Person codes: 10783942, 10561598, 10500756, 10583734, 10472602
https://github.com/magiwanders/CMLS_HW3

May 2021

1 Introduction

A harmonizer is a musical instrument which works by combining an input signal with multiple pitch-shifted versions of itself, in order to build interesting harmonies on top of melodic lines. Our goal is to create an easy to use tool which can enable both beginners and voice artists to unleash their creativity with multiple voices both in live and studio environments. We thus created Choirify, a flexible multi-mode melody line voicer-harmonizer which can be used for a variety of applications: as an experimentation tool, artistic effect, choral writing aid, or anything we have not thought about yet.
Our implementation uses Processing for the GUI, and Supercollider as the backend. The GUI and the backend communicate with each other through OSC messages. The following sections covers in detail all the implementation details of the modules and other topics like message routing, and functionality from a user-centric standpoint.

2 GUI and functionality

Choirify presents itself with an extremely straightforward and intuitive GUI. The two main modes of operations are selectable from the very top and the rest of the GUI adapts to the currently selected mode. What follows is a qualitative explanation of each mode of operation:

- GUI mode - The GUI presents itself with 12 columns, each one referred to a note of the chromatic scale and containing three drop-down selectors, as shown in Figure 1. The notes are arranged in natural order from the left to the right. In GUI mode, the microphone is the only input: when a given note is sung, the notes selected on the drop-downs of the corresponding column are superimposed upon the original signal. The final effect is a chord played for each given note. In order to make every chord fit into the scale being used, it is possible to manually change the single notes of each chord through the aforementioned drop-downs.
- MIDI mode - This mode provides a MIDI input along with the the microphone audio input. In this case there is complete freedom with the harmonizing process, as the notes being added to the original voice signal are those being played by the MIDI keyboard. As it can be seen from Figure 2, this mode also features a miniature keyboard, which display the notes that are pressed by the user. When a note is pressed, a falling bubble note visualization is triggered.

Before reaching the output, the harmonized voice signal passes through a series of effects and a variable gain. As can be seen in Figure 1, the parameters are easily tunable on the right and bottom part of the GUI by simple sliders.
The GUI is implemented using Processing with the following libraries: `ControlP5` for the GUI implementation, `oscP5` for sending and receiving OSC messages, and `The MidiBus` for sending and receiving MIDI messages. What follows is an overview of the main modules that make up the entire GUI and its logic:

- The `GUI.pde` module is the main file of the GUI, it sets the window dimensions and the aliasing properties and launches the various components.
- The `CP5.pde` module contains all components from the CP5 library, which are all the interactable parts of the GUI: buttons, sliders and dropdowns.
- The `Piano.pde` module contains the code for the piano component that appears in MIDI mode, by manually drawing white and black keys.
- The `DropdownNote.pde` module contains the definition of the dropdown which selects the notes in the columns of the MIDI mode with some options of customization.
- The `BubbleNotes.pde` module contains the code for the visualization of falling bubbles notes for MIDI mode. It is a simple physics simulation which uses Daniel Shiffman's classes `Liquid()` and `Mover()`, which help simulating massive bodies falling in a viscous fluid.

As mentioned before, the Processing code also handles the MIDI input in some modules that are not directly related to the actual graphical part:

- The `controller.pde` module contains the code to receive the MIDI input and forward it to the Supercollider backend. It also converts the dropdown GUI inputs into MIDI before sending them to the backend.
- The `sendOSC.pde` module implements wrappers for the `oscP5.send()` function in order to easily send single messages or arrays to the backend. The wrapper functions are heavily used in The `controller.pde`.

3 Supercollider Backend

The main functionalities of Choirify are implemented within its Supercollider backend. At its core there are three main synths and six effects. The synths and the effects are both interfaced through OSC messages with a controller.
The main workflow of the backed is described as follows: first of all, the pitch of the voice is estimated. Then, the pitch-shifted replicas of the voice are created and combined together with the original voice. Finally, a cascade of effects is applied to the resulting audio signal. What follows is a more detailed description of the three main synths and of the six effects.
The three main synths are:

