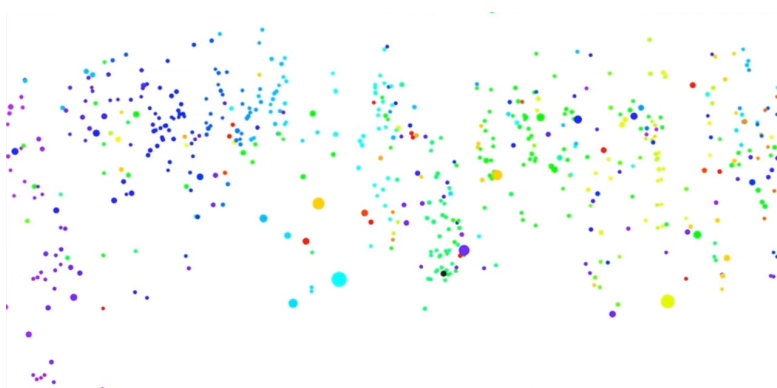




INTERNSHIP REPORT

Mapping and clustering on audio descriptors spaces for DYCI2



Supervised by Jérôme Nika and Tristan Carsault

Ilhan BEN-AMAR

January 31, 2022

Contents

1	Introduction	3
1.1	Presentation of IRCAM	3
2	Structure of the project	4
2.1	Python engine structure	4
2.2	Audio descriptors and signal processing	4
2.3	Clustering techniques	4
2.4	Mapping	7
3	Spaces and mapping	7
3.1	Definitions	7
3.2	Problematic	7
3.3	Linear mapping	7
3.4	Adaptive distribution	8
3.5	Mapping parameters	9
3.6	Space morphing for clustering	10
4	Implementation	11
4.1	Main class	11
4.2	Reversible mapper	12
5	Conclusion	13

Thanks

Many thanks to Jérôme and Tristan for taking the time to introduce me to their interesting and fascinating project. I would like to thank also Gérard Assayag that helped me to get in touch with researchers at IRCAM. Thanks to Valentin Emiya for the advices and being volunteer to review my report.

1 Introduction

This internship takes place in the context of the DYCI2 project (Creative Dynamics of Improvised Interactions [3]). This project aims notably to define the interaction between a musical human performance and a computer that improvises on it. The computer analyses the performance, takes account of the musical characteristics with the help of "audio descriptors" (indicators quantifying characteristics of the musical content). And then it decides which audio segment in its memory would be the most compatible. So the musician can explore different parts of the computer's memory by varying his music. But sometimes, the audio descriptors of the memory and those of the performance can be distant and heterogeneous, and some parts of the memory are never used whereas some others are played very often.

We define clusters on the computer's audio memory to create equivalence classes. These equivalence classes allows the audio engine to follow a predefined scenario on a given musical alphabet. So the purpose of clusterizing the audio memory is to keep a melodic and harmonic structure in the improvisation. The choice of a clustering technique and the choice of input arguments of the clustering algorithm is explained part 2 and 3. We also introduced a mapping function that returns the closest equivalence class for each audio input, and that adapts the input audio space (musician) to the computer's memory audio space. The purpose of this mapping function is to keep the improvisation similar to the musician's music and with the same variations. These functionalities are implemented in Python, and designed to be compact and reused for the rest of the project, the code structure is detailed at part 4.

The alliance of similarity, variation matching, and scenario-based musical structure, with balancing parameters; aims to define a new musical co-improvisation human/machine.

