UNIVERSIDADE DE COIMBRA

MASTER THESIS

Concatenative Synthesis Applied to Rhythm

Author:
Francisco MONTEIRO

Supervisors:
Dr. Amílcar CARDOSO
Dr. Fernando PERDIGÃO
Dr. Pedro MARTINS

A thesis submitted in fulfillment of the requirements
for the degree of Mestrado Integrado em Engenheria Eletrotécnica e Computadores
in the
Departmento de Engenheria Eletrotécnica e Computadores

February 19, 2019

Declaration of Authorship

I, Francisco MONTEIRO, declare that this thesis titled, "Concatenative Synthesis Applied to Rhythm" and the work presented in it are my own. I confirm that:

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"When I'm writing, sometimes it gets to that place where I feel like the piece is writing itself and I'm trying not to get in the way."

John Zorn

UNIVERSIDADE DE COIMBRA

Abstract

Faculdade de Ciências e Tecnologia
Departmento de Engenheria Eletrotécnica e Computadores

Mestrado Integrado em Engenheria Eletrotécnica e Computadores

Concatenative Synthesis Applied to Rhythm

by Francisco MONTEIRO

Music and Technology have a long symbiotic history together and became even more related in recent times, when technology is so present in our everyday lives. Technological breakthroughs have greatly impacted the way many tasks are carried, including the music composition process. The digital era allowed the access to large databases of music and sounds on digital format and has changed the way we create music. There is an ongoing need to develop new tools and new ways to explore these new possibilities. Different forms of sound synthesis have surfaced over the years, including Granular and Concatenative Synthesis, which offered the ability to use these unlimited sound databases. Alongside with these, developments in the Music Information Retrieval field opened doors for sound interpretation, improving the indexing ability of digital files. We have seen Concatenative Synthesis being applied in the past for speech synthesis and for music composition, but few applications focused on live generation of rhythms.

We propose a Concatenative Synthesizer oriented towards rhythmic composition, which has the ability to create its own drum kit by interpreting an input drum audio signal. The system then recreates the input through different ways, including an variation algorithm based on Euclidean Rhythms. It was implemented in the programming language Max/MSP and the extension Max For Live, in order to make it usable in the DAW environment. We have also created a basic interface to interact with the user. In the end, we present a user-based evaluation and discuss the possible future developments.

Acknowledgements

To my family, the unwavering pillar of my life. To all of them, a big acknowledgement for the unconditional love. Julia for for the comprehension and eternal patience. Diogo for always being there, even when I'm the worst. Mário for the eternal support and for being my biggest music influence. Arménio, Graciema, Emília and Adriano. All my uncles, aunts and cousins. You are everything to me.

To my supervisors, Amílcar Cardoso, Pedro Martins and Fernando Perdigão for believing in my project, for their constant support, for always leaving the door open for me and bringing me down to earth when I needed.

To all my friends. Antonio for all the help in this entire project and for the endless discussions about music. Filipe for the tips, the design help and the being the greatest person on the planet. Bicá, Craveiro, Gonçalo, Martins, João, Joana, Ritas, Maria, Francisco, Bernardo, Miguel, Ivan, Gabriel, António and many many more, way too many to mention. Seriously, thank you all.

To all my friends, colleagues and mentors from Radio Universidade de Coimbra who taught and still teach me everyday so much about music. A shoutout to the Magia Negra crew.

To all those who participated in the tests.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC	**A**nalog (to) **D**igital **C**onversion
ADSR	**A**ttack **D**ecay **S**ustain **R**elease
BFCC	**B**ark (-) **F**requency **C**epstrum **C**oefficients
BPM	**B**eats **P**er **M**inute
CAAC	**C**omputer (-) **A**ided **A**lgorithmic **C**omposition
DAW	**D**igital **A**udio **W**orkstation
DSP	**D**igital **S**ignal **P**rocessing
GUI	**G**raphical **U**ser **I**nterface tools
HMM	**H**idden **M**arkov **M**odel
LVT	**L**ive **D**rum **T**ranscription
MFCC	**M**el (-) **F**requency **C**epstrum **C**oefficients
MIDI	**M**usical **I**nstrument **D**igital **I**nterface
MIREX	Music **I**nformation **R**etrieval **E**valuation e**X**change
MIR	**M**usic **I**nformation **R**etrieval

Chapter 1

Introduction

1.1 Context

Throughout times, music served many different purposes. The preliterate cultures used music in shamanistic rituals as a way to interact with animals and entertainment purposes — practices that remain untouched until these days among aboriginal tribes. These first occurrences were limited to the sound production techniques available for the prehistoric human, such as the human voice, elementary percussion, using various objects or the human body parts, and archaic instruments (Morley, 2003).

The ability to describe and register music through notation defines the distinction between Prehistoric and Ancient Music, with the oldest found composition dating back to more than 3000 years ago. Parallelly and concurrently, different civilizations defined different musical notation systems and, as they interacted with each other, these systems spread and developed.

Along with the ability to describe music, technology allowed the invention of new ways to produce sound. Different musical instrument families, that exploited the physical interactions between solid objects and air, were able to produce a controllable array of different frequencies that had underlying mathematical relations (Benson, 2006) (Xenakis, 1992).

Instruments played very important roles in society, whether it was as accompaniment of religious rituals, for aristocratic entertainment or for the diffusion of folkloric cultures. Each instrument — and the technique employed to play that instrument — is associated with a certain timbre — the perceived characteristic that is common to every note in the instrument if played in the exact same manner. These timbres alongside the notes — and of course, besides the expressiveness that one could put into playing it —- were the essential parts of a musical composition and were the central tools for musical creation.

The concept of recycling existing material for the production of new artistic pieces was already a celebrated art form in visual arts (collage). The conceptualization of a similar process in the musical domain appeared a few years before the technology that allowed its realization (Cœuroy, 1928).

Some of the first experiments with music sampling occurred inside the *musique concrète* movement that dates back to the early 1940s. The sonic explorations protagonized by Pierre Shaeffer and its pupils in *Studio D'Essai de la Radiodiffusion nationale* defined the beginning of a new era in music production (Hodgkinson, 1987).

These new techniques, besides opening up the possibility for new methods of musical creation, also augmented the concept of music beyond the long existing paradigm focused on the coordination of melody and timbre. Music could now be expressed as sound in its most abstract form, without the traditional musical qualities required, inviting the listener to discover new musical values.

In the 1980s, when microprocessors were more widely available, more powerful and more affordable, the digital sampler appeared as an instrument for music creation. Probably the most common models were the ones with keys and pads, where each key/pad were associated with a sample stored in the sampler's memory, or it could also play one sample at different speeds/pitches and could be played just like a note from a conventional instrument. This meant a lot for music culture, as it allowed, not only the ability to emulate any recorded instrument, but also the ability to work with any recording, making it central in the genesis of many musical genres and styles such as hip-hop, electronic dance music and experimental music. It also meant one did not need to have formal musical knowledge or to have the capacity to play and the access to acoustic instruments in order to create music.

Later on, the computational power of personal computers allowed them to host the most powerful samplers. The convenience of working entirely on the digital realm made software like Digital Audio Workstations and Virtual Studio Technology very appealing —- mixing engineers gave this the name *in-the-box*. The ability to easily update and to create an internal digital signal that could easily replicate all the routing configurations that would otherwise require very expensive material, time and a place to set up a studio, were important factors to the paradigm transition.

Nowadays, there is even the ability to apply DSP techniques and create effects to use on the fly while making music, like the one we'll be using in this thesis: Max for Live.

1.2 Goals

The main goal of this thesis is to create an Audio Effect that seamlessly recreates a given drum phrase automatically and intelligently, including coherent rhythmic variations according to the controls of the user. This involves a system that is capable of translating the sound input into a sequence of meaningful drum units and reuse those units as a sound source for a rhythmic recreation. It is not intended to create music on its own, but to compliment the electronic music creative process.

1.3 Motivation

Behind this thesis is a profound passion for music, music production, DSP and Programming, powered by a background in Electrical Engineering, Radio broadcasting and a long-term involvement with electronic music composition.

As an electronic music composer, one of my main interests was the creation of generative functions that coherently compliment my creative process, in order to blur the border between music production and music listening/discovering. The music creation process can become somewhat repetitive if we approach it always with the same tools and techniques, generating similar results every time. This is ever more true with electronic dance music. There is nothing wrong with that, but what if we can make it ever-changing and exciting in every iteration? That's the main motivation for this tool: make music production even more exciting and surprising.

Plus, sampling is, for us, a fascinating approach to art: recycling and, sometimes, even reviving old artistic pieces and include them in a new narratives and new contexts. we have a world of sound sources available all the time in this digital era. It seems like a waste of time and patience to scroll through databases and find good combinations of samples. Giving the user the sole responsibility to select a database,

or a source sound, from which the program can rebuild and recreate something new is also the approach we are interested in.

1.4 Structure

Here is an outline of the remainder of this document::

Chapter 2 State of the Art revision in the relevant areas for the creation of the tool we propose. We make a small review on the historical context of automatic composition and include some examples of music that is related to our objectives and proceed to classify the different kinds of algorithmic composition. We offer an overview on concatenative synthesis and Music Information Retrieval.

Chapter 3 Description of the methodology used for the production of this project and inclusion of a Gantt Diagram.

Chapter 4 Description of all aspects of our Plug-In with as much detail as possible.

Chapter 5 Description of the user-based evaluation.

Chapter 6 Conclusions, final thoughts and discussion of the possible future of the Plug-In.

Chapter 2

State of the Art

In this chapter we will review the state of the art in the following fields: Automatic Composition, Concatenative Synthesis and Music Information Retrieval. Sometimes these fields will overlap over the different sections as they have many connection links. We tried to divide them the best we could, although arguably, some examples of Concatenative Synthesis could also be considered Algorithmic Composition. We made an effort to keep a chronological sequence inside each section.

2.1 Automatic Composition

Musical composition as a process of writing an organized set of instruments and notes to achieve a desirable result, tonal harmonic or not, has an underlying direct relationship with an algorithm. Music scales, frequency relations and tempo coherence can be interpreted as the set of rules required to aid the composer complete the piece of music that produces the desired results.

Throughout the history, there has been many occurrences of the use of algorithmic concepts to aid the process of music composition even before computers. One example is the contrapuntal form, popularized and thoroughly developed by the baroque composer Johann Sebastian Bach, which is a composition technique that initiates with a melody that will later be imitated by a different voice, aiming to exactly replicate its predecessor or create some sort of variation.

Another explicit occurrence is the *Musikalisches Würfelspiel*, a series of dice games popularized in the late 18^{th} century that allowed even non-musicians to compose music. The dice rolled numbers were associated with a certain music composition fragments that were tailored to work in a sequence. To generate new sections the users just needed to carry on rolling the dice (Edwards, 2011).

We can refer to different examples where a portion of the composition was not explicitly defined and depended on randomness, such as Marcel Duchamp's *Erratum Musical* (1912), John Cage's indeterminacy music — *Music of Changes* is a series of four "books" of music that were inspired in an ancient classic text called *I Ching* — or Karlheinz Stockhausen compositions *Klavierstücke*.

Minimal music pioneers started showing their compositions around the same time — the aesthetical results of of minimal music stand far away from the general minimal movement, meaning although the name remained, it may have not been the best choice. Terry Riley composed *In C* (1968), regarded as one of the most important and influential for the genre which consisted in a composition expressed as a set of procedures. The piece is meant to be played in a group. It consists of a set of 53 short musical segments that are supposed to be played repeatedly for as long as each individual wants as long as it remains close to the group's "mass center", creating a group dynamic that dictates the flow of the performance.

Steve Reich, who was involved in *In C* premiere, recorded a series of pieces using tape machines and voice recordings. Two of his most well-known pieces, *It's Gonna Rain* (1965) and *Come Out* (1966), involved having two tape players playing the same recording in a loop with a slightly different speed, each in a different side of the stereo space, slowly generating a phasing effect. He later applied this principle to conventional/instrument based composition - *Piano Phase* (1967), *Violin Phase* (1967) and *Pendulum Music* (1968).

Alongside these, Philip Glass is also a landmark composer in the genre. His music is characterized by very densely populated and intricate note loops which discretely evolve throughout the music pieces, many times going back and forth between sections, giving a feeling of constant stationary movement. *Music with Changing Parts* (1971) is one of his first works with these techniques. What all these have in common is the underlying procedural nature of the composition process – in fact, in the beginning, it used to be called *process music*.

The advent of computers opened a new world of possibilities for music composition, including the possibility of transforming it into an entirely automated process or preserve some decision abilities to the user. In the computer era, it is possible to distinguish two different branches in this kind of composition (Fernández and Vico, 2013):

Computer-aided algorithmic composition focused on the creation of languages, frameworks and graphical tools to provide support for the composition process but still with considerable human intervention.

Algorithmic Composition techniques, languages or tools aiming to compose music in an entirely automated way, or that foresees minimal human intervention

2.1.1 Algorithmic Composition

Illiac Suite (1957) composed by Lejaren Hiller in collaboration with Leonard Issacson, is commonly regarded as the first piece of music composed by a computer and is characterized by the use of random processes and a constraint-based system, using probability tables to control the distribution of melodic intervals (Sandred, Laurson, and Kuuskankare, 2009).

Around the same time, the Greek-French composer Iannis Xenakis, motivated by the relationship between mathematics and music, pioneered and coined the genre stochastic music. Throughout his compositional work, he applied different mathematical concepts to music, like statistical structures in *Pithoprakta* (1956), Markov Chains in *Analogique A et B* (1959) (Di Scipio, 2006), Minimal Constraints in *Achorripsis* (1957) (Harley, 2004). He even wrote *Formalized Music: Thought and Mathematics in Composition* considered a groundbreaking work in the field (Iannis Xennakis, 1963) that included FORTRAN instructions for his music composition algorithm. In the late 60s he developed his own Stochastic Music Programme (SMC). His passion with the use of mathematics and computers in the studio was a theme of divergence between him and Pierre Schaeffer during his involvement with *Groupe de Recherches Musicales* projects (Gibson and Solomos, 2013).

Brian Eno, an ambient music pioneer was also experimenting with tape recorders and developed a tape delay system which he used in collaboration with Robert Fripp – *King Crimson*'s guitar player at the time - for their album *No Pussyfooting* (1973). In fact, part of his work *Music for Airports* (1978) included the same phasing technique of Reich, except this time there were multiple different recordings and the loop durations were also more disparate. He was the one who coined the term *Generative*

MUsic in 1995 while developing SSEYO's Koan Pro and created one of the first generative music software products, which he later used to create his album *Generative Music I* (1996). To this day he continues to produce music and develop interactive generative music software products — see Figure 2.1.

> *"And essentially the idea there is that one is making a kind of music in the way that one might make a garden. One is carefully constructing seeds, or finding seeds, carefully planting them and then letting them have their life. And that life is not necessarily exactly what you'd envisaged for them. It is characteristic of the kind of work that I do that I'm really not aware of how the final result is going to look or sound. So in fact, I'm deliberately constructing systems that will put me in the same position as any other member of the audience. I want to be surprised by it as well. And indeed, I often am."*
>
> Brian Eno, Composers as Gardeners (2011)

FIGURE 2.1: Brian Eno's *Scape*, an interactive music application released for *iPad*

Michelle O'Rourke (2014) puts it in words very synthetically: "Generative music can be broadly defined as a compositional practice which sets a system into motion with some degree of autonomy which in turn results in a complex musical generation". Algorithmic Composition is hard to categorize as there are many different authors that express this differently. Our taxonomy is very simple and we suggest (Nierhaus, 2009) or (Fernández and Vico, 2013) for further reading. The categories overlap frequently in many different projects, making it ever so difficult to associate one's work with a single category:

Stochastic Models Compositions that rely on the use of stochastic processes. We mentioned a few historical examples in 2.1

Grammar Models rule-based systems that respect to the definition of a formal grammar. Further reading about this, particularly about the applications of Lindenmayer Sytems in music can be done in (Mccormack, 1996).

Machine Learning Models Systems that are not explicitly programmed. Instead they create their generation processes by the analysis of musical databases, inferring the rules that make the decisions for the system. They benefit from computer science procedures based in Machine Learning. We can recommend (Fernández and Vico, 2013) and (Briot, Hadjeres, and Pachet, 2017) for further reading on the matter.

Evolutionary models Genetic Algorithms stand as an optimization algorithm that mimic evolutionary biology natural selection. A population of individual solutions is tested against a fitness function and each individual is rated accordingly. The best performing individuals will be more probable to breed the next generation, either by selection – the maintenance of a similar solution in the next generation –, crossover – the creation of a child individual that takes properties from two fit individuals – and mutation – the creation of a new child individual that takes random values. An historical example of the application of Genetic Algorithms to jazz improvisation can be found in (Papadopoulos and Wiggins, 1999).

2.1.2 Computer-Aided Algorithmic Composition

The origins of CAAC date back to 1950s, when engineer Max Mathews started seeing the possibilities of computer music while experimenting on digital transmission of telephone calls. His programme *Music I* is considered the first computer programme and he went on to developed a family of software products named *Music-N*, which set the blueprint for most modern computer audio synthesis programmes, such as CSound, SuperCollider and Max/MSP – in fact the latter is named after Max Mathews himself as a tribute.

According to Lejaren Hiller (1981) "computer-assisted composition is difficult to define, difficult to limit, and difficult to systematize". Cristopher Ariza (2005) proposes a definition for CAAC: "A CAAC system is software that facilitates the generation of new music by means other than the manipulation of a direct music representation."

2.2 Concatenative Synthesis

By analyzing the sampling ability of the first sequential samplers – late 1980s – we could infer they were the first concatenative synthesizers. They allowed the possibility to record an audio buffer and store it in the sampler's memory. The user had to manually divide the recorded audio into individual units of his choice and only then use the individual units — *samples* is the electronic music community denomination — to create a new piece.

Our interest in Concatenative Synthesis is much more related to algorithmic/itelligent systems rather than the traditional manually controlled sampler. For that reason, we take a leap in time to avoid mentioning the unnecessary.

