HarMMMLonizer

Lelio Casale, Marco Furio Colombo, Marco Muraro, Matteo Pettenò

102827, 10537094, 10866647, 10868930

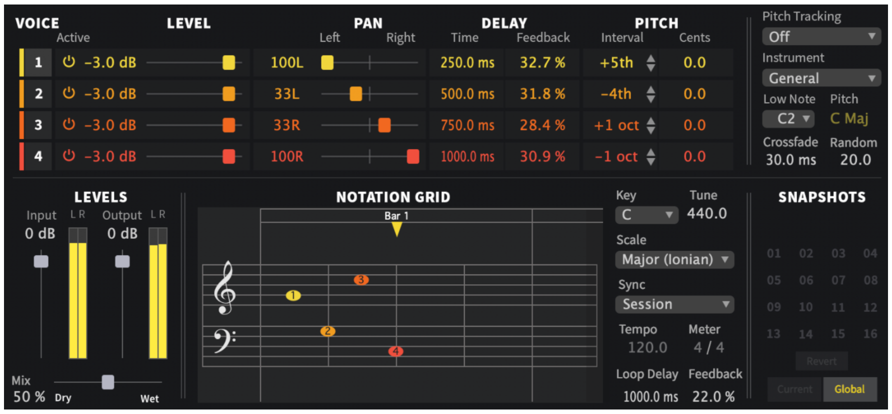
Abstract

Among several definitions, music is a category of perception as it involves cognitive processes. Harmony is one of the building blocks of Western Music, along with melody. Indeed, exploiting the vertical dimension of music, the musician can manage the mood of the song and convey specific emotions to the listener. Harmonizer is one of the most used audio effects within the music professionals community. The project presented in the following paper is focused on the implementation of such a digital signal processor through SuperCollider, including a graphical user interface to control the available parameters. Moreover, the software provides the user with an additional component devoted to delay effect to be applied individually on pitched voices generated by the harmonizer itself. Different feedback settings can be chosen by the musician.

1 Introduction

1.1 Basic Principles of the Harmonizer

The main structure of an harmonizer is based on many voices, usually three or four. Each voice is the pitch-shifted version of the input of a certain amount of semitones. Pitch shifting means multiplying every frequency of the input by the $n-th$ power (n is the number of semitones we want to shift the signal of) of a factor that represents the ratio between a note and the note that is one semitone below, which is the twelfth root of two (around 1.05946). As an example, if we want to obtain a voice shifted by a third (4 semitones), we have to multiply each input frequency by $(1.05946)^4$. Pitch shifting differs from frequency shifting: in frequency shifting a certain amount of Hertz is added to every frequency, with the result that the ratios between the fundamental frequencies are not maintained. Instead, in pitch shifting, the ratio between fundamental frequencies of the input is guaranteed by the fact that every frequency is multiplied by the same quantity. Pitch shifting is recommended for small intervals, usually maximum an octave or a little bit more, for bigger intervals we may introduce a distortion that affects the quality of the shifted signal. This is the reason why usually in a harmonizer we're allowed to shift the input signal by two octaves above (which means a factor of 4) and below (which means a factor of 0.25). Other features can be added to a harmonizer, also depending on the needs of the artist and the musical genre. Usually a gain knob is provided to the user, to adjust the volume of each voice. This choice is made both for an artistic reason and for a psychoacoustic reason. In fact, the human ear is not sensitive to all frequencies in the same way: sounds with a mid-high fundamental frequency (100 Hz to 4000 Hz) tend to be perceived as louder. So if we want to hear a low and mid-high frequency pitch shifted voice of the harmonizer at the same level, we need to set the gain of the low voice higher than the gain of the mid-high voice. Also a delay control and a pan control for each voice are usually provided to the user, the first one to add some delay effects, and the second one to have fully under control the spatial stereo direction of each voice.