1.1 Presentation of IRCAM

IRCAM is a french public institute located in Paris, Beaubourg. It is dedicated to bring together musical creation and scientific research. The research laboratory STMS (Science and technologies for the music and sound) is hosted by IRCAM, CNRS and Sorbonne Universty. The lab is located mainly underground, where there are an anechoic chamber, a modular concert hall, and many studios. The lab has an international recognition for their cutting-edge research in that specific domain. Because the place is very limited at IRCAM and because of the Covid restrictions, this intership took place remotely. IRCAM presents many concerts and artistic creations, conferences, trainings, and events during the year. A new branch of the lab Ircam Amplify created in 2019, works on commercializing their audio softwares and expertises for companies. The DYCI2 project follows one of the research axis of the lab : creative dynamics, which aims notably to work on Artificial Intelligence for Creativity in music and augmenting the musician's performance. It is driven by the musical representations research team. DYCI2 is born by bringing together 2 projects

: Somax and ImproteK, that headed toward the same goal with 2 two different approaches.

2 Structure of the project

For the DYCI2 project, the team works on a Python library [4] for the audio generation engine and a Max library that wraps the audio engine for a live and interactive use. Because our time on this project is limited, we worked only using the python DYCI2 library.

2.1 Python engine structure

Let's summarize the complete processing of audio in the engine. To setup the engine, we need an input audio file which is going to be the computer audio memory. The engine first computes a segmentation of this audio file, and saves all of the audio descriptors for each segment. After having chosen one or several descriptors to follow, the engine is setup and ready for a co-improvisation human machine. This is where the Max library for real time is useful. When the musician plays a note, the engine extracts all of the descriptors from the sound in real time. Using the mapping function and the structures of the equivalence classes determines the right sample in the memory to go with that sound (1). The computer then plays on stage the chosen sample. In this section, we will detail all of the techniques used in that process.

2.2 Audio descriptors and signal processing

The DYCI2 library provides a lot of different audio descriptors. To begin with, each onset correspond to the attack, the start of a musical note. The onsets defines the time code where each audio segment starts. Here we use the onset detection function from librosa [5], a standard audio processing python module. Once the memory is well segmented thanks to onsets (fig. 2), the tools from librosa enable to extract the other audio descriptors from every single segment. The energy is a number obtained with the Root Means Square (RMS) of the segment, then converted on a \log_{10} scale. The pitch is also a value obtained using a peak detection method on the frequency spectrum of the audio segment. The chroma of a segment is a vector of length 12, each coordinate of the vector detects the presence of a specific note on the chromatic musical scale. For a first approach, we made the choice to restrict ourselves to the use of 1-dimensional and ordered descriptors : energy, pitch, duration (see fig. 3).

2.3 Clustering techniques

The purpose of creating clusters is linked to other works in the DYCI2 project. DYCI2 is partially based on the ImproteK project, and this article [2] summarises how they guide human/computer improvisation. One of the guidelines

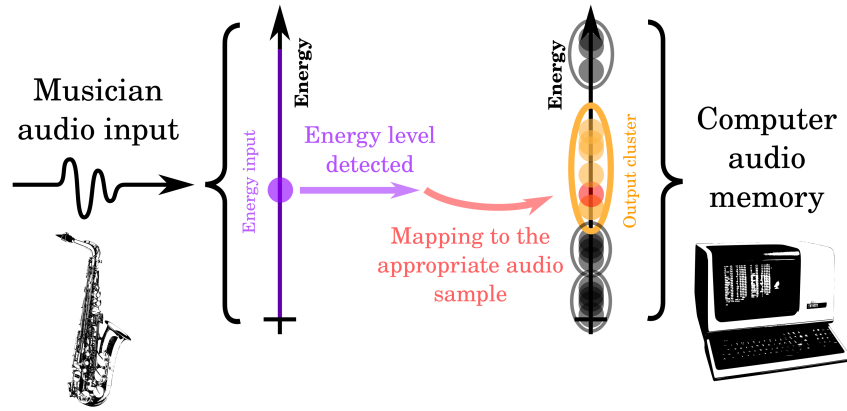


Figure 1: Illustration of the global operation of the program in the context of the project. The "energy" descriptor is chosen for this example.

