its parameters. It was proposed to perform its implementation in the *SuperCollider* program, designing the relevant code and considering the different minimum requirements for the completion of the assignment, as well as the added aspects interesting for its implementation.

The development of the assignment has been carried out as follows. An initial approach to the code was made with all the minimum requirements requested. These consisted of taking an external input (soundcard or microphone), pitch shifting the input, adding it back to the original signal in order to create harmonization, and, finally, implement a graphical user interface which allows the user to adapt the values of different parameters of the harmonizer. Once the minimum requirements have been satisfied, different options were contemplated in order to add more functionalities to the project:

- **Pan**: Possibility of orienting the sound through the speakers/headphones for the different harmonized voices. It can be only on the left, only on the right or mixed according to the user.
- **Volume**: Variation of the intensity of each sound, both of the original sound and of the harmonized ones.
- **Reverb**: additional added effect that has within it different parameters to obtain the desired amount, such as *Wet Amp*, *Room*, *Wet/Dry* and *Damp*.

Once the code was implemented, an interface was designed that would cover all of the possible user needs according to the implemented functions. However, in terms of design, what was valued the most was the convenience, usability, and accessibility of the harmonizer parameters having the average user in mind.

2. Development of the assignment

The final implementation of the code uses five different synthesizers: two for handling the input signals, two for implementing the harmonizer, and one for applying the final reverberation effect. Those five synthesizers are then combined in two different functions that control the correct flow of the signals to be correctly harmonized, filtered, and mixed.

The harmonizer is mainly based on two *UGens* from *Super Collider*: *"Pitch"* and *"PitchShift"*. *"Pitch"* is used to track the pitch of the original signal in order to correctly harmonize it. *"Pitch"* is an autocorrelation pitch follower, which is executed every fixed amount of samples and returns the tracked frequency of that window and a boolean that states if a peak was recognized or not. In order to determine whether a peak is detected or not it uses the argument *"ampThreshold"* and it compares the peak to peak ratio with this threshold: if it's higher, then a Pitch is recognized. *"PitchShift"* was used to create the harmonized voices. It is a time domain pitch shifter, whose arguments and implementation will be briefly explained in the following sections.

2.1 Synthesizers

- **“\mic”**: This synth receives the signal from the input hardware of the computer (one or two microphones). If there is only one bus taking input from the hardware, then it duplicates the signal to make it stereo. Then, it modulates the received signal by a constant value (volume parameter) and pans the signal by a desire quantity. Finally, the modified signal is sent to a selected output bus.

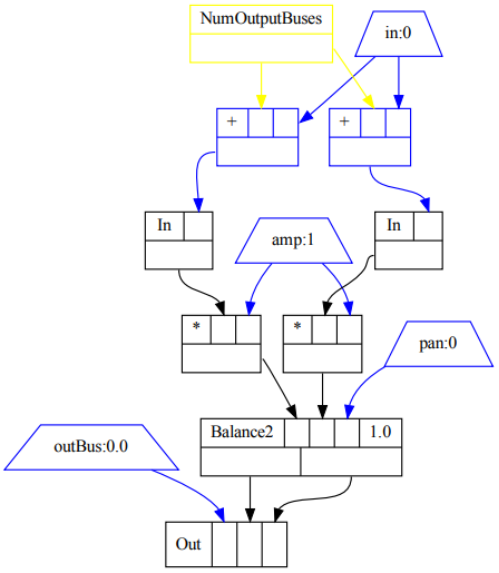

Image 1.- “\mic” synth

- **“\audio”**: This synth takes a preloaded sound that is saved in a buffer, modifies its amplitude and panning by a desired value, and sends it to a selected output bus.

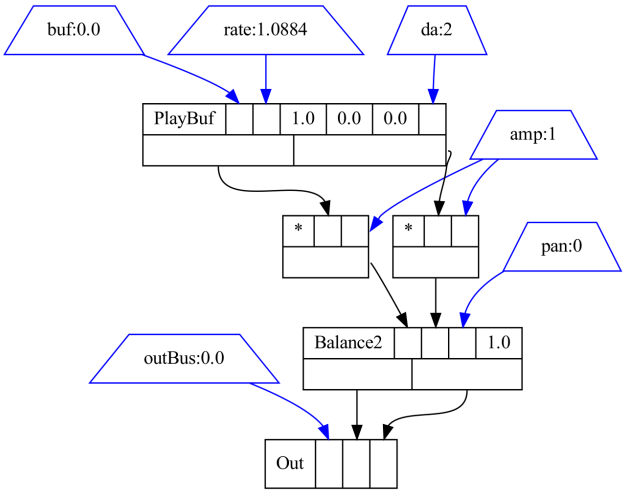

Image 2.- “\audio” synth

- **“\harmonizer”**: This is the main synth regarding the harmonizer, it receives a stereo signal from a selected bus, then uses the *UGen ”Pitch”* to do pitch recognition on the signal, the recognized pitch is associated to one note of the chromatic scale to assign to it the correct interval to be harmonized with. Then it uses the UGen *“PitchShift”* to pitch-shift the signal by the desired amount and sends the pitch shifted signal to the selected output bus. It is worth mentioning that in the last implementation, our harmonizer creates four different voices: third up, fifth up, fourth down, sixth down, so actually it sends each of the voices to a different output Bus, so we can mix each of the harmonizing voices as we want (deeper description will be done in the following section).
- **“\harmController”**: This is an auxiliary synth to the harmonizer, it takes one of the voices created by \harmonizer and applies to it the desired amplitude (volume) and panning, then the modified signal is sent to the selected output Bus.