VOICE	LEVEL		PAN		DELAY		PITCH	
	Active		Left	Right	Time	Feedback	Interval	Cents
1	-3.0 dB		100L		250.0 ms	32.7 %	+5th	0.0
2	-3.0 dB		33L		500.0 ms	31.8 %	-4th	0.0
3	-3.0 dB		33R		750.0 ms	28.4 %	+1 oct	0.0
4	-3.0 dB		100R		1000.0 ms	30.9 %	-1 oct	0.0

Pitch Tracking: Off
Instrument: General
Low Note: C2 Pitch: C Maj
Crossfade: 30.0 ms Random: 20.0

Input: 0 dB Output: 0 dB
Mix: 50 % Dry Wet

Key: C Tune: 440.0
Scale: Major (Ionian)
Sync: Session
Tempo: 120.0 Meter: 4 / 4
Loop Delay: 1000.0 ms Feedback: 22.0 %

Figure 1: An example of a four voices digital harmonizer - Quadravox by Eventide.

1.2 Purposes

The project aims to develop a real-time harmonizer, i.e. an audio-based software which provides the user with the possibility of creating a two or more notes harmony. The application was implemented through SuperCollider, a powerful engine for sound synthesis and algorithmic music composition which offers a sound-based, real-time and object-oriented programming language.

- Implementation of a pitch-shifter block which takes a signal as an input and returns the shifted version as an output. The software must support a maximum of three pitch-shifted voices.
- Implementation of a delay line block which accomplishes delay effect on the pitch-shifted signal, with the possibility of different feedback configurations. Each pitch-shifted voice is processed individually within this block.
- Implementation of a Mixer component so that the user can control the master volume and stereo pan of the final output.
- Development of a Graphical User Interface (GUI) in order to provide the user with the possibility of controlling the available parameters, such as Pitch Ratio, Delay Time and Feedback Mode.

2 Implementation

2.1 Software Architecture

The architecture of the whole software consists of two main components: the **audio processing block**, responsible for signal processing, and the **graphical user interface (GUI)** which implements the graphical component of the software itself.
The **audio processor** in turn is composed of three functional blocks which are the Harmonizer, a Delay Line and a Mixer. These blocks were implemented through SuperCollider language as SynthDefs (Synthesizer Definitions) with specific available arguments with respect to what parameters the musician needs to control through the GUI. An additional block was implemented with the purpose of collecting the input signal coming from the sound card and writing it on a specific bus so that the implemented synthesizers can read it and compute processing operations on the signal.
The **graphical user interface** shows three main sections, one for each harmonized voice so that the user can select the desired values for the parameters of the harmonizer and delay line. Furthermore, a mixer section is made available in order to allow the musician to control master volume of the final output and dry/wet balance between the clean input signal and harmonized voices.
As the software is designed, the server allocates the following mono audio buses:

- ~***inputAudioBus*** collects the input audio signal coming from the sound card.
- ~***pitchShiftedVoiceBuses*** is an array of three audio buses representing the pitched voices generated by the harmonizer.

- ~***delayedVoiceBuses*** is an array of three audio buses representing the delayed voices of the harmony.

In addition to this, three arrays of one channel control buses are used with the purpose of sending parameters values related to each single voice from the GUI to the different synthesizers instantiated on the server:

- ~***pitchRatioControl*** regards the pitch ratio value set by the user through the corresponding knob.
- ~***stereoPanControlBuses*** is related to the stereo pan balance.
- ~***modeSelectionBuses*** communicates the selected feedback mode to the pitch-shifter, the delay line synthesizer and the mixer, thus changing the behavior of each one.

Moreover, the server allocates two mono control buses related to pitch-shifter parameters to be sent from the GUI to the pitch-shifter block:

- ~***pitchShifTimeDispersionControlBus*** regards the Time Dispersion value.
- ~***pitchShifPitchDispersionControlBus*** communicates the chosen Pitch Dispersion value to the pitch-shifter.

