CMLS Homework 3
Assignment 4 - Voice Harmonizer

Group 18
10751302 - 10531235 - 10314186 - 10703215

2nd Sem. A.A. 2019/2020

GitHub Repository

https://github.com/BroCorra/hw3-cmls-voiceharmonizer

Introduction

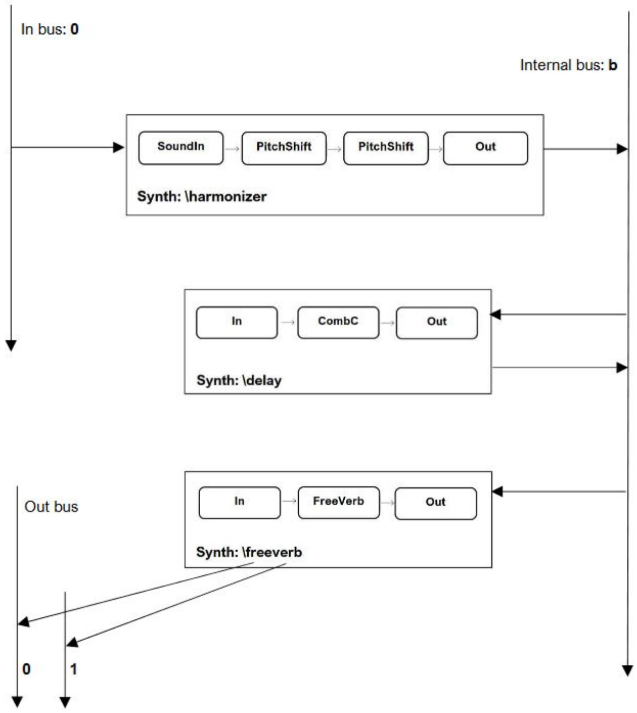
The assignment that has been provided to our group consists in an implementation of a vocal harmonizer. A vocal harmonizer pitch shifts the input voice track and add it back to the original unpitched audio to create a two, three, or more note harmony.

Our implementation adds up to two voices, following 4 different type of chord. We implemented the processing part (taking the input from the mic and processing it) using SuperCollider while for the Graphical User Interface we used Processing 3.

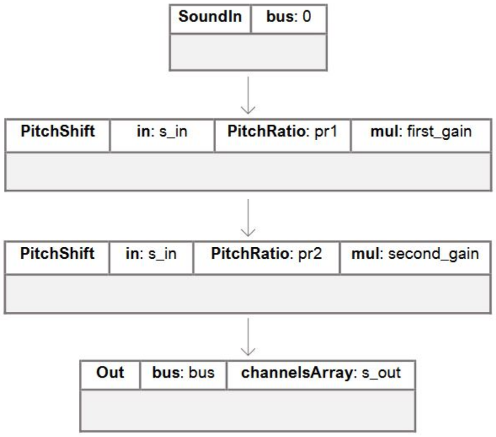
SuperCollider

In SuperCollider we take care of capturing the input from the sound card and then process it in order to produce the desired harmonization. We take the opportunity to add also some more effect, in order to create a more powerful harmonizer. In particular, as can be seen in the Bus Management graph (Figure below) we implemented three *SynthDef*, each one taking care of one ”processing step” and all part of the same *Audio Bus*. We implemented an internal bus in order to make use of the powerful SuperCollider bus implementation. We were aware of the fact that if the SuperCollider receive into a specific bus more than one signal, it automatically *mix* them, as a real hardware mixer would do.
The processing chain is:

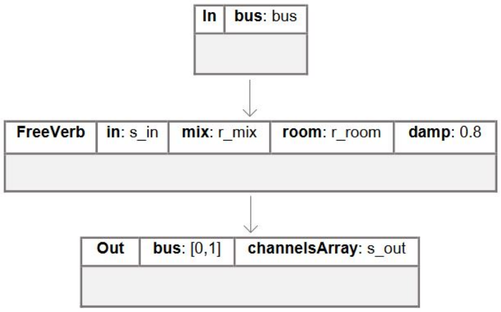
- **\harmonizer**
  This *SynthDef* takes care of reading from the sound card input, creating the two voices by pitch shifting the original signal accordingly to the settings and then summing the three signals. Finally it outputs the resulting signal on the internal *b* bus. The harmonization part is implemented by using the PitchShift UGen method.
- **\delay**
  This *SynthDef* takes the signal coming from the internal *b* bus, where a signal from previous stage (the harmonization) is present, process a delayed signal using the UGen *CombC* and then sum it to the dry signal. Finally it outputs the resulting signal again on the *b* internal bus.
- **\reverb**
  This *SynthDef* takes the signal coming from the *b* bus, where we expect the signals from the previous stages (the delay), applies a FreeVerb reverb method and finally output the resulting processed signal into the *default SC output* busses, noted as [0] (left channel) and [1] (right channel).