In the musical domain, there was already a method of synthesis called Granular Synthesis that inspired Concatenative Synthesis. In granular synthesis, the units are called grains and their typical length ranges between 1 to 50 ms. The grains are unlabeled and the corpus is defined by the user by the segmentation of a single audio file. As a tool for music creation, its main objective is to create evolving and non-repeating soundscapes that comprise of the layering of multiple grains playing at different speed, duration, frequency among other controllable parameters. In its

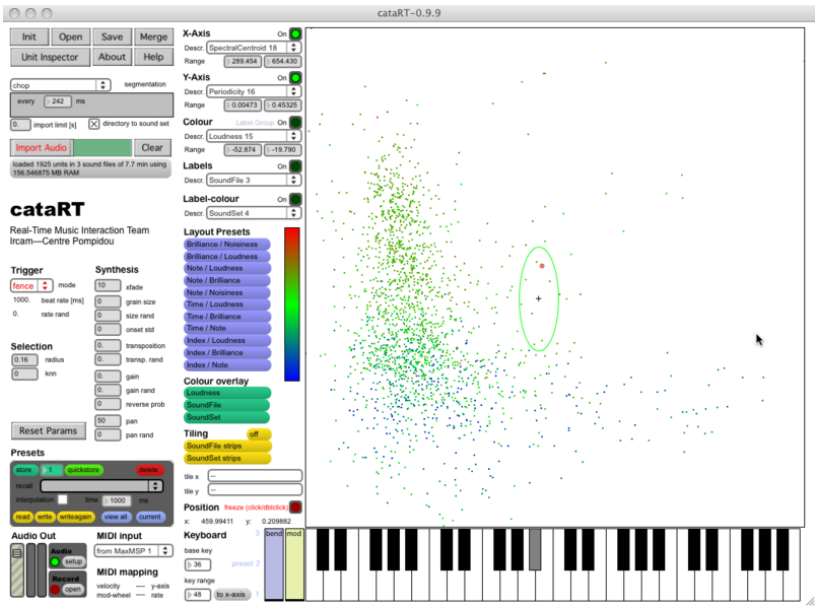


FIGURE 2.2: *CataRT* interface (Schwarz et al., 2006). Source: `http://imtr.ircam.fr/imtr/CataRT`

traditional usage, it is not suitable for rhythmic composition, as the grain size range does not comply with the typical temporal proportions of conventional music, neither does the randomness for the selection of units. Concatenative synthesis units can have a larger range of duration and can also include meta-information that describes and distinguishes units in a corpus.

One of the first application of concatenative synthesis was actually in the speech synthesis field (Hunt and Black, 1996). Using a database of labeled prerecorded speech units, and given a phonetically described target, the algorithm linked the elementary units — phonemes — according to a Hidden Markov Model. Two indicators were selected for the calculation of the transition probabilities: the distance between the object and the database unit and the concatenation cost. The best possible solution was always desirable, so the Viterbi algorithm was used — an optimization algorithm used to find the most likely sequence of Hidden States.

Inspired by the speech synthesis approach, Schwarz (2000) applied the same principle to music creation in his *Caterpillar* system. The use of the Viterbi algorithm was dropped, as it narrows down the output to the best possible solution every time, which is not beneficial in the artistic perspective. The value of the algorithm is to find coherent units and transitions but also new and ever-changing solutions, resulting in a different approach related to his speech synthesis peers, changing the unit selection paradigm from "ideal" to "acceptable", thus giving the algorithm more room for innovation.

This constraint-satisfaction mechanism or *music mosaicing* is proposed in (Zils and Pachet, 2001). We can also recommend (Anders and Miranda, 2011) for further reading on constraint programming in music. Besides the unit constraint, they defined a sequence constraint related to the sequence of units. The algorithm makes an heuristic search to find a sequence that satisfactorily minimizes the global cost.

Although Markov Models were proved able to produce new musical patterns given a corpus of patterns of a certain style, the generative power of the algorithms wasn't able to have the same effect on the long-term domain of full musical pieces.

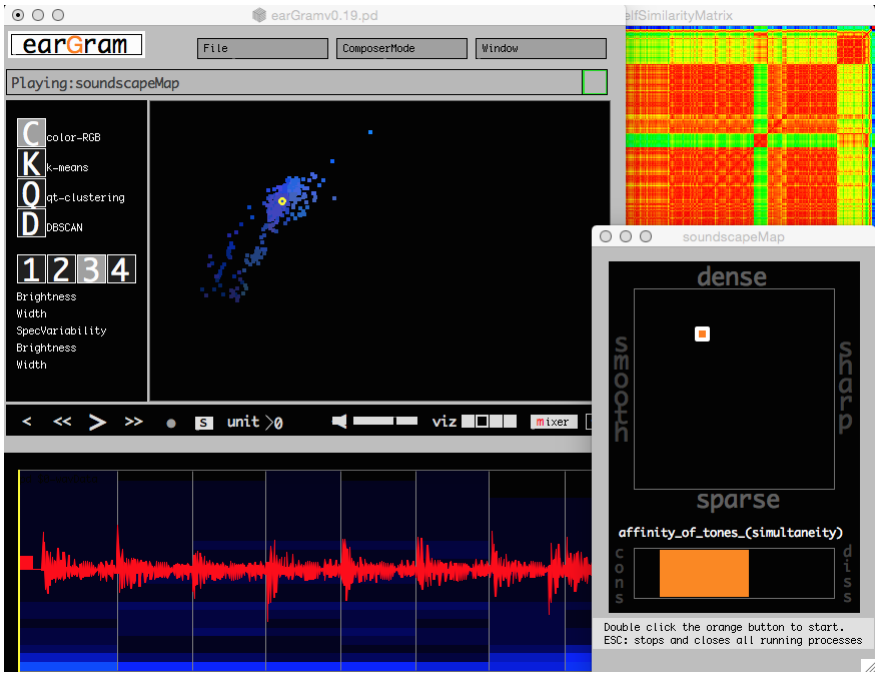


FIGURE 2.3: EarGram interface (Bernardes, Guedes, and Pennycook, 2013). Source: `http://imtr.ircam.fr/imtr/CataRT`

In (2003), François Pachet proposed the system *The Continuator*, an application that could emulate the style of a performance. He included in that software what he called *Elementary Markov Cosntraints* (EMC), a Markov Model system that also included concepts from *constraint satisfaction*.

Most these past approaches lacked real-time user interaction. In order to counter that, in 2006 Diemo Schwarz (Schwarz et al., 2006) introduced *CataRT*, which included what he named *interactive timbre space*. The corpus is mapped in a 2-dimension space according to two user selected parameters. Given a maximum distance of transition, the sound sequence possibilities are controlled by the user's mouse.

Bernardes, Guedes, and Pennycook (2013) combined the user interface of Schwarz with Tristan Jehan's *Skeleton* (Jehan, 2005) — a creative system that accounted for the perceptual listening models of the human hearing system — to create *EarGram*, a software designed for PureData which introduced new visualization features along with clustering possibilities. It included four different playing modes: *SpaceMap*, which resembles the interactive timbre space by Schwarz with the ability to control the concatenation possibilities through mouse positioning; *InfiniteLoop* endlessly recombines a track with respect to its structural properties in a non-repeating way; *StuffMeter* groups audio segments from different tracks to allow their manipulation in real-time and assist mashup creation; *soundscapeMap* allows the creation of soundscapes through the combination of environment recordings and different real-time controls.

Cárthach Ó Nuanáin (2017) introduced *RhythmCAT*, a VST Plug-In that uses a Concatenative Synthesis approach for rhythm generation. Given a *seed* rhythmic loop, the plug-in segments up the different individual instruments in the sequence through host tempo information, extracts features (loudness, spectral centroid, spectral flatness and MFCCs) from each of the *seed* units and then matches the individual units to a user selected and previously analyzed corpus of units. The seed sequence

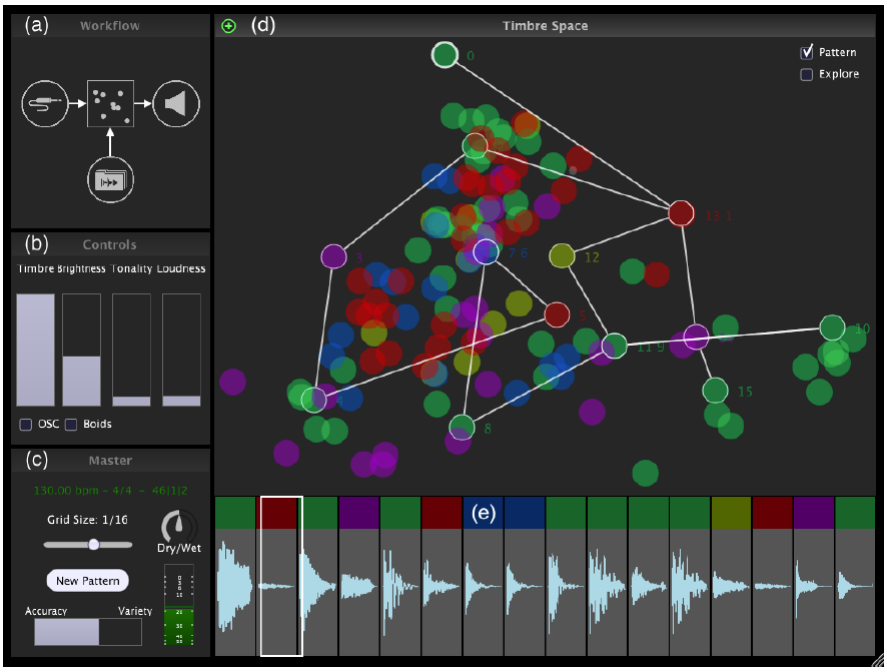


FIGURE 2.4: *RhythmCAT* interface. Source: (Ó Nuanáin, 2017)

is then recreated with the corpus units with respect to a controllable concatenation cost, calculated by the weighted Euclidean Distance of the extracted features. The interface also includes the 2-dimentional map of previous examples.

2.3 Music Information Retrieval

Music Information Retrieval is a multidisciplinary field concerned with discovering meaningful information from audio sources through the extraction of features, indexing of music and develop different search and retrieval schemes (Schedl, Gómez, and Urbano, 2014). With the development of computational power, problems that used to be computationally demanding, became viable to process even using personal computers. In (2002), Vinet, Herrera and Pachet initiated *CUIDADO*, a framework that made available a large set of feature extraction implementations allowing the utilization of these concepts in various applications.

The International Society for Music Information Retrieval Conference (ISMIR), hosts an annual research challenge called MIREX (Music Information Retrieval Evaluation eXchange). The objective is to compare and evaluate the accuracy and effieciency of different approaches to different tasks related to MIR, such as Audio Classification, Audio Tempo Estimation, Melody Extraction, Speech Detection and Audio Fingerprinting.

MIR can be, simplistically, split up in the following subfields (Schedl, Gómez, and Urbano, 2014) — see Table 2.1:

Feature Extraction The subfield concerned with the technical aspects of the creation of automatic descriptors. Some typical features are time and frequency domain representations, timbre, pitch content. As we raise the level of description ability, we can include perceptual models like MFCC and BFCCs — representations of the power spectrum of a sound filtered by psychoacoustic models that aim to mimic human hearing — or even multi-pitch estimation, onset detection and rhythmic properties.

Similarity The ability to find structure and the repetition/propagation of that structure on the same musical piece — self-similatiry — or to compute distance between two different pieces.

Classification Extracted descriptors can be combined to retrieve high-level information about the audio source.

This can seem like a field whose applications are very specific, however it is definitely not the case, as it has very tangible applications in contemporary society, and seem to only increase its relevance in the future as music is tending more and more to a fully digital paradigm. This is a small list of applications that can benefit from this field's findings, although a lot more can be found in (Schedl, Gómez, and Urbano, 2014):

Music Retrieval Intended to find music in large collections by a particular similarity criterion. Audio fingerprinting is already very present in our daily lives with applications like *Shazam*.

Music Recommendation Intended to help users find new music based on their own music preferences. E.g. *Last.fm, Pandora*.

Music Playlist Generation Related to Music Recommendation, though it also comprises the idea of automatically mix a sequence of tracks with some sensibility (Bernardes, Davies, and Guedes, 2017).

Music Transcription Translate the pitch or percussive content of an audio file into annotations meaningful to music theory.

Music Production Assist the process of music creation. Classification/labelling of sample/instrument databases; creation of new DSP based effects or the development of tools that aid the automatic generation of musically coherent content.

TABLE 2.1: Typical MIR Subfields and Tasks. Adaptated from (Schedl, Gómez, and Urbano, 2014)

Tasks	
FEATURE EXTRACTION	APPLICATIONS
Timbre description	Audio fingerprinting
Music transcription and melody extraction	Content-based querying and retrieval
Onset detection, beat tracking, and tempo estimation	Music recommendation
Tonality estimation: chroma, chord, and key	Playlist generation
Structural analysis, segmentation and summarization	Audio-to-score alignment and music synchronization
CLASSIFICATION	Song/artist popularity estimation
Emotion and mood recognition	Music visualization
Genre classification	
Instrument classification	
Composer, artist and singer identification	Browsing user interfaces
Auto-tagging	Interfaces for music interaction
	Personalized, context-aware and adaptive systems
	SIMILARITY
	Similarity measurement
	Cover song identification
	Query by humming

TABLE 2.2: Individual Task Definition for the different Drum Transcription Modalities. Source: (Wu et al., 2018)

Category	**Acronym**	**Abbreviation for**
Drum Transcription Task	DSC	Drum Sound Classification
	DSSS	Drum Sound Similarity Search
	DTC	Drum Technique Classification
	DTD	Drum Transcription of Drum-only Recordings
	DTP	Drum Transcription in the Presence of Additional Percussion
	DTM	Drum Transcription in the Presence of Melodic Instruments
	OD	Onset Detection
	VPT	Voice Percussion Transcription

2.3.1 Drum Transcription

The most relevant MIR subfield for this paper is Classification, particularly drum classification or drum transcription. Equivalently to pitch estimation, which aims to retrieve the notes present in a piece of audio, drum transcription aims to identify the drum instrument that is being played.

Although it does not receive as much attention as pitch estimation, Drum Transcription has been the subject of research since the early years of MIR and different approaches have been proposed to solve the problem. It has recently reappeared as one of MIREX challenges.

Even inside Drum Transcription, there is different problems — see Table 2.2. Our main focus will be Drum Sound Classification, which deals with instrument classification of an isolated drum sound. Ramires (2017) presents a good overview of this subject and the solutions available while also proposing a system that translates vocalized percussion.

Drum Classification can be divided in 4 different approaches:

Segment and Classify Dividing the input sound into individual segments, usually by onset detection, extracting features from each segment and draw conclusions from those features to classify, usually through machine learning algorithms such as Support Vector Machines (SVMs)

Separate and Detect The signal is separated into individual streams associated with a target class and onsets are detected in each stream. The components can be computed through Independent Component Analysis (ICA), Independent Subspace Analysis (ISA) or Prior Subspace Analysis (PSA) (Bader, 2018)

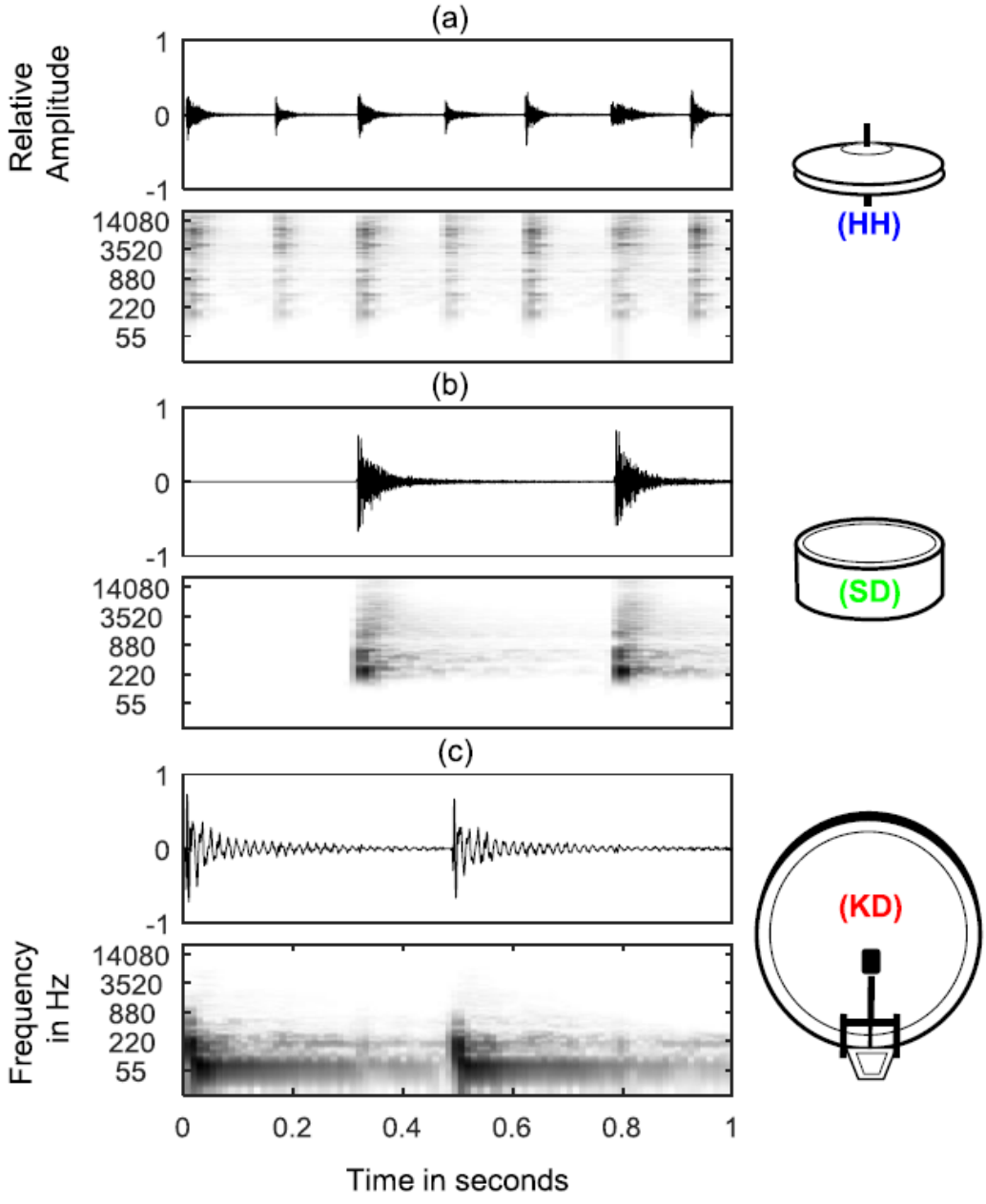
Match and Adapt Relies on temporal or spectral templates of typical drum sounds. The individual events are matched with the templates and then the templates are iteratively adapted to the given signal (Yoshii, Goto, and Okuno, 2014) (Pampalk, Herrera, and Goto, 2008)

HMM-based recognition Use a network of connected HMMs — a particular case of Markov Models where the states are hidden — in the transcription in order to locate sound onsets and recognize the content (Jouni and Anssi, 2010)

As every classification problem, one of the key definitions is the classes themselves — the different outputs possible by the algorithm. We are going to stick to the three central elements of Western music drum kits: Kick, Snare and Hi-Hat — see Figure 2.5.