Unlike the previous set of buses which are arrays of control buses, these are single control buses. As a matter of fact, the parameters they are related to are shared between all three pitch-shifters instantiated by the server (see *Figure 2 - Diagram of the overall architecture*).
As the implemented harmonizer supports three pitched voices, arrays of buses make it easier to automatically generate GUI sections corresponding to each voice simply by using a loop structure within the code. Similarly, the server creates three different pitch-shifters and delay synthesizers, one for each voice. This provides the software with scalability features. In fact, although multiple voices are supported, a single synthDef for the pitch-shifter and the delay block, respectively, is implemented. Thus, the programmer can change the number of voices through a specific global variable within the code without changing the software design.

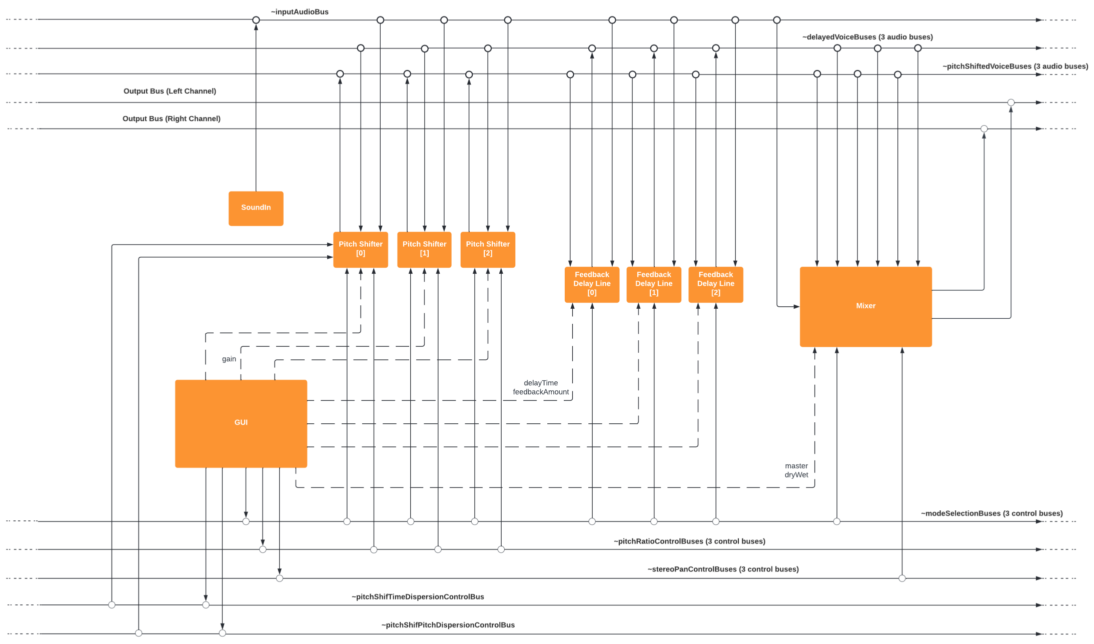

~inputAudioBus
~delayedVoiceBuses (3 audio buses)
~pitchShiftedVoiceBuses (3 audio buses)
Output Bus (Left Channel)
Output Bus (Right Channel)
SoundIn
Pitch Shifter [0]
Pitch Shifter [1]
Pitch Shifter [2]
Feedback Delay Line [0]
Feedback Delay Line [1]
Feedback Delay Line [2]
Mixer
gain
GUI
delayTime
feedbackAmount
master
dryWet
~modeSelectionBuses (3 control buses)
~pitchRatioControlBuses (3 control buses)
~stereoPanControlBuses (3 control buses)
~pitchShifTimeDispersionControlBus
~pitchShifPitchDispersionControlBus

Figure 2: Diagram of the overall architecture

2.2 Audio Processing Block

The audio processing chain consists of four SynthDefs: **"soundIn"**, **"pitchShifter"**, **"feedbackDelayLine"** and **"mixer"**, whose complete UGenGraphs are available on our Github repository

The only goal of **"soundIn"** is to capture the input from an external device, which could be an instrument such as a guitar or a microphone, maybe routed through an external sound card. The output is then written on ~***inputAudioBus***.

