# Final Project

## IRCAM - Real Time Processing, Synthesis, Computer-Aided Composition, and Spatialization

Christos Plachouras

May 2019

My final project is a series of Max patchers and python scripts with a graphical interface for the generation of sounds that can enhance one's composition. The main idea is that the composer can pre-record a significant number of short recordings, and the algorithms will categorize them, and then, according to parameters set from the user's input, play them back with various effects. The motivation for this tool was to add interesting sounds with some, but not a lot, user control to combine them with compositions using processes like granular synthesis or vocoding to create textures or sound drones. In these cases, the sounds created by the tool can take up a role of an ornament or, with a broad definition, a melody.

The process starts with the audio input Max patcher. The user can use the simple interface to start and end a recording, as well as to save or discard it. The patcher stores .wav files with the values of an ascending arithmetic series starting from 0 as their names. This helps the recovery of the sound files by the algorithms used in python and Max later.

When the user has finished recording, they need to execute the python script 'project_general.py'. To execute the script, you will need to have the libraries librosa, pydub, numpy, and sklearn installed. The script reads all the .wav files in the same directory, and finds the one with the shortest duration. This duration value is used to create new files generated by truncating the existing .wav files to the value of the shortest duration around their center. This procedure is necessary in order for the spectral comparisons to take place (it would not be possible to compare spectrograms of different lengths using the procedure that follows). After the audio segmentation is performed and the new files are appropriately named to indicate that they are truncated, the computation of spectral features commences. In this case, I decided to extract the Mel Frequency Cepstral Coefficients without any form of further dimensionality reduction since for files of small duration these operations are performed very quickly. To categorize the audio files, I used k-means clustering, an unsupervised learning method of vector quantization. The script uses 3 clusters by default, but the user can change that number to any other number they desire by altering the number in line 77 and line 85 of the script. The script then creates .txt files with the coll format with appropriately chosen filenames. The script will create correct .txt files and names regardless of the number of clusters chosen by the user. The computation of other spectral features is next. A tempo estimation, the mean spectral flatness, the mean spectral centroid, and the centroid deviation are computed for every audio file. These parameters are later going to be expressed in the user interface as

tempo, noisiness, frequency range, and inner-file frequency range. The maximum and minimum values of these features are also calculated per category in order to be used as the range of sliders in the user interface. Lastly, .txt files are generated in the format of coll storing these values, as well as those for the maximums and minimums.

The audio parameters Max patcher follows. It is designed to read the .txt file with the maximums and minimums of each category, and map them accordingly to rslider objects in Max. The user is then provided with a simple, 4-parameter, interface of criteria. The choice of rsliders was made to allow the user to choose an interval of possible values for a parameter.

The audio selection algorithm does a lot of work on the background. After being fed the values of the criteria sliders, it retrieves audio files and compares their stored spectral features from their respective .txt file. It then determines if a file is eligible to be played for the given interval of values from the criteria sliders.

The parameters patcher provides an interface for effect selection and other playback parameters. As with the audio parameters, rsliders are used to allow interval- rather than single-value selection. The density parameter encodes the waiting time between successive playbacks. On a lower level, it controls when to determine the eligibility of a sound file too. The multiplier has two functions: its maximum value encodes how many voices -up to 5- are going to be used at the same time, while its range determines the variation between starting time of the playback between the voices. The delay and feedback

sliders provide their values to the simple delay patcher used. The speed variance describes the range of playback speed of the sound file. The option to enable or disable time stretching is also provided. Lastly, the movement slider encodes the speed of rotation for the spatialization using the around patcher from spat.

The mentioned patches are then collected and used in one patcher, which also realizes the audio playback and some more arithmetic operations needed. The general patcher by default uses 3 modules of the user interface, but the user can use as many as the cluster number they have chosen earlier.

From an artistic perspective, I am really interested in the sounds and the amount of control that the tool provides for my compositions. Unfortunately, time did not permit me to record an example of how these sounds can be combined with some recordings of texture/granular drones from modular synthesizers that I recorded in the past. Another aspect that was one of the main points of consideration for this project is modularity. Being a composition tool, its adjustability plays an important role. I decided to only use an unsupervised learning algorithm for the sound classification, but this part can be substituted for a neural network trained on the database of over 300,000 instrument sounds from the Google Magenta project, or one for voice recognition. This flexibility would also be useful in the scenario of a live performance; the artist can be recording multiple instruments in real time and then use a module per type of instrument, or they can make voice recordings of multiple people and use a module for each one of them. The

parameters used as criteria for sound file selection fit the composition style that I was envisioning; the noisiness of the sound file and the frequencies are good descriptors when you try to combine these sounds with textural drones. These criteria are, however, easily adjustable by further exploring the DSP algorithms that librosa provides, or using any other kind of feature of the sound file that you think is suitable for the composition style you are envisioning.