Since we intend to create a music production tool with live applications, we must be aware of computational restrictions of online applications.

For event detection, an High Frequency Content onset detector is used, as according to (Brossier, 2006) it is the most accurate type of onset detector for percussion.



00

FIGURE 2.5: Illustration of typical drum sound events of (a) Hi-Hat, (b) Snare Drum, and (c) Kick Drum, in time-domain signal in black and the corresponding spectrogram representation, with darker shades of gray representing higher energy. Source: (Wu et al., 2018)

For the Drum Transcription task we will be using *Live Drum Transcription* or LDT, an open-source Max/MSP object (Miron, Davies, and Gouyon, 2013). The object works with audio signal as input — ideally a sequence of traditional drums: Kick, Snares or Hi-hats — and outputs impulses out of each of the three channels corresponding to each of the three drum instruments. In order to avoid false positives, they implemented an Instance-Filtering method in order to create another layer of sub-band complex onset detectors, each corresponding to the frequency bands associated to each drum sound. If no onset is detected in a certain sub-band, that particular instance is excluded from the classifier. Energy in 23 bark bands, 23 bark frequency cepstrum coefficients, spectral centroid and spectral rolloff were the selected descriptors to feed the *k*-Nearest Neighbour Classifier, a classification procedure that assigns the the class of a certain input according to which one has the majority among *k*-Nearest Neighbours from the input.

2.4 Conclusion

Our bibliographic revision was an important starting point to understand what we could achieve with this project. This insight was key in the concept definition of our project and what tools we would use. It became clear that Max/MSP and the Max For Live extension was the best platform to create the plug-in. It had all the tools needed: the capacity to operate live, the DSP possibilities, the compatibility with a popular DAW — Ableton Live — and the front-end interface objects.

RhythmCAT — see Section 2.2 — was definetely a very big inspiration for our work, although we weren't interested in making a 2-dimensional iteractive space like the ones presented in that section. Mostly we wanted to make a system as autonomous as possible, with very little interaction and, ideally, no interface. We only wanted to give the user the choice to choose its own database.

We struggled to find a reliable way to classify drums. We made some experiences with MIR toolboxes — namely MIR toolbox for MATLAB — and pondered building our own simple classifier in Max environment, until we found LDT — see Section 2.3.1 — and it became clear that this would be included in our plug-in implementation.

Chapter 3

Methodology

3.1 Conceptualization

It is important for us to state that this was never a closed project and it was proposed by the author to the supervisor. Since the beginning we've been adapting our system and objectives according to limitations, experiments, discussion and accidents. This project started with a much more generalist approach, although it never fell of track from the main idea: the proposal of a composition tool. The first ideas leaned towards an harmonic generative tool that would recreate new melodic sequences given a corpus of analyzed and labeled audio samples — similar to *EarGram* that we mention in Section 2.2. After diving into the pitch detection state of the art and making some experiments with available algorithms, we found the problem much more complex than expected, particularly with non-isolated sound sources that were our initial analysis object. We decided to narrow down our spectrum to a simpler field: percussion generation.

We started doing experiences with concatenative synthesis since the very beginning and applying stochastic processes to generate our outputs, but eventually we decided it would be important to have some sort of analysis-synthesis system to confer it some intelligence. We pondered the use of Machine Learning algorithms to analyze databases of rhythms and translate that into Markov Models to focus on the synthesis part. Eventually the challenges of creating the analysis system itself absorbed the major part of the work and the variation algorithm ended up being the result of some happy accidents.

It was also important that the program would result in a production tool with live applications and particularly developed for *Max For Live*. There were some temptations to leave this platform and focus solely on Max/MSP, but we decided to stick to the plan.

3.2 Objectives

The Objectives for the thesis are:

1. To explore the State of the Art of the fields related to our system
2. To learn the tools necessary for its development
3. To experiment new approaches to rhythmic generation
4. To create the system using the Max For Live extension
5. To test and evaluate our work

3.3 Process

We started this project with brainstorming reunions with the intention of defining the specifications of what was a excessively ambitious initial idea. A scientific introduction to the fields involved in these brainstorms helped direct our efforts towards a more realistic objective. We also started experimenting with tools that we would use to complete the project and learn them, particularly the main language Max/MSP about which the author had no prior knowledge before this project, except the use of complete Max For Live devices. These experiments were crucial, as were the analysis of other Max For Live devices for the definition of the concept and the early versions of the algorithm. Along started coming new ideas — they still come everyday — and the whole development phase was based on this method of experimentation and testing.

3.4 Task Planning

We will break down our work in the following individual tasks:

1. Make brainstorming reunions to define the project objectives.
2. Tool Study and Exploration.
3. Explore the fields of interest for the project. Write the Chapter regarding the State of the Art.
4. Learn from scratch Max/MSP and achieve the level of proficiency necessary to make the Plug-In.
5. Define the final concept.
6. Develop the System.
7. Build a reliable audio analyzing system that is capable of identifying units and label them.
8. Build the synthesis process that aggregates those units into meaningful drum phrases. Create a way to convert MIDI files into the system's language.
9. Design a simple but effective interface with minimal controls.
10. Write the thesis.
11. Make user-based tests to evaluate the system performance.

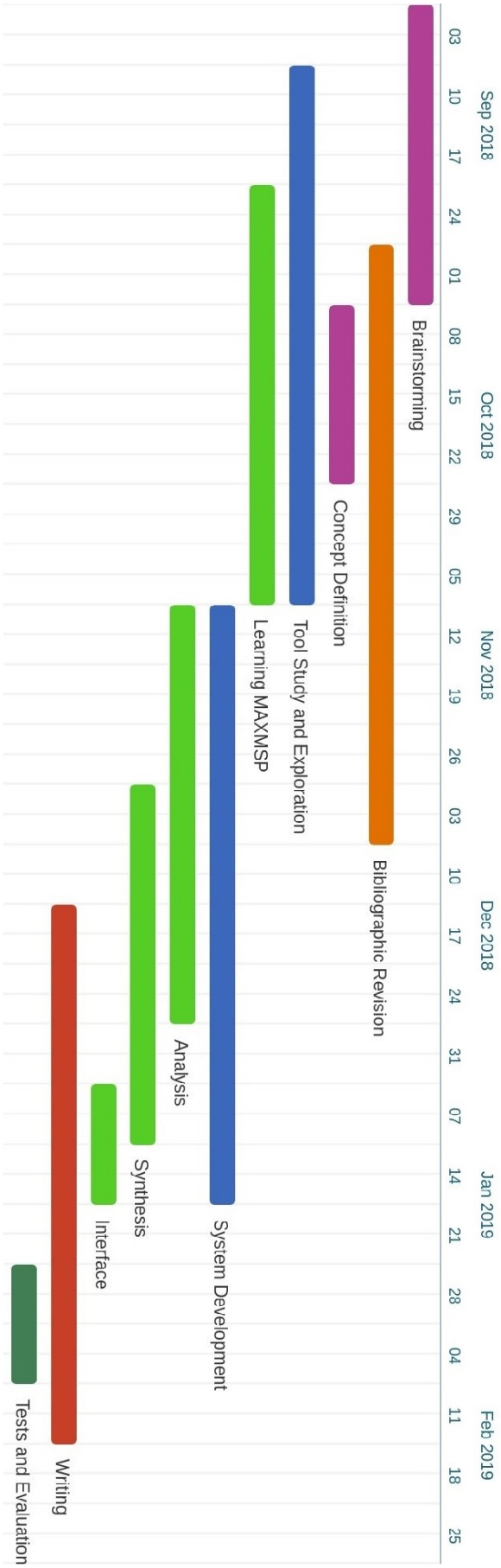


FIGURE 3.1: Task Planning

Chapter 4

System Description

We will devote this chapter to the description of our system. The host program of our application is the Digital Audio Workstation (DAW) *Ableton Live 9* and the application itself will be developed using the *Max For Live* extension, an application that coordinates Ableton Live capabilities with the *Max/MSP* language.

4.1 Definition

Using the Max For Live extension, it is possible to create different kinds of devices to make different kinds of operations in Ableton Live — see Figure 4.1 —, such as:

Audio Devices Plug-Ins that get involved in the Audio Signal Processing Chain, like regular sound effects.

MIDI Devices Plug-Ins that get involved in the MIDI Note Processing Chain.

Instrument Devices Plug-Ins that synthesize sound with respect to MIDI input.

MIDI is a protocol introduced in the 1980s with the purpose of standardizing the communication between electronic music devices. It carries an array of possible controls for live applications and it also includes a file format with the same name — extension *.mid*.

Our Device is an Audio Device, meaning its input and output will be both audio signals. The plug-in receives a drum loop and returns a reorganization of the individual instruments into a new sequence and introduces algorithmic variations, dependant on the user's decisions. The communication with the user will be through the Graphical User Interface tools included in the Max For Live library.

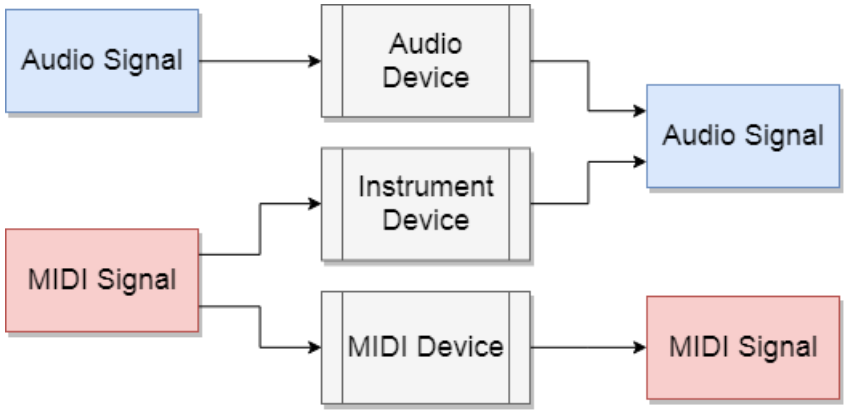


FIGURE 4.1: Diagram describing the different types of devices

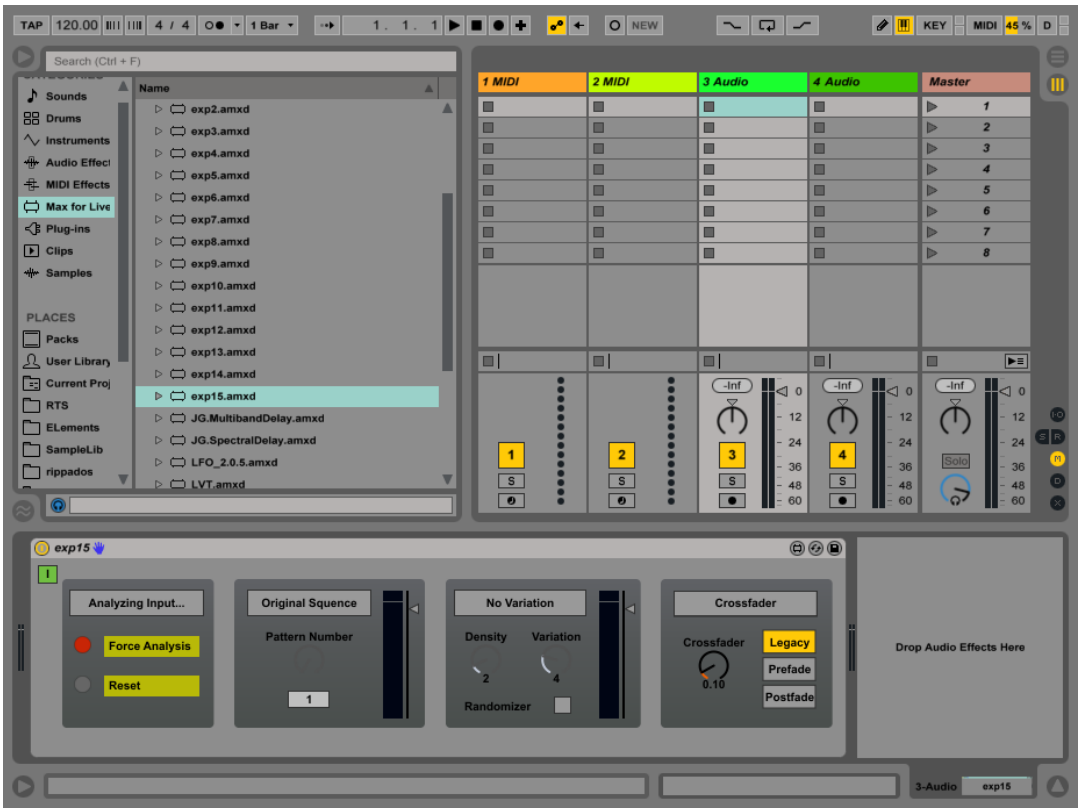


FIGURE 4.2: Ableton Live 9 interface

4.2 Tools

In this section we will make a brief reference to all the software and programming languages required to create the Plug-In we propose.

4.2.1 Ableton Live

Ableton Live, in its core, like all major DAW, works like a traditional Mixer. Different tracks that are treated as individual sound sources that get summed and end up as the output of the Master Track. There are Audio Tracks and MIDI Tracks. MIDI tracks transform MIDI notes into audio signals. Audio Tracks transform audio input into an audio output.

Besides being the audio host of the Plug-In, it will be the synchronization Master to our Slave Max/MSP application, meaning the definition of absolute time lengths — or clock — will be entirely controlled by Ableton — see Figure 4.3. Max/MSP will only work with relative time lengths. We will further describe the definition of our units in 4.3.1.

FIGURE 4.3: Ableton's Temporal Control. Here we can select the BPM — beats per minute —, the time signature and track down the actual arrangement position

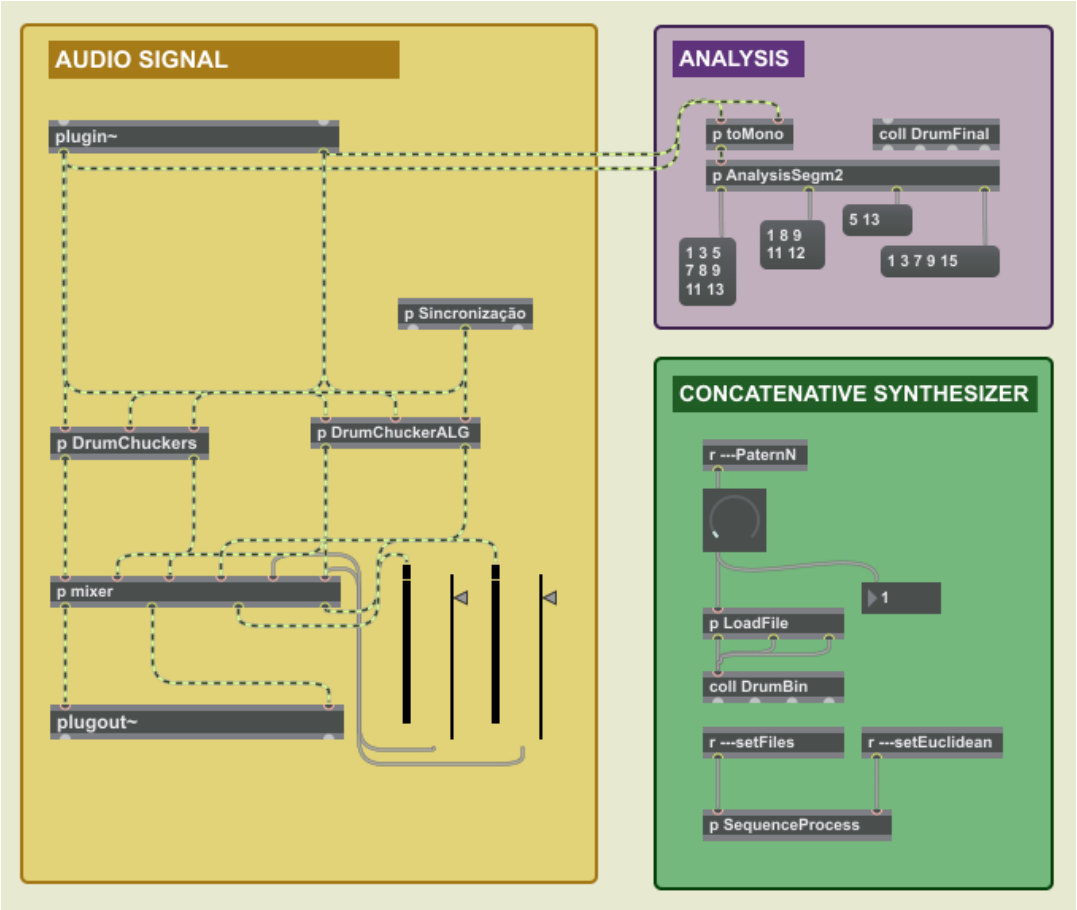


FIGURE 4.4: MAX/MSP final patch

4.2.2 Max/MSP

MAX/MSP is a visual programming language directed towards multimedia applications. It will be our main environment. Data, which can either be numbers, lists, signals or pulses — *bangs* —, flows through the *patch chords* — see Figure 4.4. Our system is synchronized with Ableton and uses its clock to generate bang messages to control the data flow — see 4.4 for an example.

We divided our plug-in back-end into three sections — see Figure 4.4 :

Audio Signal Section that deals with the audio signal flow: input, the mixer section and the output.

Analysis Section that deals with the Analysis of the audio and the segmentation process

Concatenative Synthesizer Section that makes the decisions in the concatenation process and communicates with our sampler.