The **"pitchShifter"** SynthDef takes the input from ~***inputAudioBus*** and ~***delayedVoiceBuses*** and shifts it of a specified amount of semitones controlled by the GUI with the pitchRatio control, routed by ~***pitchRatioControlBuses***, which is an array of 3 buses. The delay mode for each voice is selected thanks to the ~***modeSelectionBuses***. The goal of this block is to implement the pitch-shift, using SuperCollider's function PitchShift, which parameters were carefully chosen to obtain a sound that would satisfy our tastes. Experimenting both with a microphone and a guitar routed through an audio interface, we found out that the optimal window size for a sound as natural as possible, was between 0.06 ms and 0.09 ms. It's notable that we also selected a time dispersion equal to the window size, in order to keep the characteristic effect of granular pitch-shifters, such as the pitch PitchShift provided in SuperCollider. The output of ***"pitchShifter"*** is then sent to ~***pitchShiftedVoiceBuses***, an array composed of three buses.

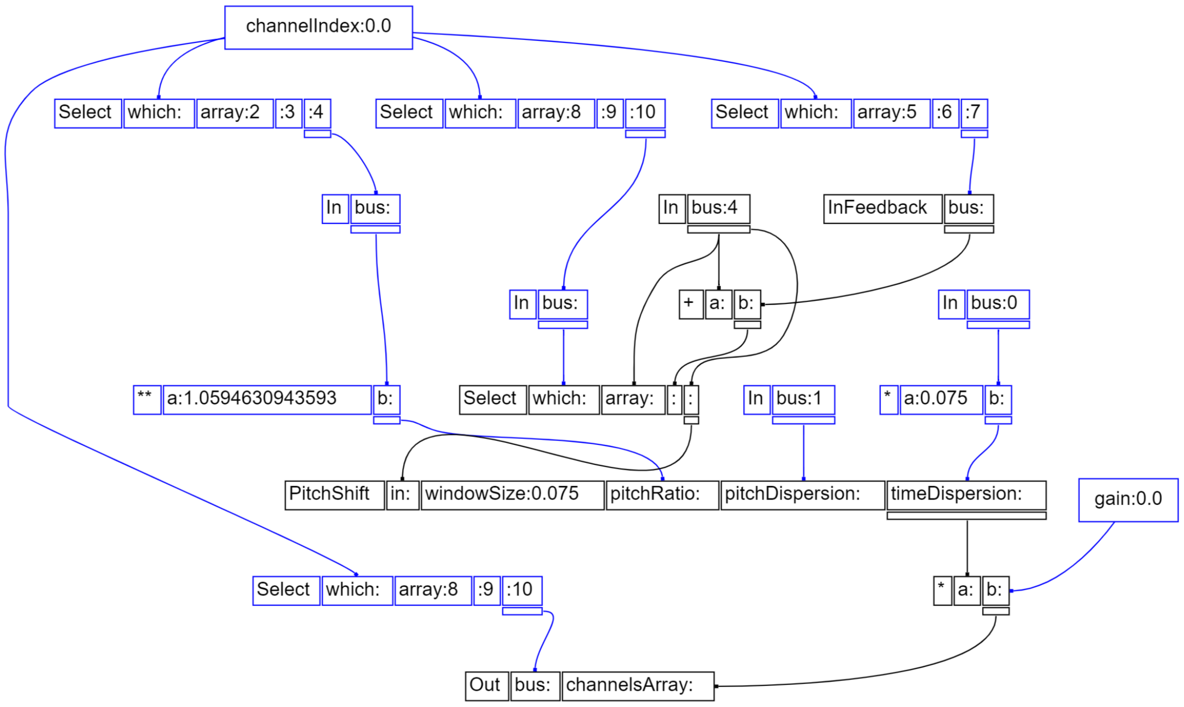

channelIndex:0.0
Select which: array:2 :3 :4
Select which: array:8 :9 :10
Select which: array:5 :6 :7
In bus:
In bus:4
InFeedback bus:
In bus:
+ a: b:
In bus:0
** a:1.0594630943593 b:
Select which: array: : :
In bus:1
* a:0.075 b:
PitchShift in: windowSize:0.075 pitchRatio: pitchDispersion: timeDispersion:
gain:0.0
Select which: array:8 :9 :10
* a: b:
Out bus: channelsArray:

Figure 3: UGens diagram of the Pitch Shifter.