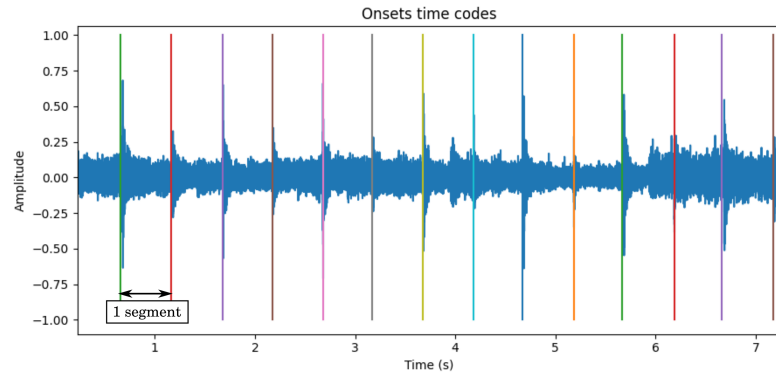


Figure 2: Onset detection function applied to an audio example ("Voyager - Daft Punk")

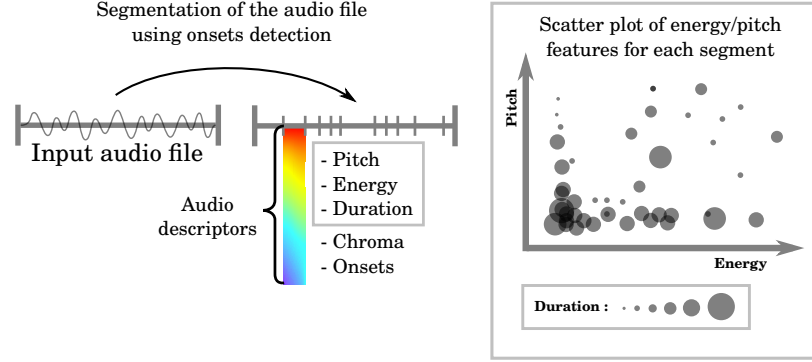


Figure 3: Audio descriptors extraction for DYCI2

of ImproteK was to predict a sequence following long term scenarios using a given musical alphabet. Here, the goal is to cluster our output space in equivalence classes, so in each equivalence class the engine will have the freedom to choose the right sample that will follow the scenario. This a hybrid decision process, that allow to get similar audio content by choosing the most similar equivalence class, but also let the freedom to follow a pre-written musical scenario. Let be k the number of clusters desired. If k is big, clusters will be smaller and more numerous, so samples in each clusters will be more similar with each other but the engine will have less choice to follow its scenario. And if k is small, there will be bigger and less clusters. In each cluster the samples will be less similar, but the ImproteK process will be able to better follow the musical structure. We are going to choose a partitional clustering technique because the clusters will be most of the time spherical or at least convex, and must be non-overlapping. Moreover it is important to have as much as possible well-balanced clusters in size and population. We could either use the K-Means algorithm or the Gaussian mixture model. K-Means uses k centroids with random positions at start, and they will converge to a certain position trying to minimize the Sum of the Squared Error (SSE). Each point belong to the closest centroid's cluster, and the SSE is computer by adding up all of the squared distances between a point and its closest centroid. We will use the clustering algorithms from the Python scikit-learn library [6]. Gaussian mixture model converges with the minimization of the SSE by adjusting the parameters of k Gaussian distributions. It allows to have more "oval-shaped" clusters for 2 and more dimensions.

Créer des classes d'équivalences description de la mémoire sur un alphabet
 -> créer un modèle de séquence sur les classes, repérer des similarités internes sur les sous-séquences

2.4 Mapping

How to deal with functions between discrete and continuous spaces.

3 Spaces and mapping

3.1 Definitions

Let E be the input space, F the output space, $d : E \times F \mapsto \mathbb{R}$ a distance between descriptors from E and F , E and F are built using the chosen descriptor. For each input from the descriptor in E , the mapping function will determine a descriptor output in F corresponding to a sample in the memory. F is a discrete space with n values corresponding to the n samples in the memory whereas E is considered as a continuous space because the mapping function must support all possible input descriptor values from the musician playing.