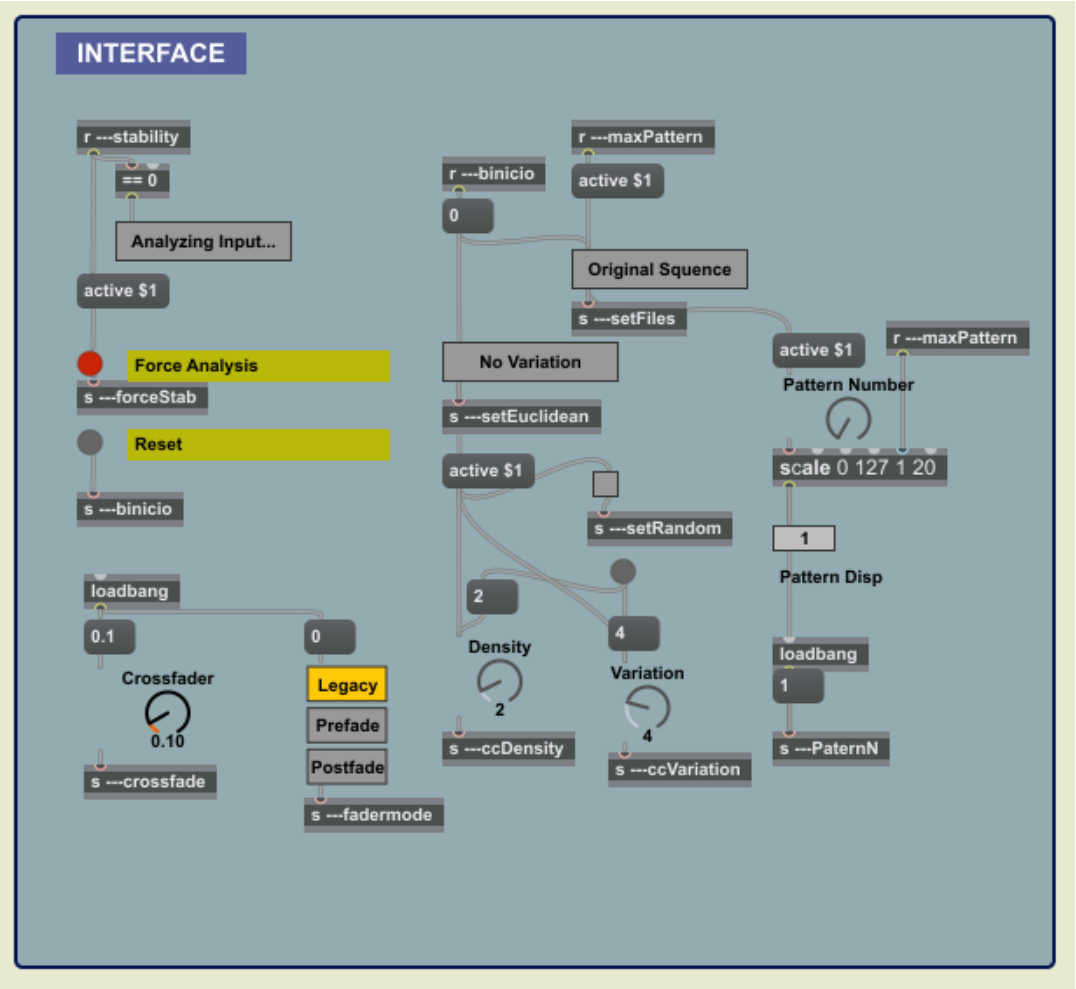


FIGURE 4.5: Interface MAX/MSP patcher

4.2.3 Max For Live

Max For Live is the extension that allows the dialog between MAX/MSP and Ableton Live. It has a series of generic controls available to create the GUI — Figure 4.17. We will explain the controls in Section 4.5.

4.3 Implementation

Throughout this section there will be references to Appendix A where we will include the most important patchers we made. These may not be illustrative for someone who is not familiar with Max/MSP.

4.3.1 Unit Definition

First, it is important to define time signature and bar — or measure in traditional music composition (we will use the word bar as it is the electronic music community denomination, although it may not be a universally accepted synonym to measure). A bar is the total time length defined in the time signature.

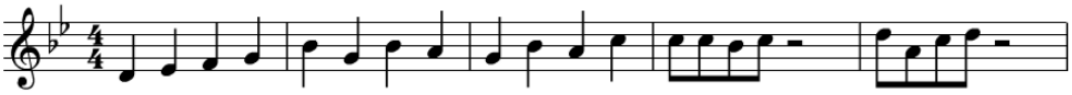


FIGURE 4.6: Example of 5 bars of a music score in ¼ signature

Figure 4.6 shows a music composition with 5 bars. In the beginning, just before the first note, the time signature ¼ is represented, meaning each bar has 4 beats and the kind of notes that are included in that bar are ¼ notes, or, to put it simply, in each bar there are 4 ¼ notes.

The bar gets represented as the space between two vertical lines, hence the name *bar*. It got included in musical notation when music started having more regular rhythms and it served as a visual queue to ease the reading process.

The ¼ time signature is the most common time signature in popular music, having entire genres like hip-hop, techno, house and many others almost entirely composed with it.

The first samplers, drum-machines, synthesizers and sequencers included, at least the ¼, which further augmented its importance for popular culture, as the tools that artists used had it hardwired in its system. To maintain the tradition, our Plug-In also works solely with this time signature. Although it is possible to alter this with some edition, we decided to keep it simple.

FIGURE 4.7: Roland TR-909 Rhythm Composer, a landmark tool in the genesis of house and techno music. Each of the 16 buttons at the button refer to a 16^{th} note or a step. Source: `https://en.wikipedia.org/wiki/Roland_TR-909`

For electronic music, particularly dance music, which has the intention to maintain a steady and predictable rhythmic structure, the bar is the core of the structure. Even when we "zoom out" of the bar temporal dimension and look into the whole musical structure, every main section transition will happen in 4^{th} multiples of the bar.

In the same way, if we "zoom into" the bar, and section it into equally sized segments, using the ratio 4, we will have meaningful units that have been staple in electronic composition since its early days, widely known as *steps* — see Figure 4.7. Of course, this relationship reaches a limit in terms of definition of meaningful units because if we divide it too much our brain does not have the capacity to process those small time segments.

Going back to the unit definition in our program, we will maintain the legacy of the first drum machines and use the bar as our main repetition structure and the sixteenth note as our smallest division. This list resumes our nomenclature:

Transient or Onset The absolute beginning of a new note/instrument. We will use the terms interchangeably.

Step Smallest time unit defined and corresponds to the time length of a 16^{th} note.

Bar Largest unit defined and corresponds to the entire length of a drum loop — Figure 4.8.

Segment Smallest audio unit defined and corresponds to the audio content within the length of a 16^{th} note.

Phrase Audio content within the length of a bar.

Event - Segment that contains an audio onset — Figure 4.9.

Sample Largest group of Segments in between two consecutive events — Figure 4.13.

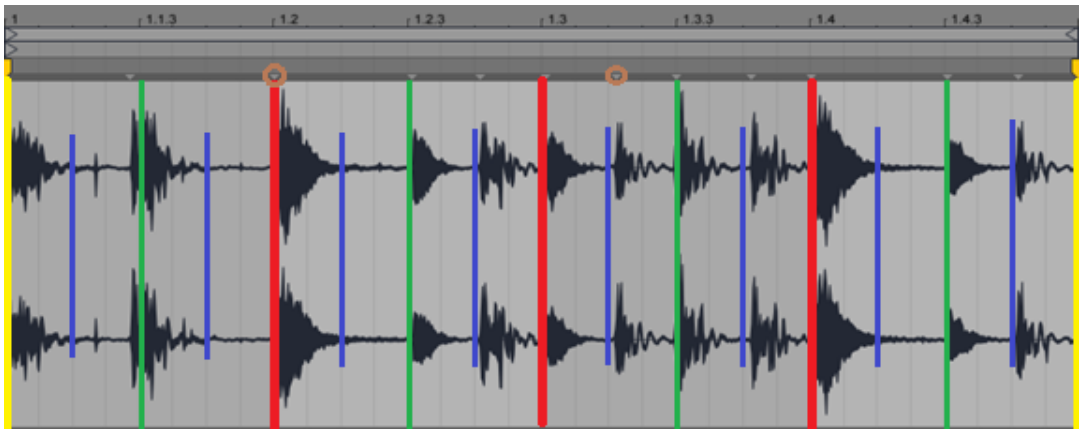


FIGURE 4.8: Time domain visual representation of the first bar of a drum break in *Melvin Bliss – Synthetic Substitution* in Ableton Live environment. The different measurement limits are the vertical lines. Yellow lines represent bar, red for quarter notes, green for eight notes and blue for sixteenth notes. The brown circles underline transient occurrences.

4.3.2 The Sampler: *chucker~*

A sampler in electronic music, is a device capable of extracting an audio portion of a bigger audio file and play it back differently according to the users intention. In this case, our sampler's objective is to receive an audio signal and chop it into individual segments. For that purpose we will use the *chucker~*, a MAX/MSP object that given an audio input, a division measurement — 16 in our case — and a synchronization signal — provided by our Master and Host Ableton Live, to translate our units into absolute temporal lengths —- will divide the phrase into an equal number of segments.

It operates with a bar of delay because it needs to fill its data buffer with audio content. After the first bar, chucker~ is fully operational and can receive another argument: a vector that maps the re-sequencing of the input segments into an output drum phrase with the same length.

Like the first drum machines, this segmentation process is temporally precise, meaning if we are trying to segment a real drum break recorded by a human being it is unlikely to respect this precise measurements —- observable in the third step in Figure 4.8. In order to avoid that and to achieve proper results, we use the *Quantize* function that Ableton Live includes. Analogous to quantization in an ADC process, except this quantization occurs in the time domain instead of amplitude domain: each transient — the absolute beginning of a new sound event, represented by the brown circles in 4.8 — is automatically assigned to its closest quantization interval, in our case, the closest 16^{th} note — see Figure 4.9. This "assignment" is possible through complex schemes of audio stretching that can conserve much of the frequency content — Live itself has different stretching algorithms.

FIGURE 4.9: The same drum break but now Quantized to every 16^{th} note. The yellow markers on top are called Warp Markers and denote transients that were time shifted — in our nomenclature these are the *events* of the drum break

We will refer to the input phrase as *S(x)* — see Figure 4.9 — and the output phrase as *Y(x)* and *x* will refer to the Segment position. An obvious conservation of the input phrase would be Y(x) = x. Say, we want to loop the first 4 segments throughout the bar, to create a more representative example — 4.1 illustrates that.

TABLE 4.1: Mapping of a loop of the first 4 steps

x/Step	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Y(x)	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4

To communicate with *chucker~* what is the desired output sequence, we send him a list – the MAX/MSP version of a vector: a message/string composed of spaced numbers — Patcher A.1. This list has 16 elements, in a range 1-16. The index of the list states the step, while the actual value of each index corresponds to the segment itself S(x) — see 4.9. Again, to produce the original phrase:

Y = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]

We dropped the (x) in notation because the index itself states the step value, so there's no need for it. And to produce the reverse sequence it would be:

Y = [16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1]

We also need to send a Boolean list with 16 elements, stating witch steps are supposed to be played and which ones are silent. Although *chucker~* also accepts the value -1 in this list to assign a certain segment to play backwards, we will not include that feature in our device and will conserve forward playback direction at all times. We will call this *D* message. To have a complete example, these would be our *chucker~* arguments if we wanted to reproduce only the first half of the original phrase:

Y = [1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0]
D = [1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]

We put the zeros in the 2^{nd} half of the argument, although its content is irrelevant because there are 0's in D, meaning those steps will be silent. There will be 4 distinct *chucker~* instances running at the same time in order to make it possible to have polyphony: one *chucker~* for each drum sound and one extra to run a variation algorithm.

4.3.3 Segment Analysis

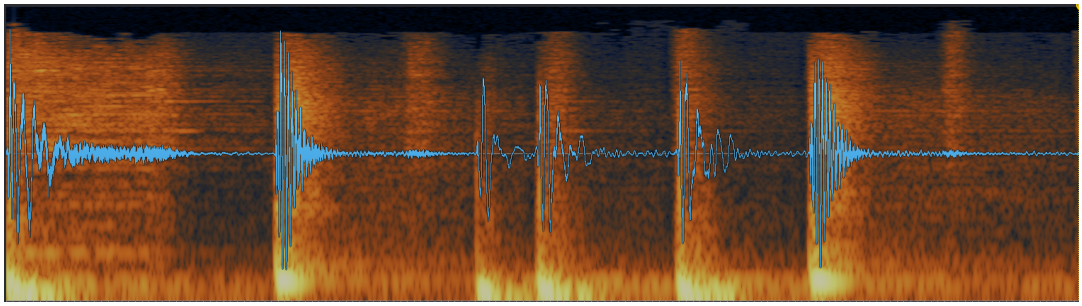


FIGURE 4.10: Spectral Representation of a drum solo excerpt from *Aerosmith - Walk This Way*. Lighter colors represent a higher amplitude in a particular frequency band.

In this section we will address the whole analysis process. We have created an info-graph to illustrate it. It was designed using a drum solo excerpt from the song *Walk This Way* by Aerosmith — see Figure 4.10 for spectral representation and Figure 4.11 for the info-graph. We also include an illustrative patcher of the process — Patcher 4.12.

Like mentioned in Subsection 2.3.1, we will be using the Live Drum Transcription device created by Marcus Miron (2013) to classify events — Patcher A.2. We sum both stereo channels to create a single mono signal for the analysis — although we recognize the potential of doing individual analysis on each stereo signal, it doubles the probability of system failures and we found LDT and *aubioOnset~* crashing Ableton with some frequency in the prototyping system. The device receives an input audio signal and outputs bangs out of 3 outlets which state the occurrence of an event in each of the 3 classes.

The lists with the segments that are associated with each class — events — are stored in a collection called *DrumAnalysis*. Whenever Ableton enters Play Mode, the analysis starts too. LDT is inconsistent and sometimes the output varies, and for that reason, analysis will run until there are 4 consecutive bars that output the same event list for each class. When this happens, the program enters *Stability* mode: analysis stops and we process *DrumAnalysis*. We would like to make a continuous analysis in order to improve the workflow of the user, but to avoid crashes, we made it this way. It is possible to use the device without the Stability, although it won't include all its functionality.

LDT can sometimes output ghost events, particularly in the Hi-hat class. We run an onset detector called *aubioOnset~* (`https://aubio.org/`) in hfc mode — high frequency content — as it is a very reliable onset detector and we also store a list of segments that include an onset — *Events*. We then match *DrumEvents* lists with *Events* list and eliminate the ghost events in *DrumEvents* — segments that were transcribed as a drum sound but weren't detected by *audioOnset* — and store the remainder in another collection called *EventFiltered* — Patcher A.3.

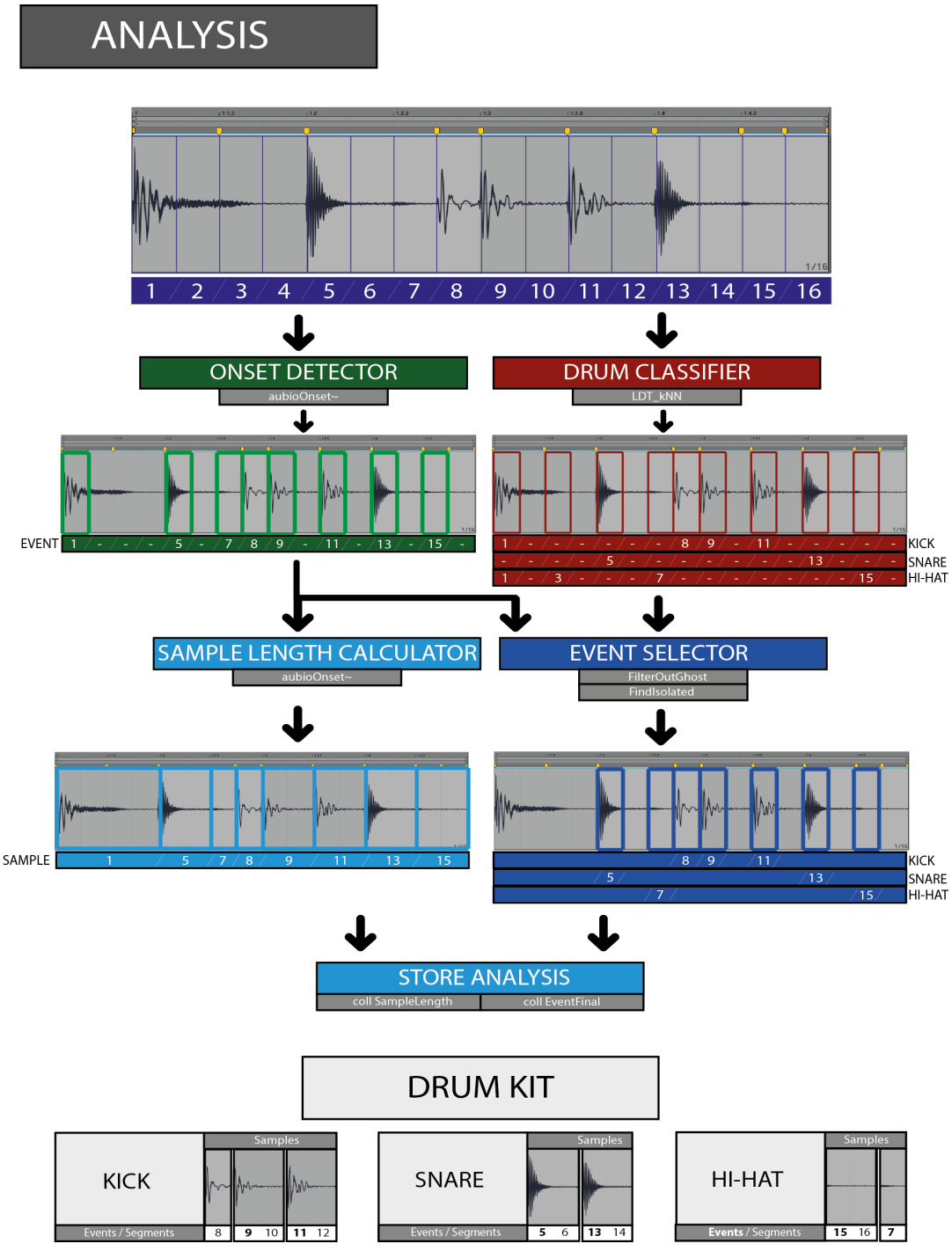


FIGURE 4.11: Diagram of the whole Analysis Process using the same file as 4.10. Through this figure, we can see how the program creates his database/drum kit from the input drum phrase.