- The `voiceTracker` synth, which tracks the note sung by the user and sends it to the GUI via OSC message. It uses the Tartini pitch estimator to achieve highly accurate pitch tracking [1]
- The `harmonizerGUI` synth, which is used when GUI mode is activated. It creates the pitch-shifted voice replicas and stores them into buffers. The target fundamental frequencies of the replicas are set inside the GUI and received as OSC messages
- The `harmonizerMIDI` synth, which is used when MIDI mode is activated. Similarly to `harmonizerGUI`, it creates the pitch-shifted replicas of the user voice. In this case, however, the frequencies of the replicas are received as MIDI messages coming from the MIDI device of the user

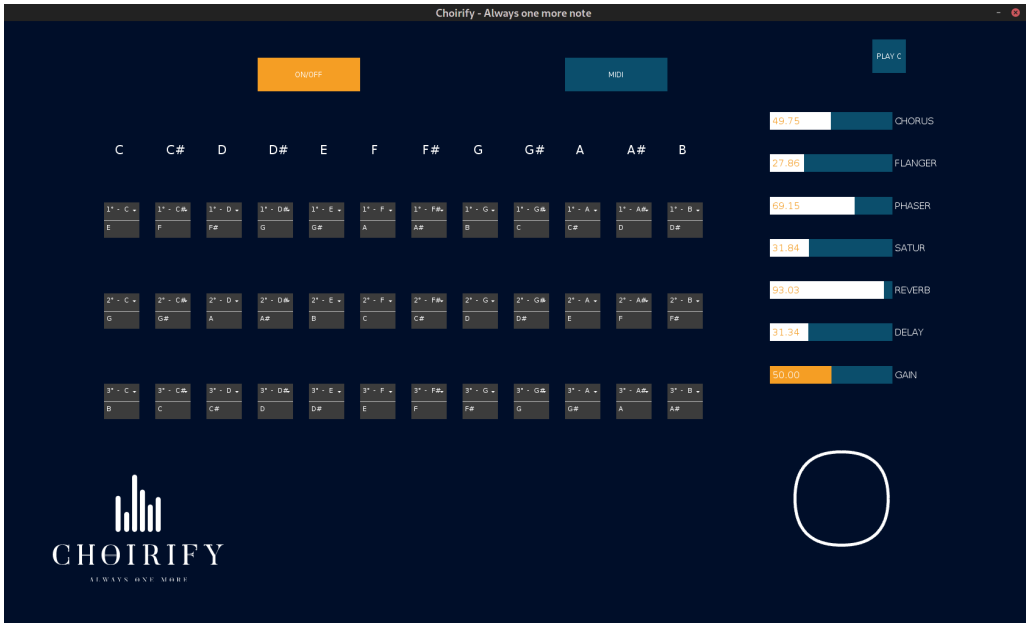

Figure 1: The graphical interface in GUI mode

The following effects are implemented:

1. Chorus
2. Flanger
3. Phaser
4. Saturation
5. Reverb
6. Delay

The effects are applied in cascade to the signal chain in the order they are listed above. Each effect has a dedicated audio bus, making it easy to add further effects to the chain.

