propacc

Computational Music Creativity

Final project, March 2022, Christos Plachouras

Overview

The main principle of this system is to allow the user to accompany themselves with sounds derived from their past input, in order to create interesting melodic output or simply practice their improvisational singing or monophonic instrument playing. The name **propacc** is inspired by **prop**agation delay in electronic circuits and music **acc**ompaniment, which are the two conceptual inspirations for the system.

The basic process for using the system is as follows. When the system detects loud enough input by the user, it starts recording the microphone input to memory. When it detects that the phrase sang or played has ended, it stops recording. It then analyzes the recorded sound, and transforms its timbre using differentiable digital signal processing (DDSP) to that of a different instrument, while keeping the pitch, rhythm, and expressive characteristics (such as vibratos, inflections, etc.). The next time a user sings another phrase, the transformed recording starts playing at the same time, accompanying the user. In this implementation, two delayed versions of the user's input are recorded and played simultaneously when the user provides new input.

While the original motivation of the system was centered around creating interesting melodic output, it quickly became apparent how fun of an improvisation and practicing system it can be. In the case the user uses this with voice, it provides a challenging, time-constrained harmonization exercise, as the user has to quickly think about their two previous inputs and come up with a fitting third melody to sing simultaneously. In addition, since DDSP resynthesizes the new track from a continuous fundamental frequency estimation of the original, there is no quantization in the resynthesized tracks, which means the user is challenged to remain almost perfectly in-tune for each three consecutive melodic phrases.

Motivation

The initial motivation for creating the system was being able to create interesting polyphonic output with a single control: the user's singing voice. While neural network-based and also simpler approaches were considered for generating variations of the input melody, they were dismissed for three main reasons:

- I wanted the output to be predictable and solely controllable by the user. The user is not dependent on unpredictable output by a neural network and has full responsibility of the output, just like in acoustic, electric, and most digital instruments.
- Unpredictability wouldn't allow for equally unconfined improvisation. Obviously, starting a saxophone improvisation without knowing what the chords that the rest of the band is going to play next will not go well in most cases. Variations generated would have had to fit the harmonic framework that the user would be required to imply, but implicit inference of it is not accurate enough, and explicit specification of it defeats the single-control model I wanted.

- Freedom in composition is important, but constraints are necessary. I felt like this level of improvisational freedom and this level of framework-derived constraint was a nice balance for making something creative while retaining simplicity in the control of the system.

While the system was built with these principles, as mentioned in the overview it became apparent that the challenge of controlling it was bigger than anticipated. While a good singer won't have any trouble staying in-tune when there is accompaniment, it opens up the possibility for people like me to practice their singing. It can also be seen as a challenging memory and improvisation game, where the person that manages to sing the most phrases without making a "harmonic" mistake wins.

Implementation

Platforms

Pure Data was primarily used for controlling the process flow, including recording to memory, detecting singing/playing, visualizing past input, applying convolutional reverb, and interfacing with the DDSP module. The IRCAM Acids DDSP implementation used provides a DDSP module and the included preprocessing in a Pure Data wrapper. This wrapper includes Pure Data objects like sigmoid~ for pitch tracking, and interfaces with Torch to do the inference (i.e. neural audio synthesis). It is primarily built in C++, which is what allows timbre transfer inference to be done faster than real-time.

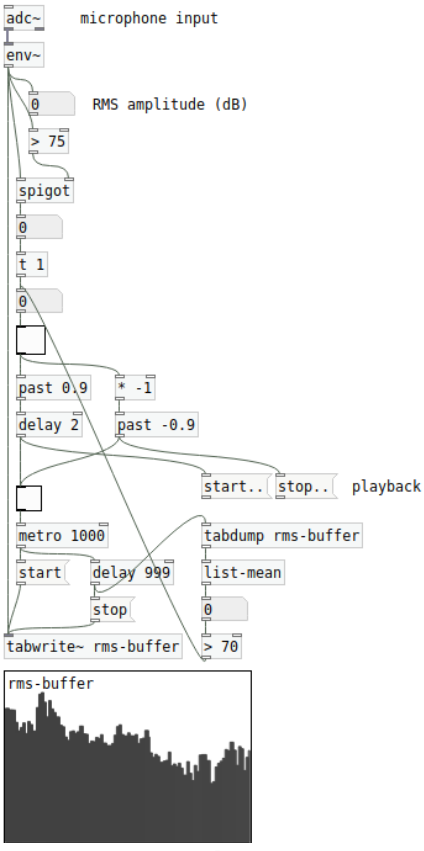

Figure 1: Recording activation

Recording

One of the challenges of this project was having a simple, predictable, but well-working approach for splitting the audio from the microphone input into phrases in real-time. The way this was done was by first calculating the RMS from the input and only declaring an active state for values over 75 using the spigot object. When the state is active, the RMS values for frames equivalent to 1 second are passed to an array and once its filled it's dumped to a list and the mean value is calculated. If this mean value is less than 70, the recording is deactivated. This ensure that momentary drops of RMS of duration less than a second don't immediately stop the recording. A more detailed implementation can be seen in figure 1.