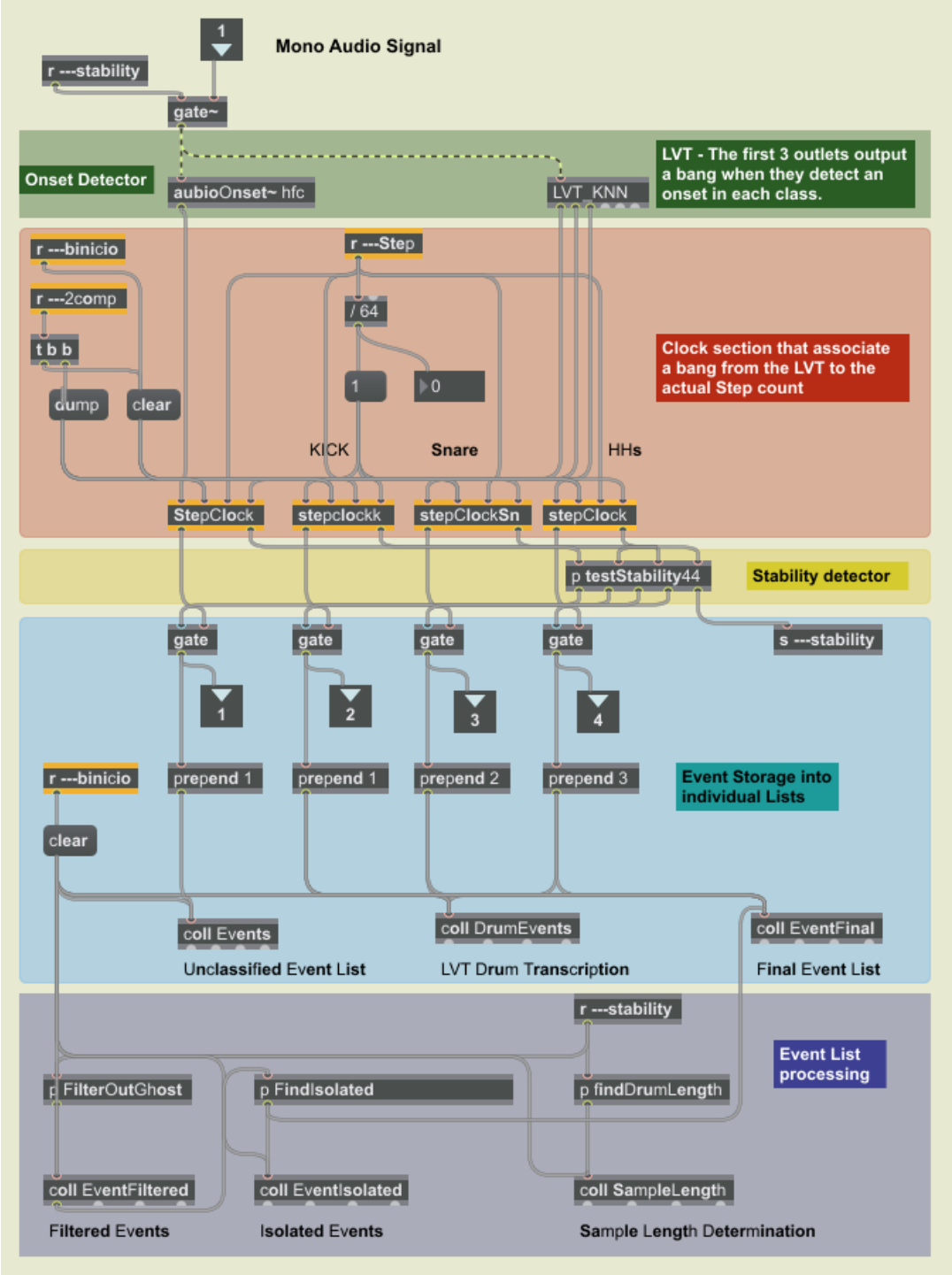


FIGURE 4.12: ***AnalysisSegm:*** Main patcher that deals with classification of segments and sample selection

In drum phrases, it is common to have multiple instruments being played at the same time, and LDT can classify a segment as multiple instruments. Obviously, it is beneficial to work with isolated drums if possible, so we created a patcher that tests which events are only associated with a single class. In case our input drum bar doesn't include any isolated drums, we maintain the *EventFiltered* results — Patcher A.4.

One of the problems with using chucker~ to segment the phrase is that it segments in fixed intervals, independently of the sound characteristics. As a consequence it may happen to cut a *Sample* that lasts longer than the period of a segment — see Figure 4.13 — and because LDT is mainly based on onset detection, the segments associated with drum sounds are only the ones that contain an onset. We also made a patcher — Patcher A.5 — that corrects this by measuring a distance between consecutive entries of the *Events* list. This way when we intend to play a sample, we can play all the segments that are involved in it in sequence.

FIGURE 4.13: First bar of a drum break from *Aerosmith - Walk This Way* to illustrate what can go wrong with fixed length segmentation. Between the first 4 segments, only S1 contains an onset — is an event —, although the drum lasts at least 3 more steps. We associate the event with its length. In this case the length is 4 segments because the next event is located in S5. So, the sample 1 includes the segments S1, S2, S3, S4.

4.4 Concatenative Synthesizer

4.4.1 Main Sequence

Like stated before, we will be using 4 different *chucker~* instances to manage each of our concatenative voices. The first 3 *chucker~* will be associated with the *Main Sequence*, a single bar structure that will serve as the base loop over which we will layer our variation algorithm.

We offer the possibility to reproduce MIDI files. We created a MATLAB script to convert MIDI files into *.txt* files stored in the plug-in folder, that can be interpreted by our system as collections — see Appendix B.1.

A MIDI file is composed by a series of MIDI notes. Each note has, among others, the following descriptors:

Pitch Numeric value between 1-127. Encodes the value of the pitch associated with the note — i.e. A4 is encoded as 69. In our case, because we are handling percussive values only, this pitch value is only related to the association of each note to a particular drum class, hence our need to define the global arrays in B.1.1. The array values depend on the Drum Articulation established by the creator of the MIDI file — see Figure B.1

Velocity Numeric value between 1-127. Encodes the velocity of that particular note. We will not include the velocity functionality in this version, although there is room for its inclusion as stated in Section 6.

Note On Relative value that states the temporal beginning of the particular note in respect to its position according to the time signature. Since we will be using a fixed ¼ signature and the length of a bar, the extracted values are in the range 1-16.

Note Off Same as the above, except this one states the note's ending.

We will store a *.txt* file for each drum class, e.g. *KickSeq.txt*: after reading the MIDI file using *MATLAB and MIDI*, we will search for the occurrence of notes associated with the kick class, group them, extract the individual *Note On* values and store them in a line of the file. This process happens simultaneously for the other classes and we maintain the line number in every text file as an identifier of a particular MIDI file conversion — Patcher A.6.

4.4.2 Euclidean Rhythms

The other *chucker~* is meant to be layered with the Main Sequence. In order to generate new notes we will have a random process running with the periodicity of one bar. This process generates notes according to the Euclidean Rhythms.

Euclidean Rhythms are a family of rhythms computed through the Euclidean algorithm — named after Euclides of Alexandria which described it in the 7^{th} book of his seminal work *Elements* — that has been used in mathematics for millenniums to compute the greatest common divisor given two integers. It's ability to generate evenly spaced patterns has seen applications in string theory, computer science and nuclear physics. This same algorithm can be used to elaborate a series of rhythmical ostinatos that describe rhythmical patterns found in various traditional music cultures all over the world. Their relationship to music was first discovered by Godfried Toussaint and is described in (2005).

The formulation of the rhythmical sequences is based on the computer algorithm invented by Eric Bjorklund (2003). He used a binary representation of the problem which can also be used in rhythmical description. Each bit describes an equal time interval, 1's represent the occurrence of an event — the onset of a note - and 0's represent the absence of events — a silence. A common and short way of representing these binary sequences is *E(m, k)*, with m being the length of that sequence and k the number of events in it.

Let us work through the algorithm of a simple sequence *E(10, 4)*:

1. We start by putting all the events and silences together:
   [1 1 1 1 0 0 0 0 0 0]
2. Then we pair each 0 with a 1, working from left to right:
   [1 0] [1 0] [1 0] [1 0] [0] [0]
3. We are left with two remaining 0's — the remainder — so we will group them with each [1 0] pair, again from left to right:
   [1 0 0] [1 0 0] [1 0] [1 0]
4. Now, the remaining [1 0] pairs will be grouped with the [1 0 0] trios:
   [1 0 0 1 0] [1 0 0 1 0]
5. The process ends when the remainder consists of a single set:
   E(10,4) = [1 0 0 1 0 1 0 0 1 0]

Because of the algorithm's left-to-right approach, we always generate sequences that begin with an event. We can then include another variable — offset: *o* in *E(m,k,o)* — which will rotate the whole sequence by an integer value. For example:

$$E(10,4,1) = [0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1].$$

A visual representation of this can be achieved using a regular polygon — see Figure 4.14. To generate these Euclidean sequences we used the Max abstraction *11euclidean*.

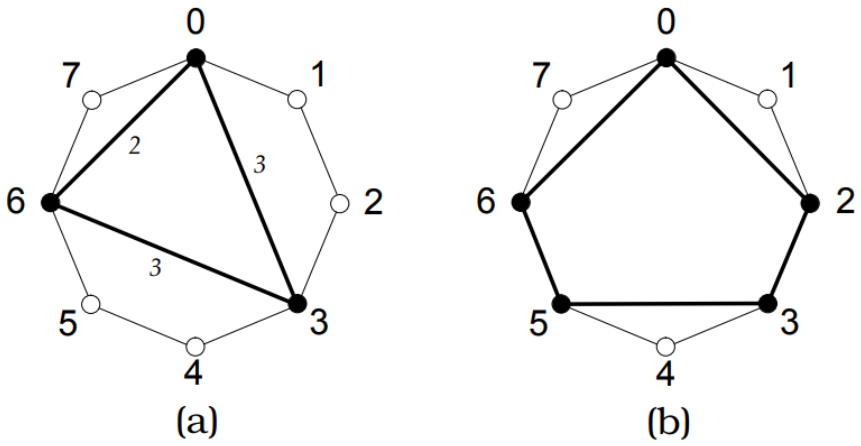


FIGURE 4.14: Geometric Representation of Euclidean Rhythms *E(3,8)* (a) and *E(5,8)* (b). Each vertex of the polygon represents the onset of each time division, the black vertexes represent the events and the white ones the silences. Source: (Toussaint, 2005)

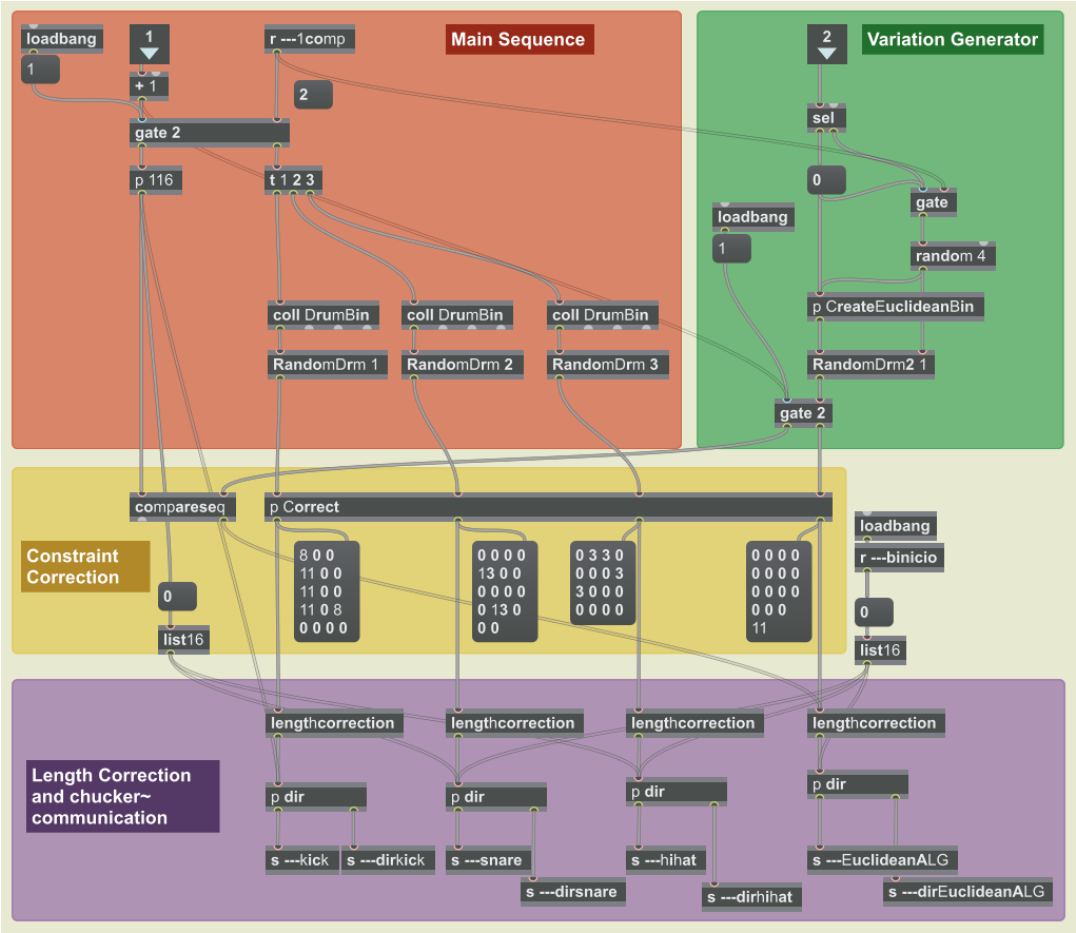


FIGURE 4.15: Patcher illustrating the whole concatenation process.

4.4.3 Concatenation Process

Since we are now working with samples that may contain more than a segment, we need a new form of sequence representation, we will call it *O(x)*, and its a binary sequence that states the occurrence of events — not to be confused with *D(x)* that states where there is reproduction of audio content and silence. All our values in this subsection will be based in the analysis example in Figure 4.11.

Besides rhythmic variation we included another type of variation in our concatenative synthesizer, that deals with unit selection. Given that we already know which steps will include events, both in the Main Sequences and in the Euclidean Rhythm, and represent that in the form of a binary sequence, we can now handle the sample assignment. Our selection is simply aleatory. If we have multiple samples classified as the same instrument, when we need to reproduce that particular instrument we will randomly assign one of the classified drum events. Example: if want to reproduce the sequence of events of snares with the following binary representation and use the unit decision variation, one alternative could be:

$$O_{SNARE} = [\,0\,0\,1\,0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,]$$
$$Y_{SNARE} = [\,0\,0\,5\,0\,0\,13\,0\,0\,0\,13\,0\,0\,5\,0\,0\,0\,]$$

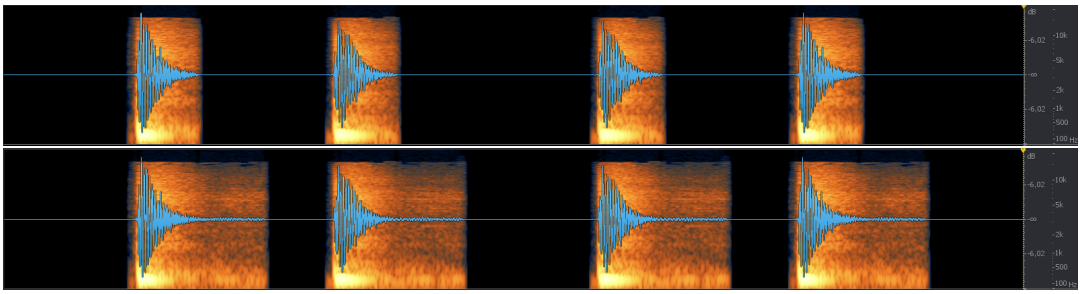


FIGURE 4.16: Spectral representation of the mentioned Snare sequence. The upper spectrum is the sequence containing only the events associated with the snare sounds. The lower spectrum concatenates the event with the remaining segment that is part of the sample.

But since we now know that the samples associated with events 5 and 13 contain more than one segment — two segments each — we have to correct the argument messages in order to make *chucker~* reproduce the whole sample lengths. See Figure 4.16 for an illustration of this.

$$Y_{SNARE} = [\,0\;0\;5\;6\;0\;13\;14\;0\;0\;13\;14\;0\;5\;6\;0\;0\,]$$
$$D_{SNARE} = [\,0\;0\;1\;1\;0\;1\;1\;0\;0\;1\;1\;0\;1\;1\;0\;0\,]$$

Even though we use a single *chucker~* for each instrument to achieve polyphony, a single *chucker~* is only capable of reproducing one audio stream. It could happen that a particular sample would overlap with another, but our system doesn't allow it. We do so by prioritizing drum events over full sample playback, because with percussion the most important part of the sample is always the onset — Patcher A.7. Going back to our previous sequence as an example. If we have an extra snare event on step 2 and it gets assigned the sample 13 from unit selection which has a sample length of 2 steps and overlaps with a posterior event — on step 3 — we cut its length to the maximum possible length that prevents overlapping:

$$O_{SNARE} = [\,0\;\mathbf{1}\;1\;0\;0\;1\;0\;0\;0\;1\;0\;0\;1\;0\;0\;0\,]$$
$$Y_{SNARE} = [\,0\;\mathbf{5}\;5\;6\;0\;13\;14\;0\;0\;13\;14\;0\;5\;6\;0\;0\,]$$
$$D_{SNARE} = [\,0\;\mathbf{1}\;1\;1\;0\;1\;1\;0\;0\;1\;1\;0\;1\;1\;0\;0\,]$$

4.5 Interface

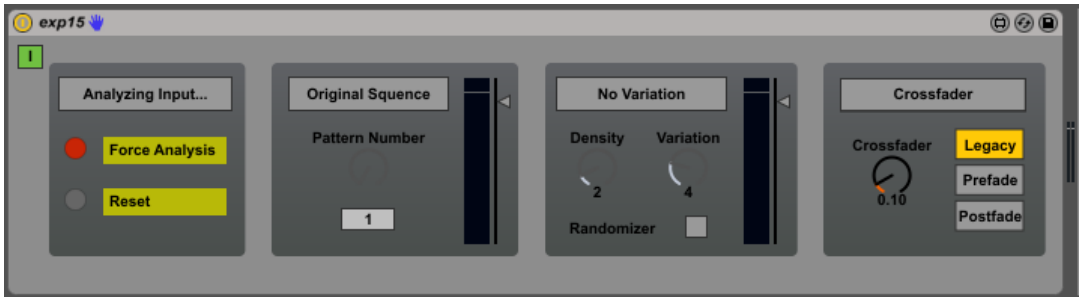


FIGURE 4.17: Max For Live integrated interface in Ableton environment.

In Figure 4.17 we present our front-end of our software. We will explain the function of each control:

Force Analysis Force the analysis stoppage and begin the concatenation process with the current analysis parameters.

Reset Reset the analysis process from the beginning. Useful in case the plug-in made a wring analysis or if the user wants to change the input drum phrase.

Original Sequence / MIDI files Button that toggles the Main Sequence option.