3.2 Problematic

The musician can't explore all of the input space because they are limited by their instrument range. Because the pitch range, the energy range is different for each instrument, sometimes the instrument descriptor range and the memory descriptor range are heterogeneous. That's why we introduce a specific mapping function that will adapt the input space to the output space and resolve these problems. The purpose of the mapping function is determined by a creative indication. A simple and intuitive example would be : "we want an output similar to the input". It results in having a simple mapping function that takes the closest sample in the memory following the chosen descriptor given an input $x \in E$: the mapping function.

$$\begin{cases} c : E \mapsto F \\ c : x \mapsto \underset{y \in F}{\operatorname{argmin}}(d(x, y)) \end{cases}$$

Because F is a discrete space, c is a stepped function and we must define it well. It's still a simple mapping function, but it has several problems in some cases. In that case, the "closest" mapping function causes some problems because only a small part of the memory is played. For instance figure 4 a), the musician's energy range is high, but almost all of the samples in the memory are lower energy. It results that only a few samples in the memory are played, or even only one sample is always played. The same problem is true with distribution disparity 4 b).

3.3 Linear mapping

The following creative indication can partially solve the range problem : "we want an output that follows the variations of the input". One of the best mapping functions for that would be to make a linear transformation on the input

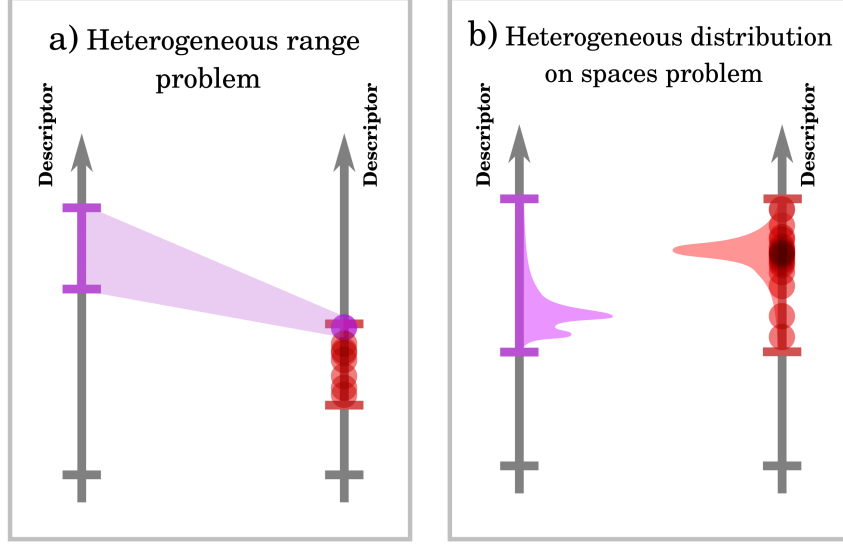


Figure 4: Illustration of the 2 different problems with mapping functions.
Left axis represents the input space range and distribution, and right axis represents the output space (samples in the audio memory)

space such that the minimum of the input space corresponds to the minimum of the output space, and same applies to the maximum. But to express this mapping function, it is necessary to first know the maximum and the minimum of the input space. We propose to introduce a **prior input training** before the live input performance from the musician. This prior training looks like a rehearsal for the musician, and it will have to be representative of the live performance. With this method, it allows to get $\min(E)$ and $\max(E)$ and to write the following functions :

$$g : \begin{cases} E \longrightarrow [0, 1] \\ x \longmapsto \frac{x - \min(E)}{\max(E) - \min(E)} \end{cases} \quad h : \begin{cases} F \longrightarrow [0, 1] \\ y \longmapsto \frac{y - \min(F)}{\max(F) - \min(F)} \end{cases}$$