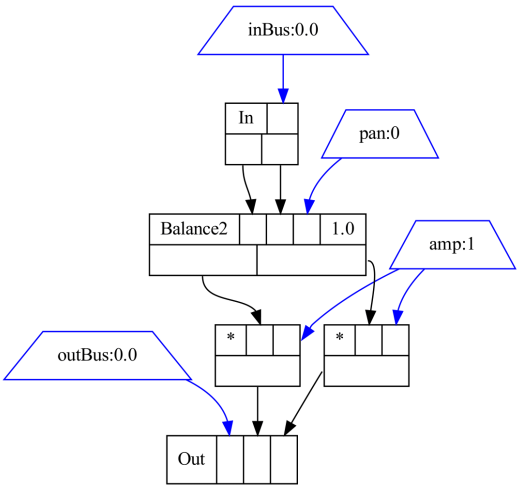

Image 3.-“\harmController” synth

- **“\rev”**: This is an effect synth, it adds reverberation to the input signal by using the *UGen “FreeVerb2”*, then the filtered signal is sent to the desired output Bus.

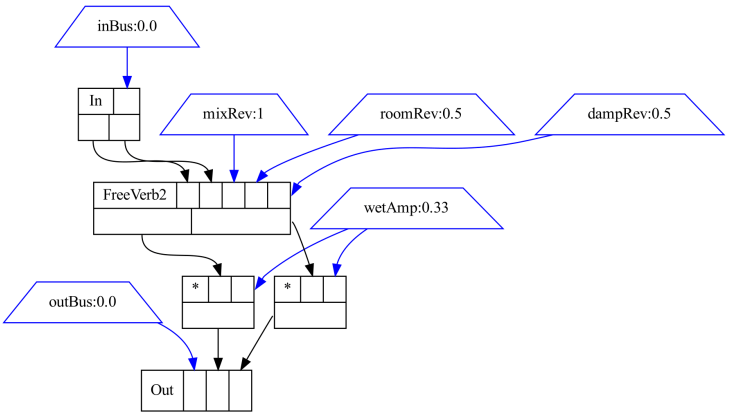

Image 4.-“\rev” synth

2.2 Architecture

Having all the synthesizers correctly working, two main functions were created *("f "*and *"g"*) to combine these synthesizers into the final harmonizer. Both functions have the same structure, the only difference between them is the fact that *"g"* uses as input the signal received from the microphones, and *"f"* uses a signal loaded to Super Collider from an audio file and allocated into a buffer.

The harmonizer has been divided into four different groups: one for handling the input signal (microphone or buffer), two for handling the harmonizer, and finally, one for handling the effects (just the reverberation in this first approach). The node tree of the function *"g"* is shown in the following figure (Image 5.):

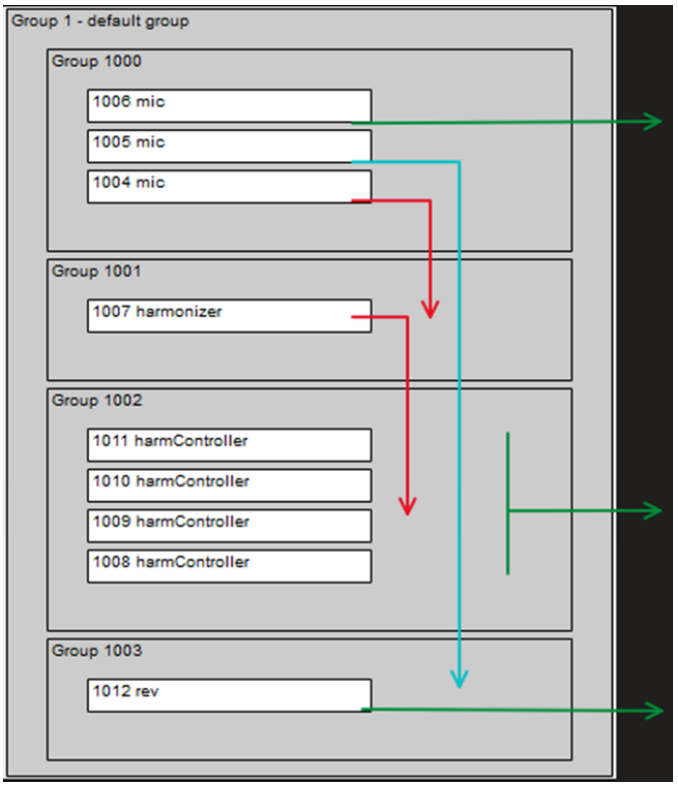

Image 5.- The node tree of the function *"g"*

As it can be seen in the node tree, three different copies of the input signal are generated: one is sent directly to the output hardware, the next one is sent to the reverberation bus, and the last one is sent to the harmonizer bus. The reason for doing it this way is that it makes the final mixing stage easier, since we will have all the voices, effects and original signal in different busses and groups. The harmonizer receives a copy of the input signal and creates all the voices, each of them is sent to the *"harmController"* group where they are mixed and finally sent to the output hardware. Finally, we have the effect group, which for now only includes a reverberation effect, the input signal is received, filtered, and sent to the output hardware.