Pattern Number Select which MIDI file should be reproduced in the output. The knob automatically sets its range according to the number of converted MIDI files in the saved text files.

No Variation / Euclidean Algorithm Button that toggles the Variation option — see equation below.

Density Minimum value of events in the Euclidean algorithm.

Variation Range of possible events in the Euclidean algorithm.

Randomizer Allow the assignment of unfiltered events in the Variation.

Crossfader Control the crossfader amount in between segments.

Legacy / Prefade / Postfade Assign the crossfade mode of all *chucker~* objects.

$$E(m, k, o) \quad \text{where,} \quad m = Density + Rnd$$

$$Rnd \in [0, Variation]$$

4.6 Conclusion

In this section, we presented the System implementation. Throughout its development, there were many features included and excluded. In the end we decided to maintain only what was reliable and what sounded musical.

Our inability to find a drum classifier in the beginning started steering our plug-in in a purely synthesis-based application, which wasn't the primary goal. The inability to properly classify segments would limit our work, and we would have to, either make an entirely aleatory rhythmic generator, give much more importance to the user and the interface or design our own classifier. Although we feel like those could also have been interesting implementations, they were not along the lines of our plans — some unfinished prototypes before the classification was included were already sounding interesting.

Accidentally, we started working with a subset of the Euclidean Rhythms, and started applying that particular case to unclassified rhythms phrases When we started investigating into rhythm generation algorithms, Euclidean Rhythms came up and we discovered this relationship with our initial sequences. Besides its inherent musicality, that was a reason why we decided including Euclidean Rhythms in the variation algorithm,.

The MIDI file conversion was also another adaptation of a different idea. We wanted to create a HMM for rhythm generation through Machine Learning analysis of a database of MIDI files. This experiment mostly failed, but we already had the MATLAB script made to translate the MIDI files, so we decided to include it too.

Finally, our objective to have a non-existent interface didn't seem like a very good idea given the particularities of our system, of the *chucker~* object and the need to adapt the crossfader section to different drum loops. We didn't want to make it too complex either so a solution in the middle of the way — a reasonable amount of automation and control — seemed like the best idea.

Chapter 5

Tests and Evaluation

In this chapter we will explain how we proceeded to test the Plug-In performance. Our tests were user-based. Various users with different backgrounds and objectives experimented the Plug-in and then submitted a response in a *Google Forms* with a series of questions.

5.1 Testing Method

One of the main problems this application imposed was to find testing method that would retrieve meaningful results. Since we designed it as a production tool, it made sense to have it tested by people who had contact with Ableton Live, as it is the only DAW capable of running it. This somehow constrained the conditions in which our tests could be done. Despite the fact that Ableton Live is a very popular application among musicians, the use of the Max for Live extension is not so popular and part of our testers had no contact with it. Plus, our software was developed in 32-bit architecture, meaning both MAX/MSP and Ableton versions in the testing computer had to be installed accordingly, which really narrowed down our possibilities. Another problem was the impracticability of the MATLAB script to generate the MIDI sequences for those who weren't used to MATLAB nor owned the software.

Ultimately, we found that taking the tool to the testers in a computer already set up properly was the best idea. This meant various things:

- We could personally explain how the system worked. This meant the user had some understanding of the tool but also meant that it became hard to evaluate the interface accessibility.
- The user was unaware of the work involved in the MIDI file conversion. Even though the script is not hard to understand, it does take some time to use the MATLAB script to convert the MIDI files. It was explained to the users, but they had already previously converted files to make their experience focused on the plug-in.
- We could select a series of examples that worked well and a series of examples that wouldn't work with the software, although we also gave the opportunity to the user to experiment his own choices.
- It allowed us to talk with the users about the benefits and the disadvantages of the Plug-In and discuss some future improvements.

.

5.2 Questions

The form was created with the Google Forms platform and the questions were written in Portuguese, as most the testers dominated the language better than English — see Figure 5.1. The translation of the questions to English follows:

1. On a scale 1-5, classify the utility of this tool in a music production context:
2. On a scale 1-5, classify the utility of this tool as a percussive accompaniment for to practice an instrument:
3. On a scale 1-5, classify the usability of the tool:
4. On a scale 1-5, classify the quality of the variation generator:
5. What other functionality would you like to see in the plug-in?
6. What is your relationship with music:

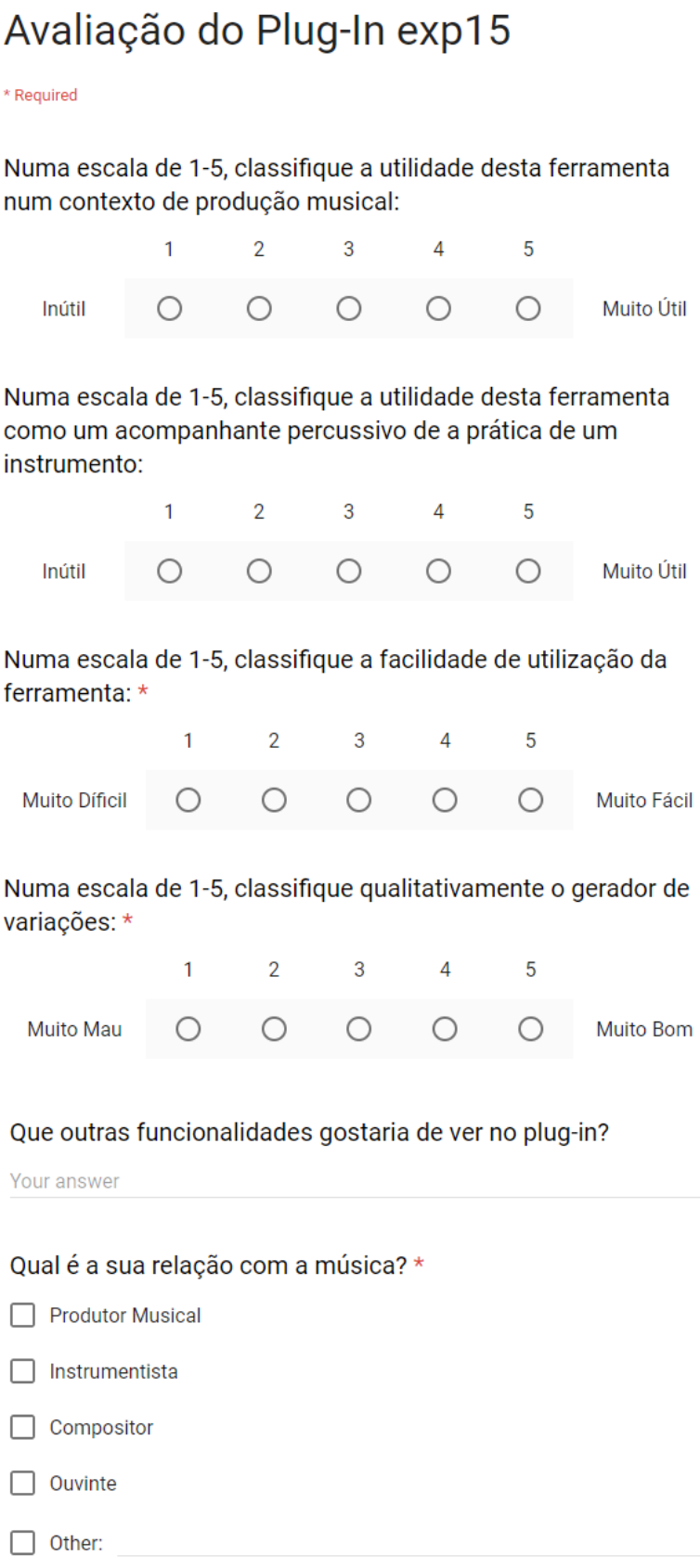
5.3 Answers

We tested 10 different individuals with different relations with music. All of them had experience with interacting with music, not just listening, but either played instruments, broadcasted radio shows, played records as DJs or were actually involved in music production/composition. The results were fairly positive in most questions. The variation generator received a lot of good comments as sounding "genuine" or "rhythmically coherent" — see Figures 5.2 and 5.3.

There was a couple negative commentaries regarding unit segmentation, which we had previously diagnosed. That is mostly a problem of *chucker~* and its fixed segmentation grid, even with the time-stretching Quantization — see Figure 4.3.2. The crossfader options are too limited and even with some tweaks on the crossfader section, it never really sounds as good as a regular sampler with an ADSR (Attack-Decay-Sustain-Release) type of envelope.

Another comment we received regarded the length of the loops. Since we only had a 1 bar loop running, it could be a good addition to be able to use longer drum loops so it wouldn't be so repetitive. That is actually just a problem of our implementation, and again using *chucker~* as our main engines complicates a lot of these ideas, even though we also considered them.

Finally, another criticism we received regarded the inability to pause the program and maintain the analysis. That was also a problem of our implementation, but possibly this one could be avoided. It was something we implemented from the very beginning, as we invested quite some time in the analysis section, and it became convenient to restart the analysis by pausing and playing in the debugging stage. Eventually, it became a central part of the program and would take a lot of time to fix, so we just decided to keep it that way and focus on other, more urgent and fracturing problems.



Avaliação do Plug-In exp15

* Required

Numa escala de 1-5, classifique a utilidade desta ferramenta num contexto de produção musical:

	1	2	3	4	5	
Inútil	○	○	○	○	○	Muito Útil

Numa escala de 1-5, classifique a utilidade desta ferramenta como um acompanhante percussivo de a prática de um instrumento:

	1	2	3	4	5	
Inútil	○	○	○	○	○	Muito Útil

Numa escala de 1-5, classifique a facilidade de utilização da ferramenta: *

	1	2	3	4	5	
Muito Díficil	○	○	○	○	○	Muito Fácil

Numa escala de 1-5, classifique qualitativamente o gerador de variações: *

	1	2	3	4	5	
Muito Mau	○	○	○	○	○	Muito Bom

Que outras funcionalidades gostaria de ver no plug-in?

Your answer

Qual é a sua relação com a música? *

- ☐ Produtor Musical
- ☐ Instrumentista
- ☐ Compositor
- ☐ Ouvinte
- ☐ Other:

FIGURE 5.1: Google Forms used to evaluate the Plug-In.

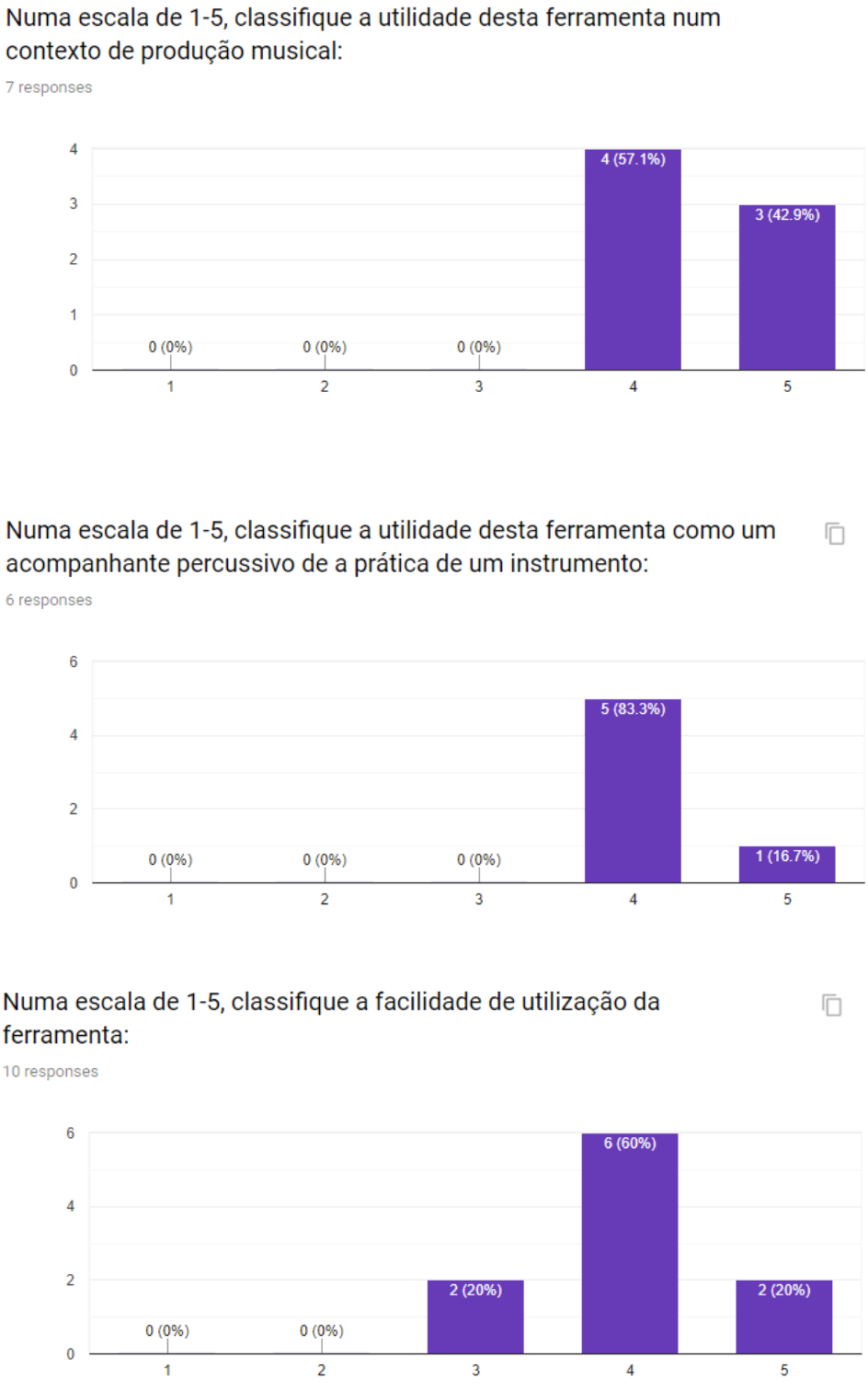
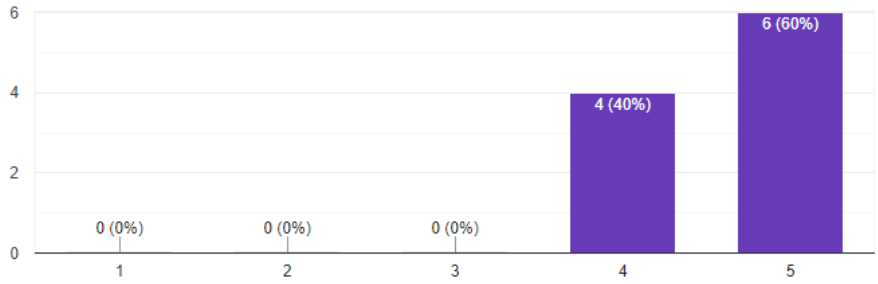


FIGURE 5.2: Answer charts for questions 1, 2 and 3.

Numa escala de 1-5, classifique qualitativamente o gerador de variações:

10 responses



Que outras funcionalidades gostaria de ver no plug-in?

4 responses

alterar o corte de cada som (mais cordadoe sintetico, ou com fade e mais natural)

um período de repetição mais comprido

Opção de Freeze

não parar quando pausa

Qual é a sua relação com a música?

10 responses

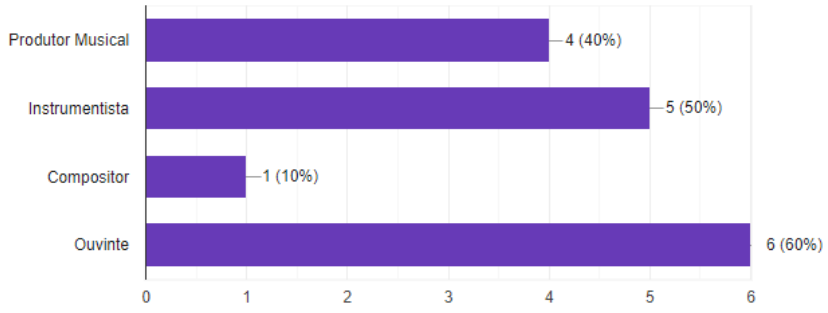


FIGURE 5.3: Answers charts for questions 4 and 6. Written answers to question 5 in Portuguese.

Chapter 6

Conclusions and Future Work

This thesis aimed to create a functional Plug-In that would be useful in the electronic music creation context. We made a state of the art on algorithmic composition, concatenative synthesizers and MIR research. We took inspiration from the bibliographic revision and, from that, defined an idea of what we could achieve and what concepts we could apply to create a system. After experimenting with melodic material, we decided to create a concatenative synthesizer focused on rhythmic-centric loop recreation. The environment chosen was the Max/MSP programming language for the back-end development, and the extension Max For Live for the DAW Ableton Live as, respectively, user-interface and host program. Our system is able to receive a drum phrase — using traditional ocidental drum instruments: Kick, Snare, Hi-hat —, segment that drum phrase into individual samples and resequence those drum samples into new drum phrases. These new drum phrases can be selected from the conversion of MIDI files, for which we provide a MATLAB script. The Plug-In includes a functional variation algorithm that applies euclidean geometry concepts to, additively, modify the drum phrases.

Various users were asked to experiment the Plug-In, evaluate it and comment it, which resulted in positive feedback. Besides having a few fixed questions for every user, we also had some productive dialogs regarding the utility and the future of the plug-in. Some of them we will address in the next Section.

We feel we have achieved our initial goal, even though we were not able to implement some of the functions we intended. Maintaining an audio source as the main creation tool was always a priority, even when it became clear that using just the MIDI protocol seemed like an easier and much more practical implementation. It lacked our initial objective: the ability to automatically create the database and make a system that would work with "anything" the user wanted — of course, we have restricted it along the way and our database became limited to the buffer size of our sampler object. Still, the system is competent within its limitations. Possibly with some future developments we can see it being used in a wider palette.

One of the main objectives that we could not address, was the creation a rhythmic generator through database analysis. Some attempts were made at creating an Hidden Markov Model. We believe we could have created a solution like that if we did not propose to create this system and focused on that aspect solely.

Still, we created a system that is unique. We made this sort of implementation to work in live applications, but this kind of analysis would make sense in a lot of systems. Ableton Live, for example, when it automatically segments a drum phrase it does not label each segment like our program does. This can be time-saving for people who rely on this technology on an everyday basis.

Ultimately, it sounds musical. And the individual samples sound good together. The database is created by the input drum phrase and, most the time, these drum phrases were recorded in a properly tuned drum kit and played by a professional drummer. Even if we change the sample sequence, they come from the same source. That is also a reason why this kind of implementation works.