Each function "normalizes" its corresponding space. With these 2 functions implemented, the linear mapping function using these two functions is :

$$\begin{aligned} E &\longrightarrow F \\ x &\longmapsto (h^{-1} \circ f)(x) \end{aligned}$$

3.4 Adaptive distribution

There is another problem with this mapping method, sometimes even if the range of the input corresponds to the range of the output space, the distribution of samples in these spaces is still heterogeneous and the memory is

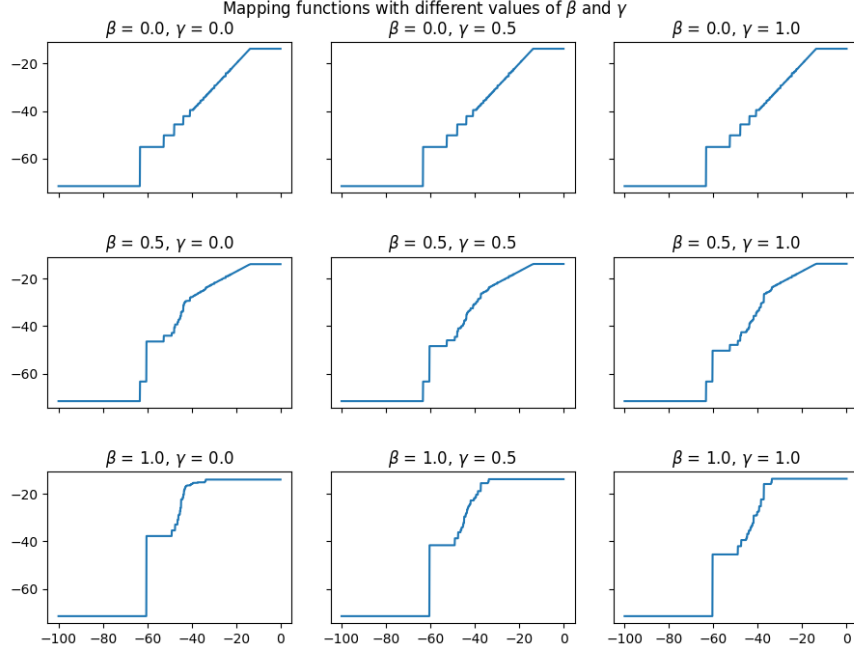


Figure 5: Plots of the ϕ mapping function.
X axis is the input space, Y axis is the output space.
Audio examples used : [7] using the energy descriptor

still not fully exploited. This is why we introduce a distribution function d , that adapts the mapping function to the sample distribution. Applied to the 2 spaces, it gives $d_E : E \rightarrow [0, 1]$ and $d_F : F \rightarrow [0, 1]$ functions. To define it, we need the index function that returns the index of the descriptor x after having sorted E .

$$d_E : \begin{cases} E \rightarrow [0, 1] \\ x \mapsto \frac{\text{index}(x)}{|E|-1} \end{cases}$$

The smaller sample will map to 0, the median sample will map to a value close to 0.5, and the bigger one will map to 1. The resulting mapping function from E to F is :

$$\begin{aligned} E &\rightarrow F \\ x &\mapsto (d_F^{-1} \circ d_E)(x) \end{aligned}$$

3.5 Mapping parameters

However, there are also drawbacks to completely adapt the input and output spaces with linear or adaptive distribution. The musical content played from the memory can be too distant from the musician's performance. Because of