The **"feedbackDelayLine"** SynthDef takes the input from the buses ~***inputAudioBus*** and the ~***pitchShiftedVoiceBuses***. Its goal is to apply one of the three implemented delay modes to the signal. We created a standard delay, that we identify with the name Normal, a delay that pitch-shifts the signal each iteration, called **Pitch Feedback** delay and lastly the **Cross Feedback delay**, which draws the signal from the pitch-shifted voices of other channels. The mode is selected through the GUI using the ~***modeSelectionBuses***.

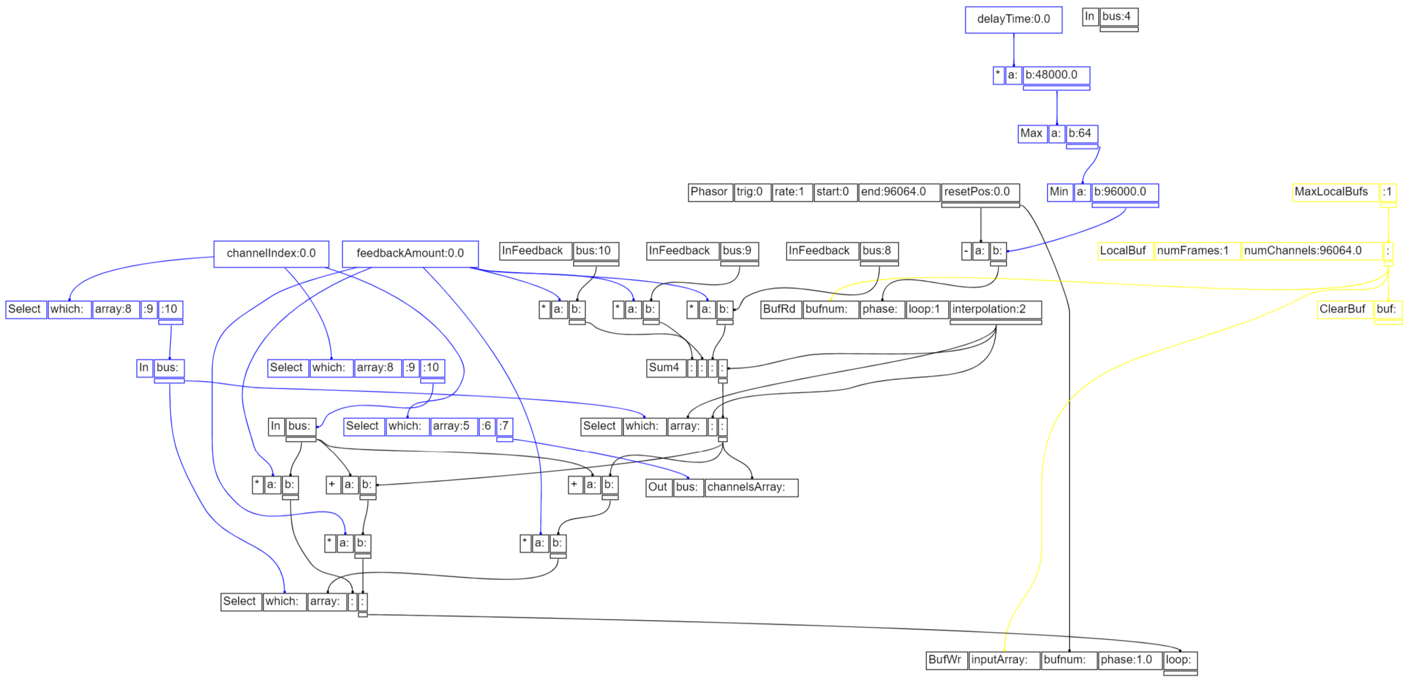


Figure 4: UGens diagram of the Feedback Delay Line.