6.1 Future Work

We recognize that a lot of what was done in Max/MSP is not the best way to implement that particular solution. Like mentioned before, this thesis also involved learning from scratch the programming language, and that process overlapped with the creation of the system. In this section we will address some possible improvements to the plug-in. A lot of them weren't added for lack of time and some of them just wouldn't fit the plug-in in its current state, but we're open to reformulate some of its features:

chucker~ Our main sampler started being used as a prototyping tool, but soon it became central to the system as it was so simple to understand and use and our understanding of Max/MSP was very limited. During the course of the development stage, we started realizing all its flaws and how we were limited by it, or, at least, how it made the implementation of new ideas way more complicated than they should. There was the fixed length segmentation, that we mentioned many times throughout this thesis, that would be a permanent problem given the current state of the object. So we would definitely rethink the segmentation department. This also means opening up the horizons for expressive simulation, velocity variation, unlimited bar length, multiple envelope modes, different time signatures and all other possibilities that most sequencers already contain.

MATLAB script It is very simple and can be definitely done in JavaScript, which MAX/MSP supports, meaning everything could be done in the Max For Live interface. It was done in MATLAB because this is the language which the author is most proficient.

Classification We have used the LDT classifier, which works good enough for what we wanted to achieve here, but if it was possible to have even more classes and more accuracy, we could widen the palette of drum phrases the system could work with.

Offline MIDI device Taking the ideas implemented here, we could implement a similar MIDI device that segments audio and automatically labels each segment.

Velocity MIDI notes also store velocity information and LDT, besides drum classification, also outputs velocity estimation in every event. We did not include velocity in this program as *chucker~* had no ability to easily handle amplitude. We considered creating a volume envelope sequencer, but we didn't implement it.

Rhythm Generation Although we have created a plug-in that is able to create rhythmic variation, we do not consider it to be generative. There are many database-oriented Machine Learning approaches to rhythmic generation that would make sense to include in a system of this kind, like analyzing a database of MIDI files from a genre and create an generate new drum phrases of that genre, or analyzing different drum solos from a particular drummer and create an automatic improvisation tool that mimics his style.

Melodic Content As we have mentioned before, our first objective was to work with melodic content. We doubt a system similar to this would be suitable for melodic content without being extremely limited, given the complexity of the classification, but another system, with all the experience acquired in the development of this one may be a possibility.

Bibliography

Anders, Torsten and Eduardo R. Miranda (2011). "Constraint programming systems for modeling music theories and composition". In: *ACM Computing Surveys* 43.4, pp. 1–38. ISSN: 03600300. DOI: 10.1145/1978802.1978809. arXiv: 9808040 [quant-ph]. URL: http://dl.acm.org/citation.cfm?doid=1978802.1978809.

Ariza, Christopher (2005). "Navigating the landscape of Computer Aided Algorithmic Composition Systems: a definition , seven descriptors , and a lexicon of systems and research". In: *Proc. Int. Computer Music Conf.*

Bader, Rolf (2018). *Springer Handbook of Systematic Musicology*.

Benson, Dave (2006). *Music: A Mathematical Offering*. Cambridge University Press. DOI: 10.1017/CBO9780511811722.

Bernardes, Gilberto, Matthew E. P. Davies, and Carlos Guedes (2017). "A Perceptually-Motivated Harmonic Compatibility Method for Music Mixing". In: *Proceedings of the International Symposium on CMMR*, pp. 104–115.

Bernardes, Gilberto, Carlos Guedes, and Bruce Pennycook (2013). "EarGram : An Application for Interactive Exploration of Concatenative Sound Synthesis in Pure Data EarGram : an Application for Interactive Exploration of Concatenative Sound Synthesis in Pure Data". In: March 2016. DOI: 10.1007/978-3-642-41248-6.

Bjorklund, E (2003). "The theory of rep-rate pattern generation in the SNS timing system". In:

Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet (2017). "Deep Learning Techniques for Music Generation - A Survey". In: arXiv: 1709.01620. URL: http://arxiv.org/abs/1709.01620.

Brossier, Paul M (2006). "Automatic Annotation of Musical Audio for Interactive Applications". In: August.

Cœuroy, André (1928). *Panorama of Contemporary Music*.

Di Scipio, Agostino (2006). "Formalization and Intuition in Analogique A et B". In: *Definitive Proceedings of the "International Symposium Iannis Xenakis"* May 2005.

Edwards, Michael (2011). "Algorithmic Composition: Computational Thinking in Music". In: *Commun. ACM* 54.7, pp. 58–67. ISSN: 0001-0782. DOI: 10.1145/1965724.1965742. URL: http://doi.acm.org/10.1145/1965724.1965742.

Fernández, Jose David and Francisco Vico (2013). "AI Methods in Algorithmic Composition: A Comprehensive Survey Jose". In: 33.3, pp. 973–982. ISSN: 10769757. DOI: 10.1613/jair.3908. arXiv: 1402.0585.

Gibson, Benoit and Markis Solomos (2013). "Research on the First Musique Concrète : The Case of Xenakis ' s First Electroacoustic Pieces Benoît Gibson 1 . Research on Xenakis ' s musique concrète pieces". In: June, pp. 1–9.

Harley, James (2004). *Xenakis: His Life in Music*.

Hiller, Lejaren (1981). "Composing with computers: A progress report". In: *Computer Music Journal* 5.4, pp. 7–21.

Hodgkinson, Tim (1987). "An interview with Pierre Schaeffer–pioneer of Musique Concrète". In: *ReR Quarterly* 2.1.

Hunt, A. J. and A. W. Black (1996). "Unit selection in a concatenative speech synthesis system using a large speech database". In: *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*. Vol. 1, 373–376 vol. 1. DOI: 10.1109/ICASSP.1996.541110.

Iannis Xennakis (1963). *Formalized Music: Thought and Mathematics in Composition*.

Jehan, Tristan (2005). "Creating Music by Listening". PhD thesis. MIT.

Jouni, P. and K. Anssi (2010). "Drum Sound Detection in Polyphonic Music with Hidden Markov Models". In: *EURASIP Journal on Audio, Speech, and Music Processing* 2009. URL: http://www.hindawi.com/journals/asmp/2009/497292.

Mccormack, Jon (1996). "Grammar Based Music Composition". In: *Complexity International*.

Miron, Marius, Matthew E.P. Davies, and Fabien Gouyon (2013). "An open-source drum transcription system for Pure Data and Max MSP". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* January 2016, pp. 221–225. ISSN: 15206149. DOI: 10.1109/ICASSP.2013.6637641.

Morley, Iain (2003). "The Evolutionary Origins and Archaeology of Music". In: *Cambridge University* 2003.October 2003, p. 265. ISSN: 1749-9194.

Nierhaus, Gerhard (2009). *Algorithmic Composition: Paradigms of Automated Music Generation*.

Ó Nuanáin, Cartach (2017). "Connecting Time and Timbre : Computational Methods for Generative Rhythmic Loops in Symbolic and Signal Domains". PhD thesis. UPF.

Pachet, Franois (2003). "The Continuator: Musical Interaction With Style". In: *Journal of New Music Research* 32.3, pp. 333–341. ISSN: 0929-8215. DOI: 10.1076/jnmr.32.3.333.16861. URL: http://www.tandfonline.com/doi/abs/10.1076/jnmr.32.3.333.16861.

Pampalk, Elias, Perfecto Herrera, and Masataka Goto (2008). "Computational models of similarity for drum samples". In: *IEEE Transactions on Audio, Speech and Language Processing* 16.2, pp. 408–423. ISSN: 15587916. DOI: 10.1109/TASL.2007.910783.

Papadopoulos, George and Geraint Wiggins (1999). "A Genetic Algorithm for the Generation of Jazz Melodies". In: June 2000.

Ramires, António (2017). "Automatic transcription of vocalized percussion". Master Thesis. Universidade do Porto. URL: https://hdl.handle.net/10216/105309.

Rourke, Michelle O (2014). "The Ontology of Generative Music Listening". In: November.

Sandred, Örjan, Mikael Laurson, and Mika Kuuskankare (2009). "Revisiting the Illiac Suite—a rule-based approach to stochastic processes". In: *Sonic Ideas/Ideas Sonicas*, pp. 1–8. URL: http://www.sandred.com/texts/Revisiting{_}the{_}Illiac{_}Suite.pdf.

Schedl, Markus, Emilia Gómez, and Julián Urbano (2014). *Music Information Retrieval: Recent Developments and Applications*. Vol. 8. 4-5, pp. 263–418. ISBN: 9781601988072. DOI: 10.1561/1500000045. URL: http://www.nowpublishers.com/articles/foundations-and-trends-in-information-retrieval/INR-045.

Schwarz, Diemo (2000). "A System for Data-Driven Concatenative Sound Synthesis". In: *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00)* September, pp. 97–102. ISSN: 0929-8215. DOI: 10.1080/09298210600696691.

Schwarz, Diemo et al. (2006). "Real-Time Corpus-Based Concatenative Synthesis with CataRT". In: *9th International Conference on Digital Audio Effects (DAFx)*. cote interne IRCAM: Schwarz06c. Montreal, Canada, pp. 279–282. URL: https://hal.archives-ouvertes.fr/hal-01161358.

Toussaint, Godfried T et al. (2005). "The Euclidean algorithm generates traditional musical rhythms". In: *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, pp. 47–56.

Vinet, Hugues, Perfecto Herrera, and François Pachet (2002). "The CUIDADO Project". In: *3rd International Society for Music Information Retrieval (ISMIR) Conference*, pp. 197–203.

Wu, Chih Wei et al. (2018). "A Review of Automatic Drum Transcription". In: *IEEE/ACM Transactions on Audio Speech and Language Processing* 26.9, pp. 1457–1483. ISSN: 23299290. DOI: 10.1109/TASLP.2018.2830113.

Xenakis, Iannis (1992). *Formalized music : thought and mathematics in composition / Iannis Xenakis*. English. Rev. ed. Pendragon Press Stuyvesant, NY, xiv, 387 p. : ISBN: 0945193246 1576470792.

Yoshii, Kazuyoshi, Masataka Goto, and Hiroshi Okuno (2014). "Drum Sound Recognition for Polyphonic Audio Signals by Adaptation and Matching of Spectrogram Templates With Harmonic Structure Suppression". In: *Journal of Power Sources* 256.1, pp. 470–478. ISSN: 03787753. DOI: 10.1016/j.jpowsour.2013.12.100.

Zils, A. and F Pachet (2001). "Musical Mosaicing". In: *Proceedings of the COST G-6 Conference on Digital Audio Effects (DaFx-01)*, pp. 39–44.

Appendix A

Max/MSP Patchers

A.1 Chucker Arguments

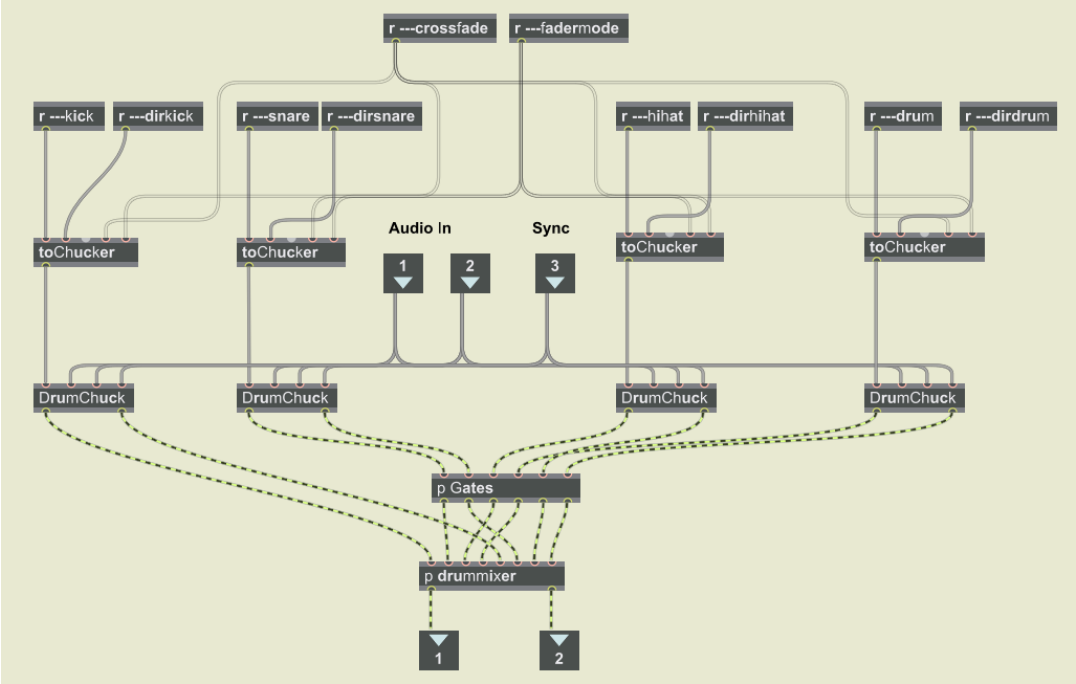


FIGURE A.1: ***DrumChuckers:*** Data flow from each *chucker~* arguments to audio outputs

A.2 Analysis

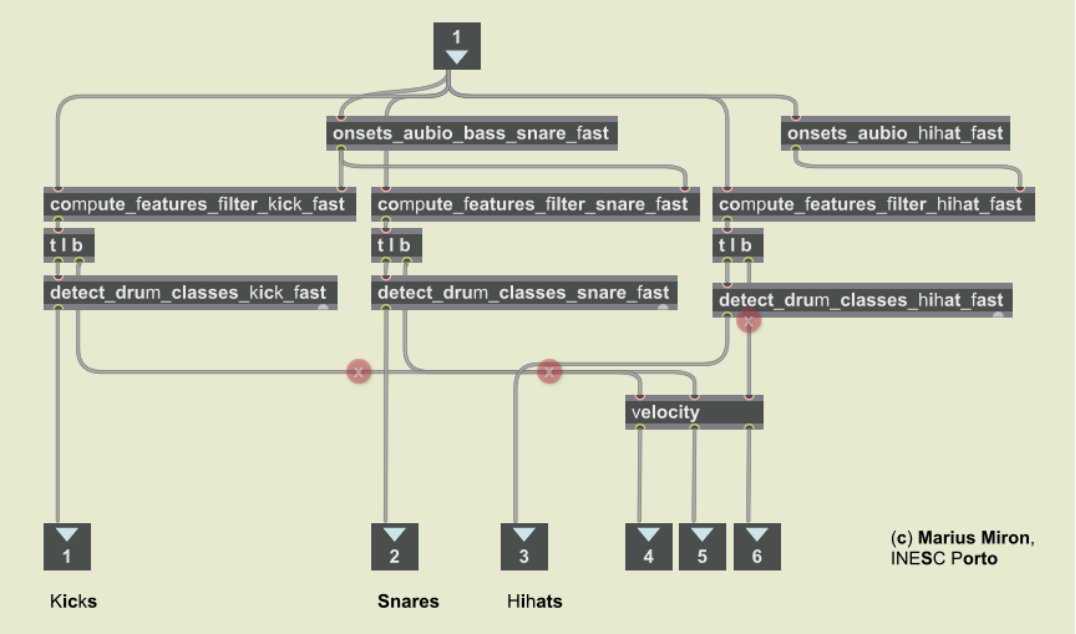


FIGURE A.2: ***LDT kNN:*** Conversion of audio signal into individual bangs associated with instrument classes. The system is described in (Miron, Davies, and Gouyon, 2013).