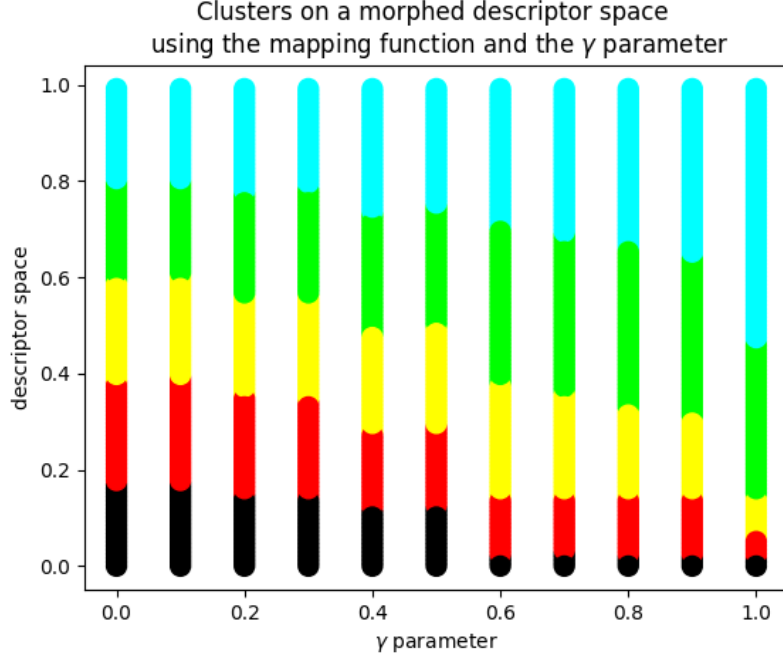


Figure 6: 5 clusters on the image by ϕ of a trombone playing [7] audio space, parameter $\beta = 1$

the creative purpose of the project, this is important to give parameters to let the freedom to adjust the mapping based on arbitrary or aesthetic choices. Here is a proposition for a complete mapping function with parameters $\beta, \gamma \in [0, 1]$:

$$\phi : \begin{cases} E \longrightarrow F \\ x \longmapsto (1 - \beta)c(x) + \beta \left[\gamma(h^{-1} \circ f)(x) + (1 - \gamma)(d_F^{-1} \circ d_E)(x) \right] \end{cases}$$

The β parameter corresponds to the matching parameter, and the γ parameter controls the distribution matching (see fig 5).

Remark : If $\beta = 0$, $\phi = c$ and the other parameter γ will have no effect.

3.6 Space morphing for clustering

To obtain a better sample repartition between clusters on the output space, we can use the mapping function ϕ . We take the image of the output space by the mapping function, and clusterize the morphed output space. Let's focus on the influence of the γ parameter that regulates the descriptor distribution. The figure 6 shows how the clusters change shape with a different mapping ϕ on the descriptor space. When $\gamma = 0$, all the clusters are approximately the same size because $\phi = d_F^{-1} \circ d_E$. The morphed space $\phi(F)$ has a constant distribution,

