CMLS Homework 3 – Voice Harmonizer

Samuele Bosi, Marco Ferrè, Philip Michael Grasselli, Simone Mariani

June 1, 2020

Contents

1 Introduction

In this third and last assignment, our task was to create a **voice harmonizer**, both the effect and the mobile control interface, plus another couple of effects (tremolo and reverb). We used **SuperCollider** to implement the former, and **TouchOSC** for the latter: a voice coming from any microphone connected to the computer could pass across this system.
For the repository, this is the URL for GitHub: `https://github.com/marcoferre/CMLS-HW3`.

2 A Glance of Harmonics

The choice of the number and the interval between notes was not as straightforward as we thought at the beginning, since the effect has to be realistic. The simplest idea was to consider only octaves, as it is typical of almost all instruments, and this is evident whilst applying the Fourier Transform of a sound excerpt of a note played by an instrument.
To give a *minor* or *major* harmonization, the minor and the major **third interval** was taken into consideration, as the musical theory clearly explains.

3 The Harmonizer

As previously touched upon, the voice harmonizer is composed by two parts: the SuperCollider algorithm and the TouchOSC GUI for touch devices.

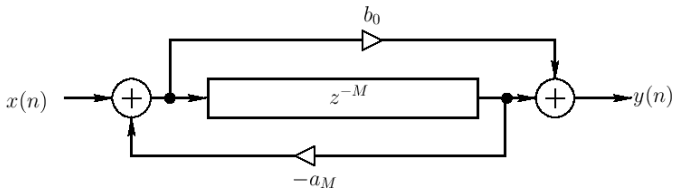

Figure 1: The Schröder all-pass filter

3.1 The SuperCollider Code

After booting the `localhost` server with the `s.boot` line, we define the arguments related to the knobs of the several effects inside the pad: the pitch shift, the array referred to the harmonics, the room effect for the reverberation, the X and Y coordinates for the tremolo (frequency and amplitude), and the attack.

The sound is captured from the microphone through the command line:

```
in = Mix.new(SoundIn.ar([0,1]));
```

It follows that the amplitude is being deemed by the `Amplitude` class for each sample, and its value brought to the `amp` variable.

We, then, define the number of harmonics by setting up the array of partials:

- if we're moving by octaves, we just divide or multiply by 2^k, with $k \in \mathbb{Z}$:
- if we're considering also the minor thirds, we add a $\sqrt[3]{2}$ correction factor;
- if major thirds are needed, we place a $\sqrt[4]{2}$ correction factor.

All the seven harmonics are naturally stored into the same number of variables `v1`, `v2`, etc.

3.1.1 The `PitchShift` Class

The frequency translation occurs thanks to the `PitchShift` class: several parameters could be set, including:

- the **width** of the frame applied to the input signal;
- the **translation rate** of the harmonics with respect to the fundamental frequency;
- the frequency **ripple** of the harmonics;
- the temporal **delay** of the harmonics;
- the **amplitude** of the output signal. Attack and release values are picked up by the previously declared `amp` variable.

3.1.2 Reverberation

The class `FreeVerb` lets you design the reverberation effect, in which you can set up a couple of parameters: the dry/wet level and the room effect. This was our choice, since it was simpler and very effective. Another way to build this spatial effect could be to implement a **Schröder all-pass filter**, which is shown on figure 3.1.2, and it's transfer function is:

$$H(z) = \frac{b_0 + z^{-M}}{1 - a_M z^{-M}} \tag{1}$$

Figure 2: The Graphical User Interface of the Voice Harmonizer in its final version

where $b_0 = a_M^*$, to let it become a real all-pass filter. In SuperCollider, the class `AllPassN` implements this function.

3.2 The GUI

The software **TouchOSC** was used for the visual part of our synthesizer, and it's shown in Figure 2.

The controller receives the harmonizer messages of `pitchFollow1`, and we declare two variables related to the input and the output signal addresses. Once completed these first tweaks, we proceed to define the default values for all the buttons, in order to initialize the GUI.
All the `OSCdef` associated to all controls are very similar and straightforward in their functionality, as they receive and send messages to the SuperCollider's post window. One last thing related to this interface is that major and minor buttons are mutually exclusive.

4 What's Next?

A possible implementation of our commitment could be – in a hardware fashion – to transfer this interface to a Raspberry-controlled pedal board; by using a LCD display we could manage the currently applied settings, and by using an expression pedal and foot-switches we could control all the other available effects. All this stuff may be useful in case of live performances.