2.3 Normal Delay

The first mode we implemented acts as the most classic of delays: any input signal is fed to the **"FeedbackDelayLine"**, delayed and attenuated or boosted by a controllable factor and then fed to **Feedback Delay** Line again, recursively. The result is finally reproduced, along with the original signal, resembling an echo-like effect.
The actual logical flow is displayed below in the 2.1 diagram. Once a signal is loaded on the input bus, the **"pitchShifter"** performs the pitch change and writes on the corresponding audio bus. Note that it only writes on a single channel of ***∼pitchShiftedVoiceBuses***, choosing the one corresponding to the correct voice channel. **Feedback Delay Line** then reads from said bus, also applying a gain controlled by the *feedbackAmount* variable. Inside this block is where the feedback happens: using Supercollider's *FBnode* function, we created a feedback node, which actually delays the signal by the controllable variable **delayTime**.
The recursion is finally obtained, writing on the correct element of ***∼delayedVoiceBuses*** and feeding it in the **Feedback Delay Line** again. Lastly, In order to reproduce the effect, the **Mixer** reads from ***∼pitchShiftedVoiceBuses*** and ***∼delayedVoiceBuses***, mixing them together with the original input.

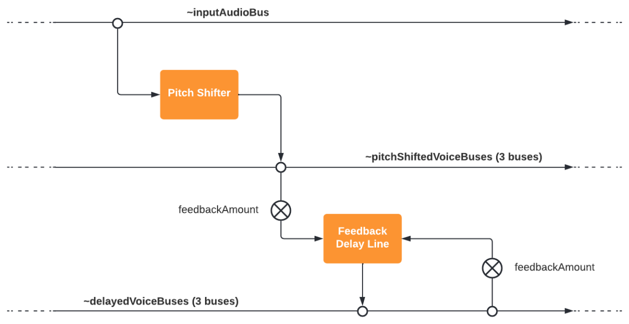


Figure 5: Diagram of the Normal delay.

2.4 Pitch Shifted Delay

The second mode is basically the classic echo, but with a twist: each recursive iteration does not simply delay the signal, but also applies the pitch-shift again. The result is an echo, which pitch increases (or decreases) each iteration by the amount shown in the Pitch Ratio parameter of the same voice.
As displayed in the *2.2 diagram*, the procedure is quite similar to the **Normal delay** one, the only difference being the **Pitch Shifter** reading from ***∼delayedVoiceBuses*** instead of the **Feedback Delay Line**. This means that the recursion loop this time also includes the **Pitch Shifter**, resulting in the already mentioned iterative pitch-shifts. In this case the mixer, which is not shown in the diagram for tidiness, only reads from ***∼pitchShiftedVoiceBuses*** and puts it together with the dry signal to obtain the final result.

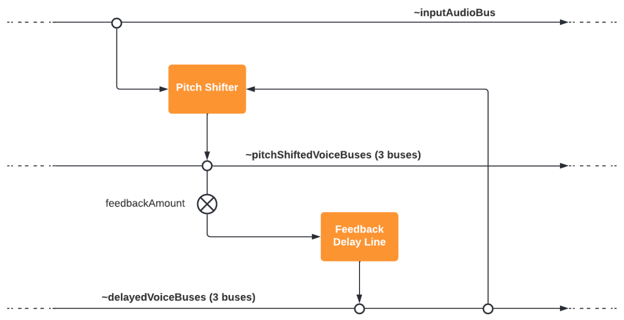

~inputAudioBus
Pitch Shifter
~pitchShiftedVoiceBuses (3 buses)
feedbackAmount
Feedback Delay Line
~delayedVoiceBuses (3 buses)

Figure 6: Diagram of the Pitch Shifted Delay.

2.5 Cross Feedback Delay

The third mode is the most experimental and complex out of the three. It was designed to generate an effect that would bounce between the two output channels, exploiting Pan usage, but adding the interchange of signals between the channels made the creation of even more whimsical sounds possible.
Although the *2.3 diagram* is a bit more complex than the others, the basic structure still resembles the Normal Delay. We can see that the recursion happens in the same way and that the feedback amount factor is still used in the same way. The main difference is that in this case the **Pitch Shifters** read from two buses instead of one, picking them to have a different index than its one. Finally the **Mixer** reads from both ***∼pitchShiftedVoiceBuses*** and ***∼delayedVoiceBuses***, adds them back together, obtaining the final result.