2.3 Harmonization

The harmonizer is a type of pitch shifter that combines the pitch-shifted signal with the original to create a two or more note harmony. The pitch-shifted signal is obtained by increasing or lowering the pitch of the original signal by a preset diatonic interval. In order to do that, knowing the musical key is essential to choose the right amount of semitones to add or to subtract, so the user is asked to enter information about the key as an input. Once that information is present, it's easier to just work with intervals, instead of notes, by associating a number from 0 to 11 to them, and then decide the pitch shift for each element of the scale.
Example in C major with the shift expressed in semitones is shown in the table below (Image 7.).

	C	C#/Db	D	D#/E	E	F	F#/Gb	G	G#/Ab	A	A#/Bb	B
	0	1	2	3	4	5	6	7	8	9	10	11
6th below	-8	-8	-9	-8	-9	-8	-9	-8	-8	-9	-8	-9
4th below	-5	-4	-5	-4	-5	-5	-6	-5	-4	-5	-5	-6
3rd above	+4	+4	+3	+4	+3	+4	+3	+4	+4	+3	+4	+3
5th above	+7	+8	+7	+8	+7	+7	+6	+7	+8	+7	+7	+6

Image 7. - Shift expressed in semitones for "C major" key

It was decided to implement this function inside the definition of the synth \harmonizer at line 27. The first thing to do is to take the input signal and to recognize its pitch.

```
>     in = In.ar(inBus,2);
>     # freq, hasFreq = Pitch.kr(in);
```

Then we convert the frequency in Hz into a MIDI number, so that we can subtract the value of the key and later, using the %, we obtain a note number from 0 to 11.

```
>     note = freq.cpsmidi.round(1);
>     tempNote = note - key;
>     tempNote = (tempNote % 12);
```

We assign a different pitch-shift value to Pitch array (one for each of the 4 harmonizations) based on the value of tempNote, with the same pattern as the table before.

```
>     pitch = Select.kr(tempNote+1 < 1, [[-8, -5, 4, 7], pitch]);
>     pitch = Select.kr(tempNote+1 < 2, [[-8, -4, 4, 8], pitch]);
```

…

Once we have these values, we can pitch-shift the signal, adding other effects like time and pitch dispersion.

```
>      harm1 = PitchShift.ar(in, pitchRatio:pitch[0].midiratio,
pitchDispersion: dispr, timeDispersion:disp);
       …
```

2.4. Interface

The interface design is shown below:

As we can see, in the upper part of the image we find the different voices that take part in the harmonization as well as their volumes and the pan of each one. On the left, it is placed the original input of the signal, and then the different voices.

In the lower part we find the reverb effect and its different parameters to be configured by the user (from left to right): volume of the reverb (WetAmp), room, wet or dry and damp.

Finally, on the right side there are the different options to choose the input, the stop and some parameters of the harmonizer and the shift of the different voices. Here we would like to highlight the two parameters that are handled with a slider. The first one is pitch dispersion (pitch disp), which makes the second voice vary from the original, making them not exactly the same even though it is shifted, giving an error range in the frequency for the harmonizing voice. Time dispersion (time disp), does the same but in time, making the second voice not sound at the same time but a little before or a little after.

3. Conclusions

For the realization of the minimum requirements, it is worth mentioning some difficulties that have encountered during its first approach, as well as its subsequent development and on which we had to take special care for its operation. Among them, the buses and their good arrangement was something fundamental and that we had to take into account at all times so that it could sound correctly, because if the same bus was assigned to different outputs it gives rise to problems and it is not heard correctly. Also, the signal flow was a little bit difficult to implement as well as the addition of effects to the previously mentioned minimum requirements once we completed the main implementation of the code.

Another difficulty that was encountered was how to harmonize correctly. The easiest approach would have been to just choose a fixed semitone offset, regardless of the pitch and of the key, obtaining basically a major chord for every note played. Instead, by implementing the harmonizer in this way with pitch recognition and having a knowledge of the key, we avoid adding out of tune notes, even when the original signal is off-key.

Overall, the work done has been quite satisfactory and we are happy with the obtained result. Our main goal was to get to understand SuperCollider and its structure as well as performing the assigned task as completely as possible. We believe that we have achieved it as we have completed successfully all the goals that we had at the beginning of the assignment.