A.3 Synthesis

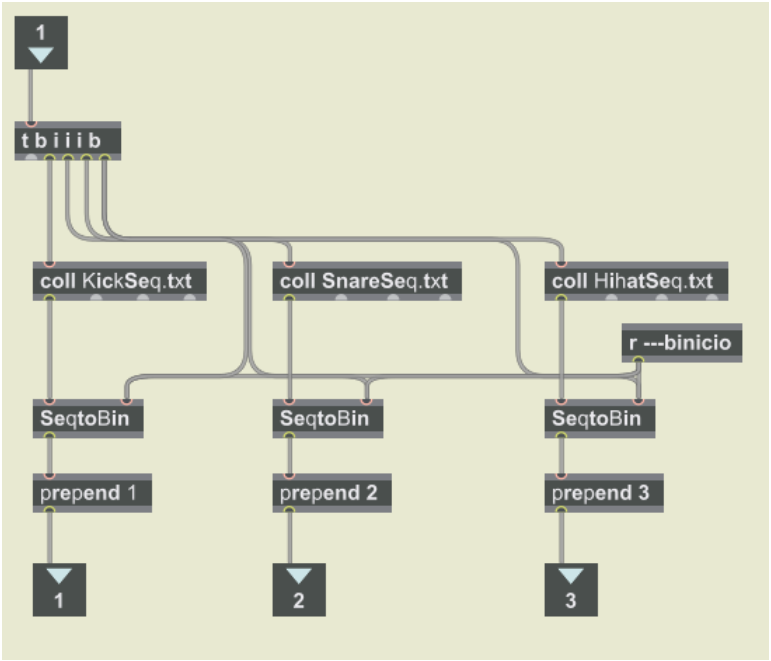


FIGURE A.6: ***loadfile:*** Load MIDI files into individual messages

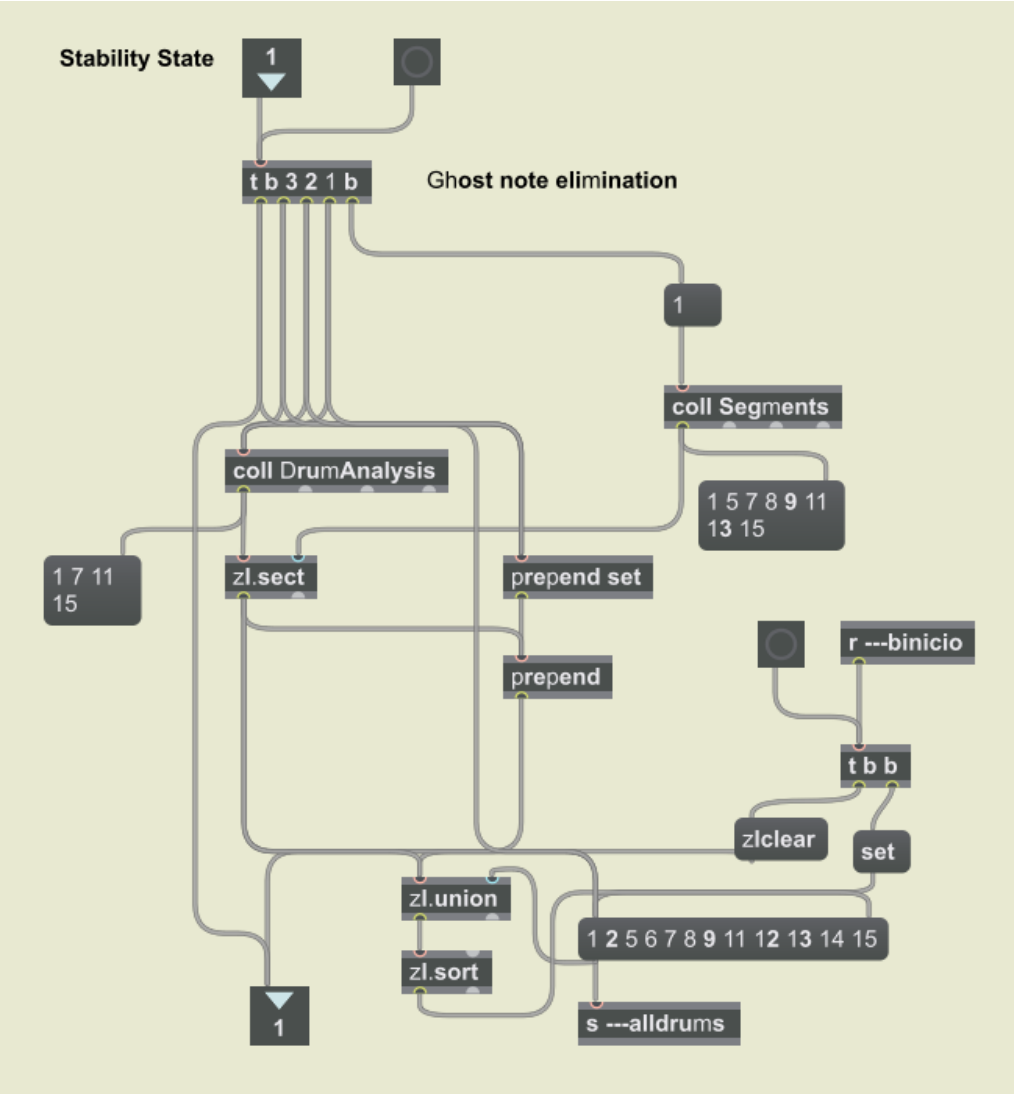


FIGURE A.3: ***FilterOutGhost:*** Elimination of ghost events

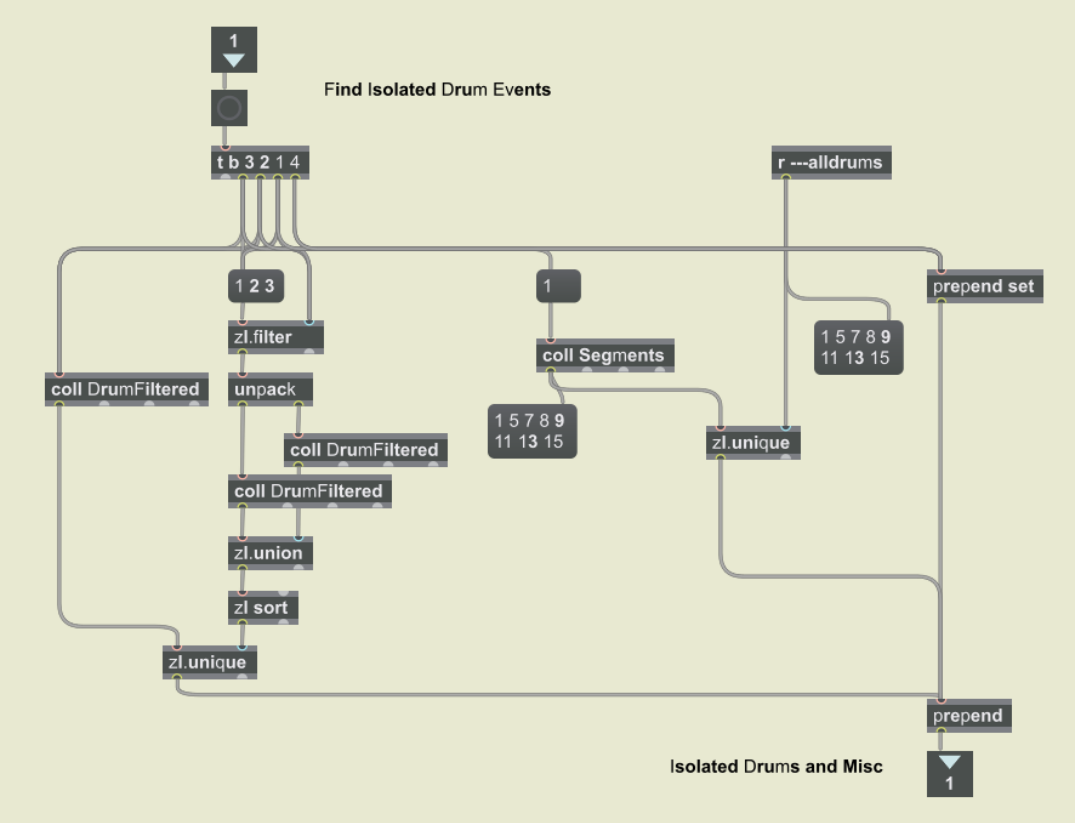


FIGURE A.4: ***FindIsolated:*** Test which segments have a single instrument playing

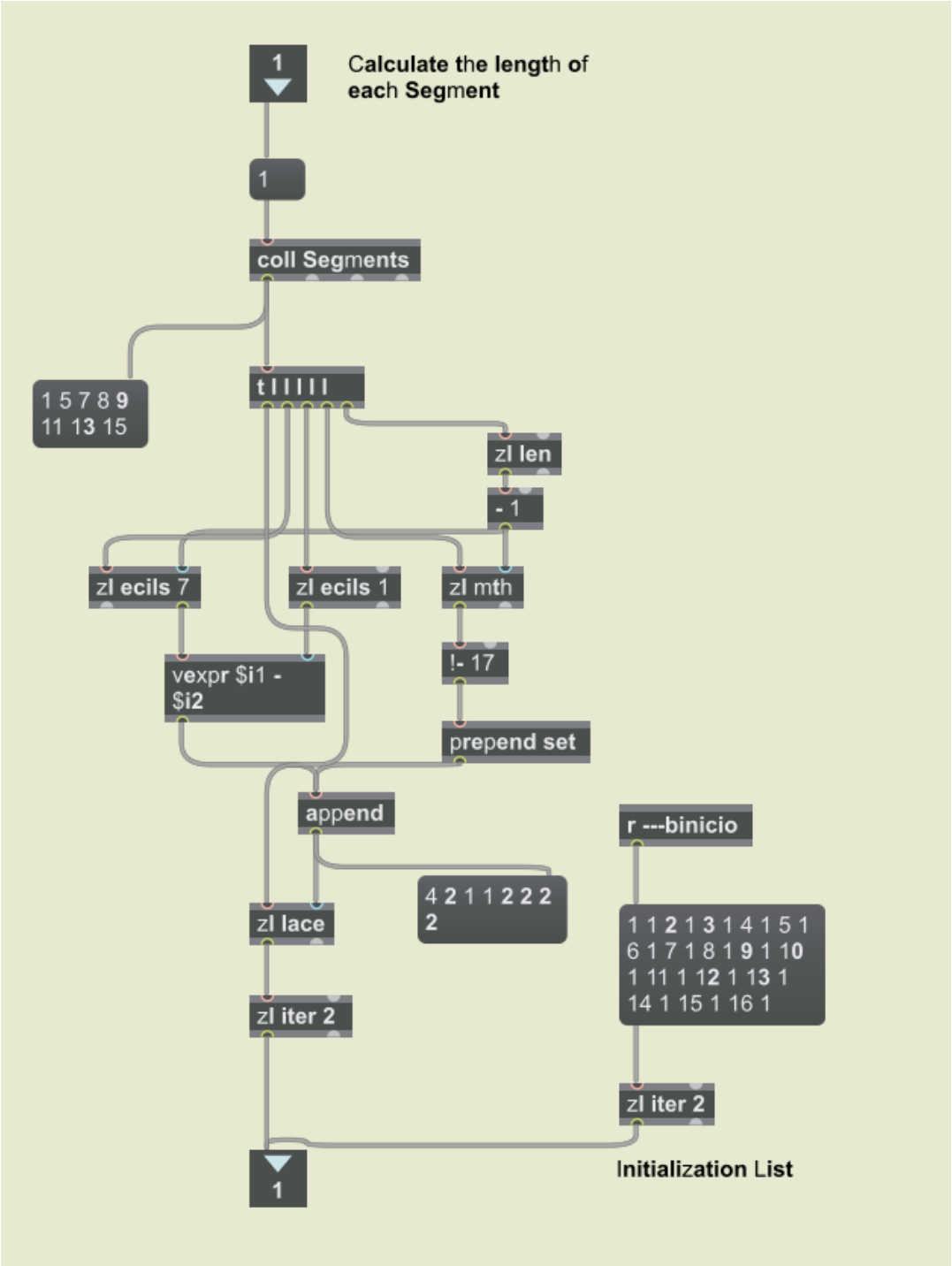


FIGURE A.5: ***findDrumLength:*** Determine the length of each sample

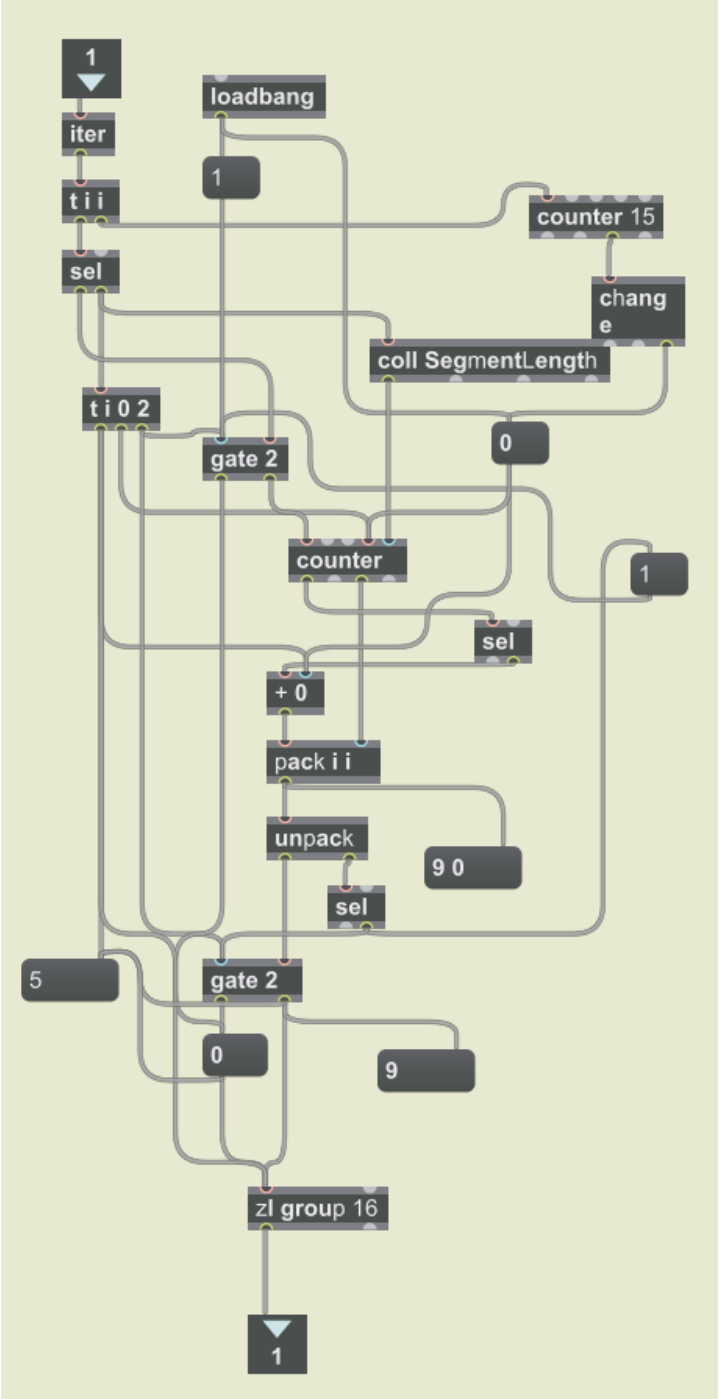


FIGURE A.7: ***LengthCorrection:*** Correct the sample length in the final sequence

Appendix B

Sequence Database Creation

B.1 Matlab Scripts

B.1.1 *main.m*

```
% READING MIDI FILES
%
% Starting Folder: C:\Users\Francisco Monteiro\Documents\TESE
% Set MIDI Files Folder

folder = '70 s Drummer MIDI Files\Tight Kit\01 Funk\01 Midtime 107BPM';

% Create Array of file names

ReadFiles = dir(folder);
folder_names=struct2table(ReadFiles);
file_list = strcat(folder, '\', ...
    string(table2array(folder_names(3:end,1))));


%% MIDI MAPPING // Define Global Variables
%
% MIDI Notes associated to a each instrument class
% Current Midi mapping set to: Native Instruments 70's Drummer ...
    TIGHT KIT
%

global Kick_Array Snare_Array Hihat_Array Misc_Array txtKick ...
    txtSnare txtHihat txtMisc
Kick_Array  = [36 60];
Snare_Array = [81 33 88 84 86 40 39 37 62 63 64 61 101];
Hihat_Array = [87 65 86 92 90 42 97 94 68 44 80 76 77 78 79 80 46];
Misc_Array  = (1:200);
Misc_Array  = setdiff(Misc_Array, [Kick_Array Snare_Array ...
    Hihat_Array]);

% Define output text files

txtKick  =  'C:\Users\Francisco Monteiro\Documents\Ableton\User ...
    Library\Presets\Audio Effects\Max Audio ...
    Effect\Tese\EXP1\KickSeq.txt';
txtSnare =  'C:\Users\Francisco Monteiro\Documents\Ableton\User ...
    Library\Presets\Audio Effects\Max Audio ...
    Effect\Tese\EXP1\SnareSeq.txt';
txtHihat =  'C:\Users\Francisco Monteiro\Documents\Ableton\User ...
    Library\Presets\Audio Effects\Max Audio ...
    Effect\Tese\EXP1\HihatSeq.txt';
```

```
txtMisc  =  'C:\Users\Francisco Monteiro\Documents\Ableton\User ...
    Library\Presets\Audio Effects\Max Audio ...
    Effect\Tese\EXP1\MiscSeq.txt';


%% Loop through the MIDI Files and write them to individual text files

numL = LineCount(txtKick);
for i=1:length(file_list)
    ExtractDrum(file_list(i), numL);
    numL=numL+1;
end

%% Clear

clear all;
```

B.1.2 *ExtractDrum.m*

```
function ExtractDrum( FileName, NumL )

% Extracting MIDI information from a .mid file and convert it to ...
    text format

global Kick_Array Snare_Array Hihat_Array Misc_Array txtKick ...
    txtSnare txtHihat txtMisc

midi = readmidi( FileName );            % Read MIDI information
Notes = midiInfo( midi , 0 );           % Create an array of MIDI notes
Notes(:,5) = round(Notes(:,5) * 8);     % Quantization
Notes16 = Notes( Notes(:, 5) < 16, :);  % Extract only 1st Bar

Kick_Midi  = Notes( ismember(Notes16(:,3), Kick_Array ), :); % Find ...
    the Kicks of the Midi File
Snare_Midi = Notes( ismember(Notes16(:,3), Snare_Array), :); % Find ...
    the Snares of the Midi File
Hihat_Midi = Notes( ismember(Notes16(:,3), Hihat_Array), :); % Find ...
    the Hihats of the Midi File
Misc_Midi  = Notes( ismember(Notes16(:,3), Misc_Array ), :); % Find ...
    the Miscellanious of the Midi File

% Correct output in case the file doesn't include one of the classes
if(¬isempty(Kick_Midi)) Kick_Bin = Kick_Midi(:,5) +1;    else ...
    Kick_Bin = 0;   end
if(¬isempty(Snare_Midi)) Snare_Bin =  Snare_Midi(:,5)+1; else ...
    Snare_Bin = 0; end
if(¬isempty(Hihat_Midi)) Hihat_Bin = Hihat_Midi(:,5)+1;  else ...
    Hihat_Bin = 0; end
if(¬isempty(Misc_Midi)) Misc_Bin = Misc_Midi(:,5)+1;     else ...
    Misc_Bin = 0;   end

%Write a Line in the respective output text file
WriteLine(txtKick, Kick_Bin, NumL);
WriteLine(txtSnare, Snare_Bin, NumL);
WriteLine(txtHihat, Hihat_Bin, NumL);
WriteLine(txtMisc, Misc_Bin, NumL);

end
```

B.1.3 *WriteLine.m*

```
function file = WriteLine (filename, seq, NumL)

% Write a line in text file

fileid=fopen(filename,'a');
fprintf(fileid, '%i, ', NumL);
fprintf(fileid, '%1d ',  seq);
fprintf(fileid, ';\n');
fclose(fileid);

end
```

B.1.4 *LineCount.m*

```
function numLines = LineCount(filename)

% Count number of lines in a text file

if(¬isfile(filename)) numLines=1;
else
fid = fopen(filename, 'rb');
% Get file size.
fseek(fid, 0, 'eof');
fileSize = ftell(fid);
frewind(fid);
% Read the whole file.
data = fread(fid, fileSize, 'uint8');
% Count number of line-feeds and increase by one.
numLines = sum(data == 10) + 1;
fclose(fid);

end
```

B.2 Drum Articulation

TABLE B.1: Drum Articulation Example. This one belongs to Native Instrument's *Abbey Road 70's Drummer* KOMPLETE instrument. Source: Manufacturer's English Manual available in: `https://www.native-instruments.com/en/products/komplete/drums/abbey-road-70s-drummer/`

Drum	Articulation	Default Key / MIDI Number
Kick	Dampened	C1 / 36
	Half Open	A#4 / 82
	Open	C3 / 60
Snare 1 & 2	Center Left Hand	A4 / 81
	Center Right Hand	B4 / 83
	Center Right/Left Alternating*	D1 / 38
	Halfway Left Hand	C5 / 84
	Halfway Right Hand	D5 / 86
	Halfway Right/Left Alternating *	E1 / 40
	Rimshot	D#1 / 39
	Sidestick	C#1 / 37
	Flam	D3 / 62
	Roll	D#3 / 63
	Wires Off	E3 / 64
	Rim Only	C#3 / 61