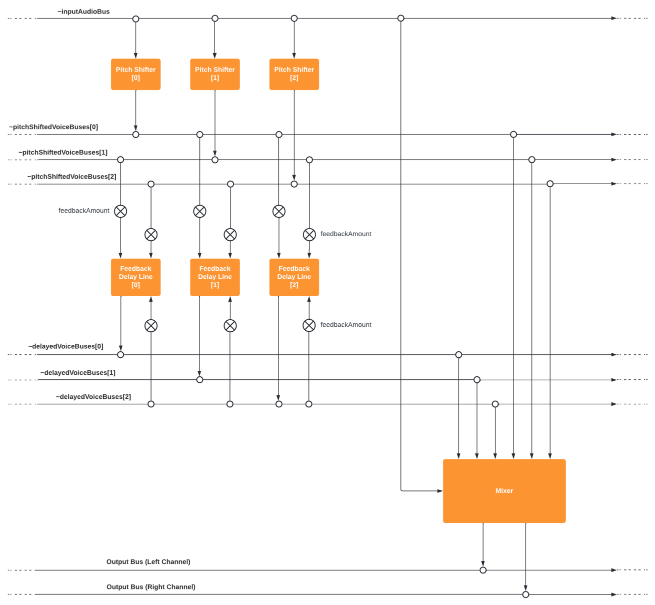
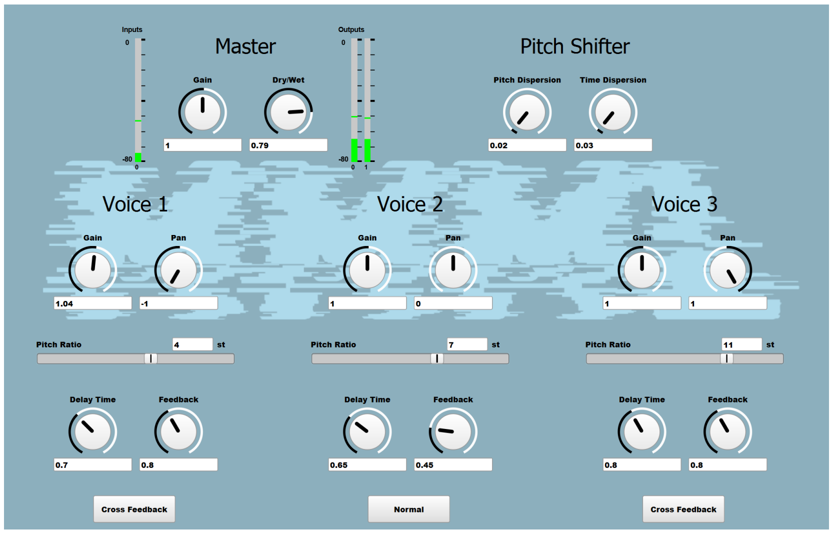


Figure 7: Diagram of the Cross Feedback Delay.

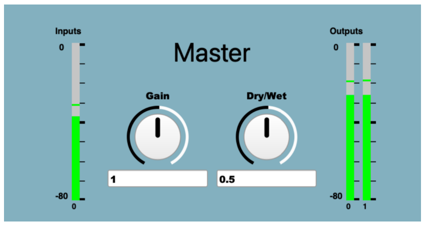
2.6 Graphical User Interface

The graphical user interface (GUI) of the software provides the musician with three main sections: **Master**, **Pitch Shifter** and **Voices section**.