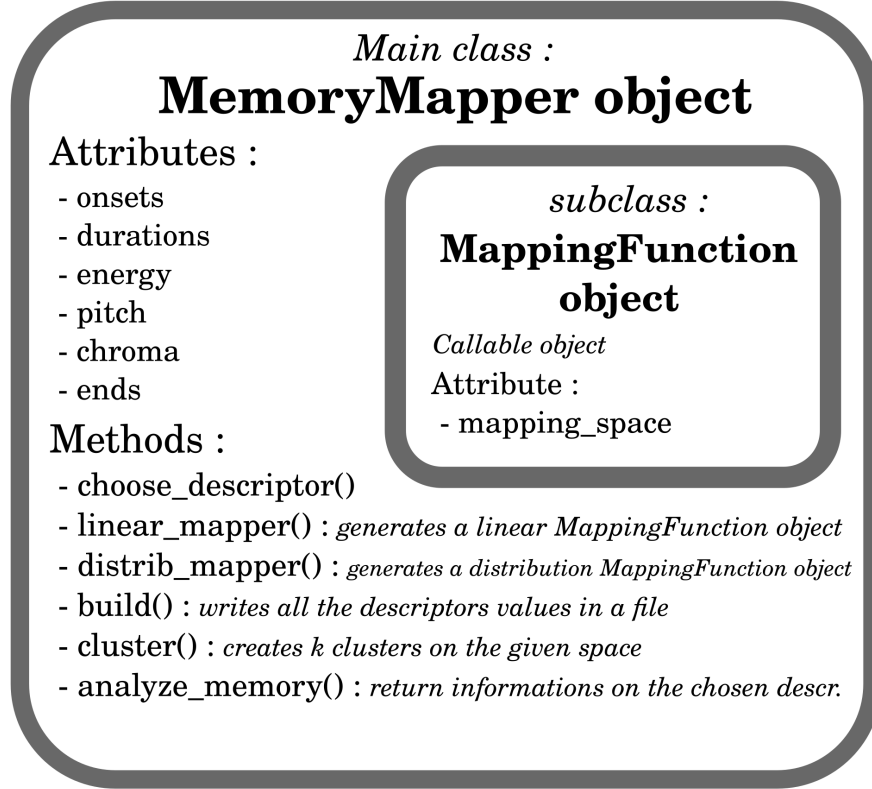


Figure 7: The structure of the python module

and thanks to partitional clustering the clusters are evenly spaced to minimize the SSE. But when $\gamma = 0$, $\phi = h^{-1} \circ f$ is an affine function. The distribution of the image space $\phi\langle F \rangle$ is the same as F , and the clusters stay the same as well. It is important to keep in mind that the image space $\phi\langle F \rangle$ is used only to compute the clusters. These computed clusters are then labeled to the original output space F . These morphings and mappings are only possible thanks to the introduction of the prior training that will define the input space E .

4 Implementation

4.1 Main class

The python module is made of only 1 class : **MemoryMapper** (fig. 7). All of the operations will be available by executing methods of a MemoryMapper object. A MemoryMapper object has to initialize with 1 argument : **infiles**. Our MemoryMapper takes one or several audio infiles, so the argument **infiles**

```

13
14 E_memory = MemoryMapper(infile_train)
15 print("Train file analysed")
16 F_memory = MemoryMapper(infile_computer)
17 print("Memory files analysed")
18 E_space = E_memory.choose_descriptor(num_desc)
19 F_space = F_memory.choose_descriptor(num_desc)
20
21 f_lin_E = E_memory.linear_mapper(num_desc)
22 f_distrib_E = E_memory.distrib_mapper(num_desc)
23
24 f_lin_F = F_memory.linear_mapper(num_desc)
25 f_distrib_F = F_memory.distrib_mapper(num_desc)
26
27
28 def c(x):
29     iy = np.argmin(np.abs(F_space - x))
30     return F_space[iy]
31
32
33 def complete_mapping(x, beta, gamma):
34     assert 0 <= alpha <= 1 and 0 <= beta <= 1
35     return (1 - beta) * c(x) + beta * \
36         (gamma * f_lin_F(f_lin_E(x), inverse=True) + (1-gamma) * f_distrib_F(f_distrib_E(x), inverse=True))

```

Figure 8: Example of use of the MemoryMapper class

can be 1 string path to an audio file or a list of string path to audio files. The initialization function computes all of the descriptors from the audio and stores them in their corresponding descriptor attribute of the MemoryMapper object : **onsets**, **durations**, **energy**, **pitch**, **chroma** and **ends**.

MappingFunction is an attribute class of **MemoryMapper**.

The **linear_mapper()** method take 1 argument, the number on the chosen descriptor for the mapping. It returns a MappingFunction object setup with the chosen descriptor exactly like the linear function described at 3.3. The **distrib_mapper()** works exactly the same, but like the distribution function described at 3.4. Also the **choose_descriptor()** method takes the number of the chosen descriptor and returns the corresponding descriptor space in the memory. The **cluster()** method returns the labels list for the chosen descriptor, but it can only be used on the attributes descriptors. **analyze_memory()** is a method that displays informations and stats about the chosen descriptor, used for debugging. An example of use of these methods can be seen fig. 8.