Delayed copies

For the configuration of two delayed copies used in this project, the system works as follows. We have 3 arrays: the input array, the order1 array, and the order2 array. When the recording state is activated, the input is written in the input array. When the RMS threshold buffer deems that the recording should be stopped (i.e. phrase has ended) we stop writing to the array. If we are to keep and play delayed recordings, we have to think about write and read locking, in the sense that we can’t

be, for example, reading from an array while the recording process writes on it. To achieve this, immediately after this point, we first copy the contents of the order1 array to the order2 array. Immediately after this, we copy the contents of the input array to the order1 array. This essentially works as a stack data structure, with the elements being arrays, meaning the first to enter the stack is the last to exit. This allows us to store two delayed copies of the input without overriding anything, and doing the copying when no array modifications are expected (right after the recording stops). This is important, as next time a recording starts we want a read lock on both the order1 and order2 array in order to play back their content simultaneously. After some testing to determine what's a good enough delay for copying arrays "immediately after" the previous process, the value of 1ms was determined to be perfectly stable. A visualization of the array system can be seen in figure 2.

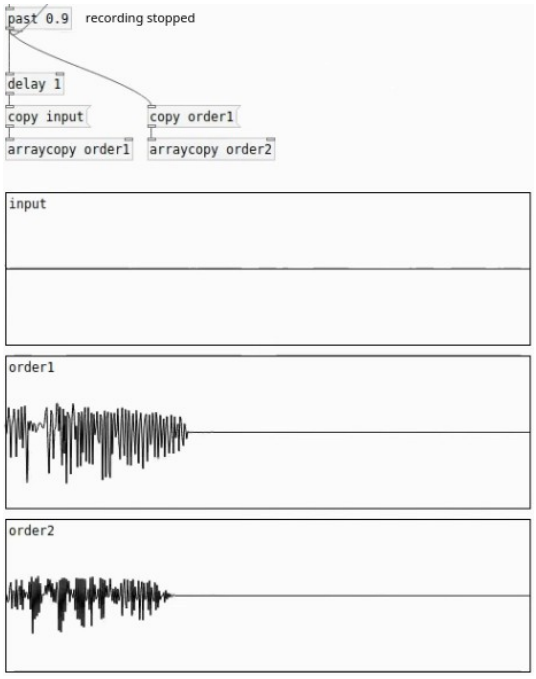

Figure 2: Delay arrays

Differentiable Signal Processing

Using the IRCAM Acids DDSP implementation in Torch in C++ which is faster than real-time allows us to not worry about whether the processing is done before attempting to playback delayed melodies. Instead, even on CPU it guarantees that we will always have resynthesized audio after passing the original array content.

I am using the two pretrained models provided, namingly the violin and saxophone ones. These can be substituted by any other model trained using the models in *github.com/acids-ircam/ddsp_pytorch*. More details on this specific adaption of DDSP can be found in this repository too, but the basic principle is that the pitch is analyzed using the sigmoid~ object, while the loudness using RMS is also registered. These two parameters are then passed to the Torch model, which infers the resynthesized audio using the instrument it was trained on, a process coined "timbre transfer". The wraping of this functionality as well as the inputs for providing the desired model path and a convolutional input response to add at the end can be seen in figure 3.

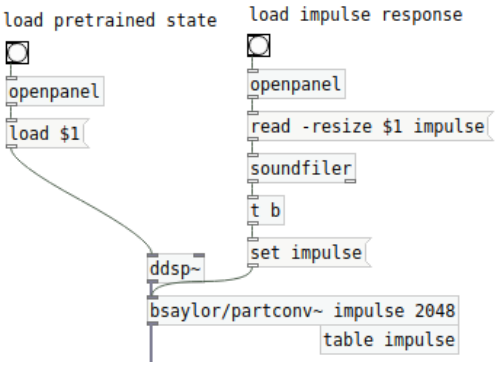

Figure 3: DDSP in Pure Data

Future versions

Ideally, future implementations of this project will be entirely web-based with Tensorflow.js and Pure Data in the web. The current dependency installation process is extremely hard and machine-dependent, with completely working installations almost only being possible on MacOS with an x86 processor. A simple interface that facilitates the game aspect of this project would also have to be developed. Still, even at its current state there's a lot of fun that can be had.