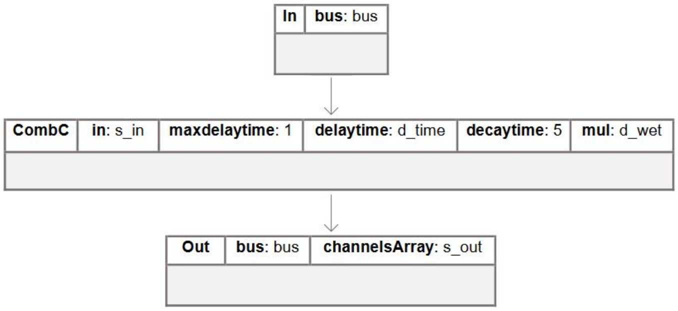
Overall bus management



Harmonizer Ugen graph



Reverb UGen graph

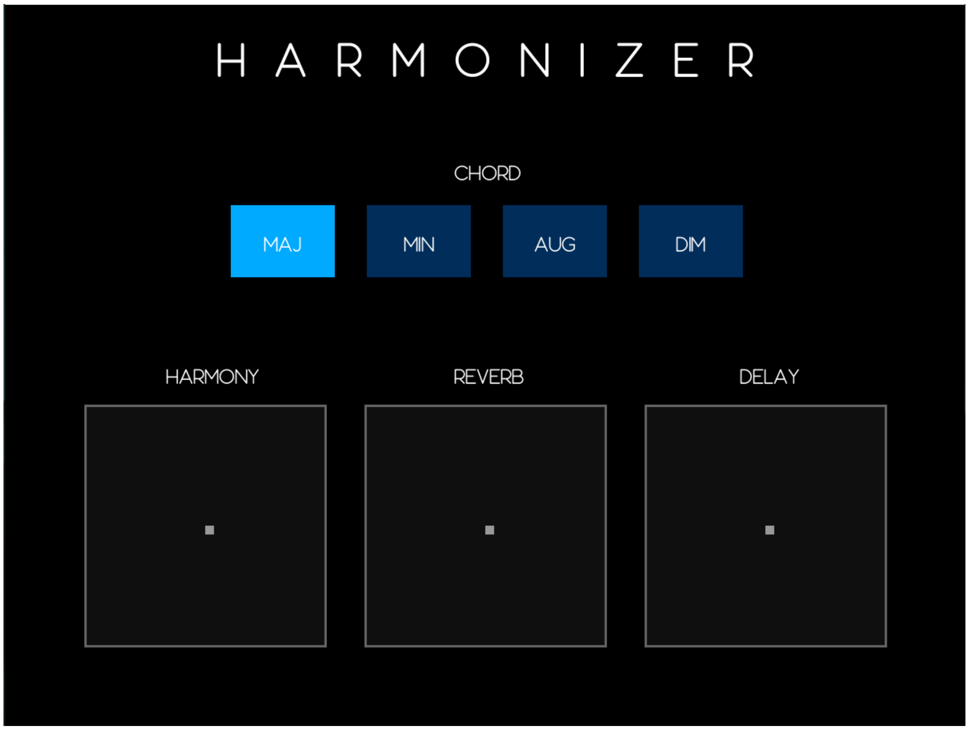


Delay UGen graph

Interface

As already said in the introduction for the user interface we decided to use Processing 3, along with its libraries OscP5, NetP5 and ControlP5 for managing OSC messages and GUI elements.

On the GUI the user can select the type of chord the two voices will follow in order to produce the harmonization by pressing the respective button and can control some parameter thanks to *2D pad controllers*, containing a cursor that will change different parameters based on the direction in which is dragged (Figure below). The values grow from left to right and from top to bottom.