4.2 Reversible mapper

A **MappingFunction** object is a callable object. The purpose of this object is to create a reversible mapping function $f : A \longrightarrow B$ from 2 discrete spaces A and B . It initializes with 1 argument **mapping_space**, a tuple (A, B) that contains the 2 spaces, for instance the input descriptor space and normalized input descriptor space. When a MappingFunction object is called with an input value x , it searches the closest value to x in A :

$$a = \min_{a' \in A} |x - a'|$$

Then it takes i_a the index of a in A and return $B[i_a]$ the value of B of index i_a . With the optional argument `inverse=True`, this is exactly the same operation but by swapping A and B . It would have been interesting with more time to try interpolation between discrete values for a better accuracy in mapping. This gain in accuracy could be significant and useful if there is very few samples in the space.

5 Conclusion

The goal of this work was to produce a creative tool to enhance the co-improvisation human/machine. We developed solutions to enable the musician to fully explore and play with the memory. First, we introduced a prior rehearsal that enables the computer to analyse and better anticipate the content of the performance. It enables to build a representation of the musician's input and the computer's audio memory. Then, we expressed a mapping function that adapts to the range and distribution of the input and output spaces. Also, we propose a way to clusterize the output space, with adjustable clusters thanks to the mapping function. The other agents of the DYCI2 project will process the output cluster to follow a musical structure. Several parameters are available to adjust all of these features. k the number of clusters, β the range matching parameter and γ the distribution matching parameter.

We can't talk about the direct impact of the parameters on the performance in this report because the code hasn't been yet integrated to the project. Clustering may not be the only way to go, maybe the k -closest neighbours of the output descriptor could be considered as an equivalence class. It would be also interesting to look at ways to automatically find good parameters values during the prior training.

References

- [1] CARSAULT T., ATIAM Master thesis : Automatic Chord extraction and musical structure prediction through semi-supervised learning, Application to human-computer improvisation - IRCAM - February-August 2017, Paris
- [2] NIKA J., CHEMILLIER M., ASSAYAG G., Guider l'improvisation musicale homme-machine : une synthèse sur le système ImproteK. Journées d'Informatiques Musicales (JIM) 2016, GMEA - Labri, Mar 2016, Albi, France - hal-01380163
- [3] NIKA J., DÉGUERNEL K., CHEMLA-ROMEY-SANTOS A., VINCENT E., ASSAYAG G., International Computer Music Conference, DYCI2 agents: merging the "free", "reactive", and "scenario-based" music generation paradigms. 2017, Shangai, China - hal-01583089
- [4] The github repo of the DYCI2 Python and Max libraries : <https://github.com/DYCI2/Dyci2Lib>

- [5] McFee, B.; Lostanlen, V.; McVicar, M.; Metsai, A.; Balke, S.; Thomé, C.; Raffel, C.; Malek, A.; Lee, D.; Zalkow, F.; et al. LibROSA/LibROSA: 0.8.1rc2
- [6] PEDREGOSA, F. AND VAROQUAUX, G. AND GRAMFORT, A. AND MICHEL, V. AND THIRION, B. AND GRISEL, O. AND BLONDEL, M. AND PRETTEHOFER, P. AND WEISS, R. AND DUBOURG, V. AND VANDERPLAS, J. AND PASSOS, A. AND COURNAPEAU, D. AND BRUCHER, M. AND PERROT, M. AND DUCHESNAY, E., Journal of Machine Learning Research, Scikit-learn: Machine Learning in Python, volume 12, 2011, pages 2825-2830
- [7] Example audio extracts used for input and output spaces :
 - Trombone solo : https://www.youtube.com/watch?v=NKwGMI_6E8U
 - Igor Stravinsky - Le Sacre du Printemps (introduction) : <https://www.youtube.com/watch?v=hJH7XxTpWCI>
 - Daft Punk - Voyager : <https://www.youtube.com/watch?v=INbgG9MOWYE>