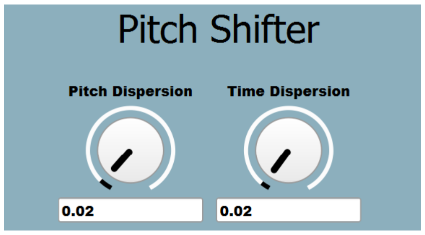
Master: Gain 1, Dry/Wet 0.79
Pitch Shifter: Pitch Dispersion 0.02, Time Dispersion 0.03
Voice 1: Gain 1.04, Pan -1, Pitch Ratio 4 st, Delay Time 0.7, Feedback 0.8, Cross Feedback
Voice 2: Gain 1, Pan 0, Pitch Ratio 7 st, Delay Time 0.65, Feedback 0.45, Normal
Voice 3: Gain 1, Pan 1, Pitch Ratio 11 st, Delay Time 0.8, Feedback 0.8, Cross Feedback

Figure 8: Graphical User Interface of the software



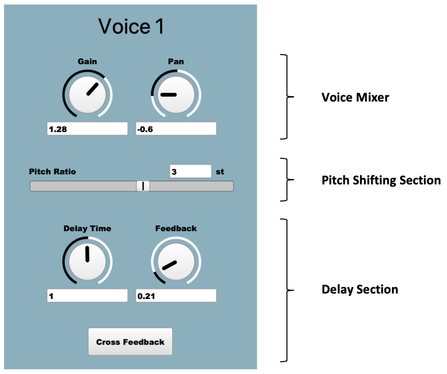
Gain 1, Dry/Wet 0.5

Figure 9: GUI - Master section.



Pitch Dispersion 0.02, Time Dispersion 0.02

Figure 10: GUI - Pitch Shifter section.



Gain 1.28, Pan -0.6, Pitch Ratio 3 st, Delay Time 1, Feedback 0.21, Cross Feedback

Figure 11: GUI - Single Voice

At the top left, the **master** section offers two knobs *(Figure 7)* through which the musician can control the gain of the final output and dry/wet balance between the clean input signal and the pitched voices. In addition to this, two meters related to input and output signals respectively are rendered so that the user is aware of potential distortion.

At the top right, the **Pitch Shifter** *(Figure 8)* control section allows the user to set Pitch Dispersion and Time Dispersion values. As said before, these parameters are shared between all three pitch-shifter instances which process the different voices individually.

Then, the voices section is divided into three panels, one for each voice generated by the harmonizer. Every panel is in turn composed of three subsections:

- **Voice Mixer** sets the gain and stereo pan of the corresponding voice.
- **Pitch Shifting** section shows a pitch ratio slider which makes it possible to set the target pitch of the voice by selecting the pitch ratio value given in the number of semitones above or below the original pitch.
- **Delay Section** provides control over some parameters of delay effect to be applied on the single voice. In fact, one can set the delay time in milliseconds and the feedback amount. Moreover, a button at the bottom center allows the artist to choose between three different feedback setups (implementation described within the previous section of the report): Normal, Pitch Feedback and Cross Feedback.

3 Conclusions

3.1 Results

Summing up our work, we produced a working harmonizer that allows manipulating all three voices' gain, pitch-shift and pan, as well as global volume and wet controls. On top of that, we added three different delays that allow both more traditional effects and more experimental ones.
The first challenge we faced was setting up the pitch-shifting block, which is the core of our harmonizer. Moreover, we really found the part about delay modes interesting and, in particular, we appreciated the part of designing increasingly complex models that we would use to obtain them.
Besides those aspects, what we are most satisfied with is for sure the architecture we achieved. Thanks to a modular structure and extensive usage of buses we were able to generate all harmonizer's voices iteratively, achieving a scalable and sustainable code, making any future changes easier. For example, even major changes such as increasing the number of available voices would be possible just by adding iterations to the loop, adjusting bus dimensions and elements position in the graphic user interface.

3.2 Future Improvements

Even if we are quite satisfied with our final result, due to limited time, we couldn't implement all the ideas that we had. Between the ideas in line we thought of creating an harmonic mode selector, that would allow a scale selection and we indeed started creating a pitch follower, that didn't find usage in our final version. Beside that, we had some ideas for more complex graphics and to apply a reverb to to the final mixed signal, but had them momentarily set aside.