Graphical User Interface in Individual mode

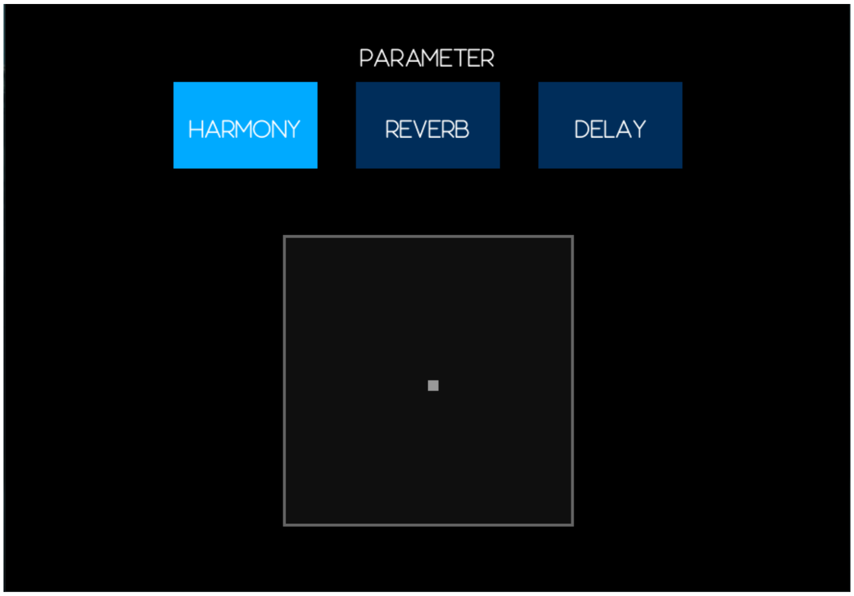
More in details the four controllable elements are:

- **HARMONY**
  - X-Axis: changes the gain of the first voice (the third of the chord).
  - Y-Axis: changes the gain of the second voice (the fifth of the chord).
- **REVERB**
  - X-Axis: changes the dry/wet balance.
  - Y-Axis: changes the room size.
- **DELAY**
  - X-Axis: changes the wet signal gain.
  - Y-Axis: changes the delay time.

Furthermore the user can decide to use the Harmonizer in two different modalities:

- **INDIVIDUAL**
  As can be seen in the Figure above in this modality the user has access to all the parameters and can directly modify them.
- **COLLABORATIVE**
  The user can collaborate with other people to play with the Harmonizer. In particular the first person to open the Collaborative mode will be the *Host*, and will have access to the chord type selection.

  In this modality, every user have the access to one pad controller at a time (Figure below) and his changes will affect the output. The recommended way of using this modality is to have three users, each of them modifying a different parameter.



Pad controllers access in Collaborative mode

OSC

The OSC messages are mainly sent by the Processing GUI to SuperCollider in order to change the values whenever a parameter is changed.

More in detail there are 5 different type of OSC messages sent:

- **/chord**
  This message contains the intervals (in semitones) of the chord type selected and is sent every time the chord type is changed. Once received SuperCollider takes care of setting the respective parameters (pitch shift ratio) of the vocal harmonizer accordingly.
- **/harmony**
  This message is send by the Harmony pad and contains the coordinate of the cursor on the pad. Before being sent the coordinates are scaled in order to be in a range of 0:1. Once received SuperCollider takes care of setting the parameters corresponding to the two coordinates (see Interface for the mapping between coordinates and parameters).
- **/reverb**
  Same as */harmony* but sent by the Reverb pad.
- **/delay**
  Same as */harmony* but sent by the Delay pad.

Network

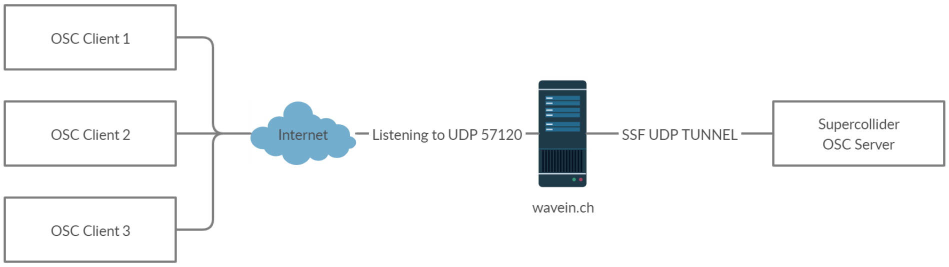


Figure 3: Network schema

In order collaborate over the internet we build the infrastructure described by the schema in the Figure, where we can identify three components:

- **OSC Server (Supercollider)**
  Supercollider act as OSC server, listening on port 57120 UDP, where the actual audio processing is done.
- **Internet gateway**
  A server that exposes to the internet the port 57120 (UDP) and forward all the traffic to the OSC server.
- **OSC Client (Processing)**
  Application for controlling the parameters of the audio processor, in collaborative mode the best would be to have 3 instances. The application sends OSC messages to the internet gateway.

Since the clients cannot communicate directly to the server we need an intermediate server (Internet gateway server) that accepts OSC messages and forward them to the actual OSC server, to do so the host that runs the OSC server connects to the gateway creating a tunnel. The first idea was to use the standard SSH tunneling, but since OSC is on top of UDP is not applicable, we therefore opted for the less known Secure Socket Funneling that natively support UDP tunneling.