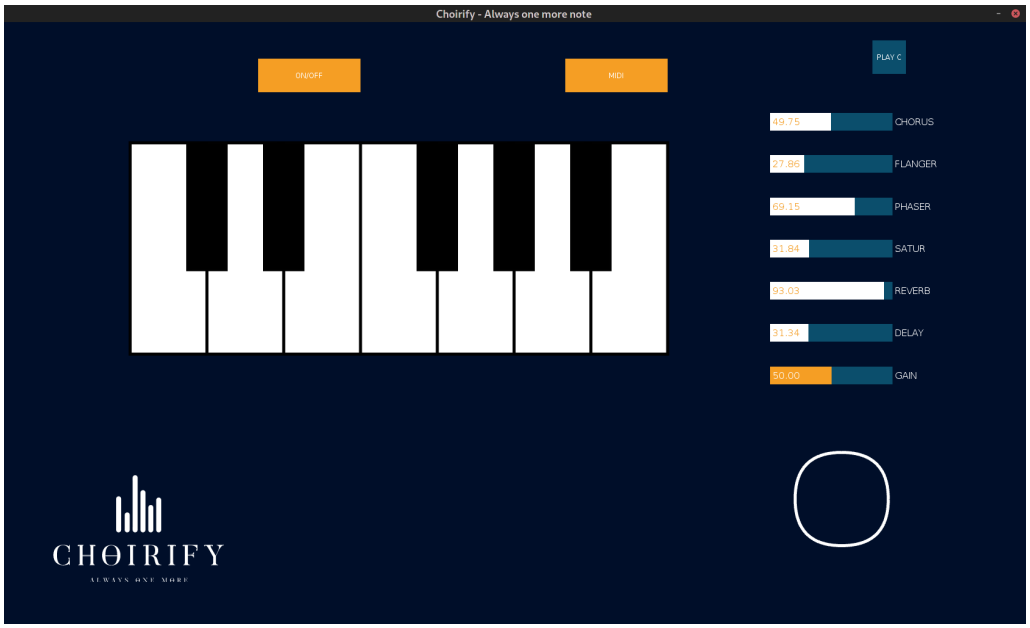

Figure 2: The graphical interface in MIDI mode

4 Audio and message routing

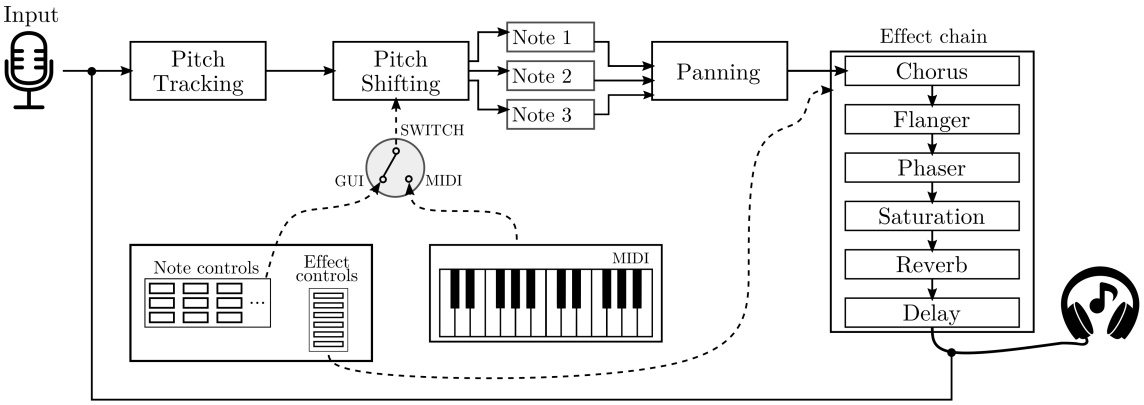

Figure 3: Schematic representation of audio, OSC and MIDI messages routing. **Solid line**: audio routes. **Dashed line**: control routes.

A schematic representation of the audio and OSC/MIDI message routing is reported in Figure 3. First of all, the audio from the microphone input is sent to the pitch tracker, which is based on the Tartini estimator. At this point, the flow differs depending on the mode of operation:

- If GUI mode is selected, the result of the tracking is used to select the correct column on the note controls of the GUI, corresponding to the sung note. From there, the chord notes are read, and the pitch-shifted copies of the original signal are generated.
- If MIDI mode is selected, the MIDI input from the keyboard is received. The notes of pitch-shifted copies of the original signal are generated based on the received MIDI notes. Please note that only the last three notes that are received via MIDI are considered.

After adequate panning between the original signal and the new voices, the signal is passed through the chain of series effects before being outputted.

5 Conclusion and possible improvements

Were this project to be so lucky to be kept in development, there are a few things that would be surely coded in. Generally, they would remove existing constraints and add useful or more advanced artistic features. Here is a list of the few we identified:

- Make the note adding process more flexible, as right now it is limited to only three notes. It could be useful to add a mute option for them or a way to add more arbitrary ones.
- About artistic visualizations, there are countless possibilities. Any good idea can be implemented in place of (or added to) the dropping bubble note animation.
- An immediate visualization of the state of the sliders could be added, for example via a polygon chart.
- Also when talking about deep learning or AI applications, there are countless possibilities. Some ideas could be adding a third input and implement existing timbric style transfer techniques before the output, or setup a third mode of operation where an AI-based algorithm chooses the notes to be added to the original voice in a reasonable way according to the current melodic line.

References

[1] Philip McLeod and Geoff Wyvill. A smarter way to find pitch. 01 2005.