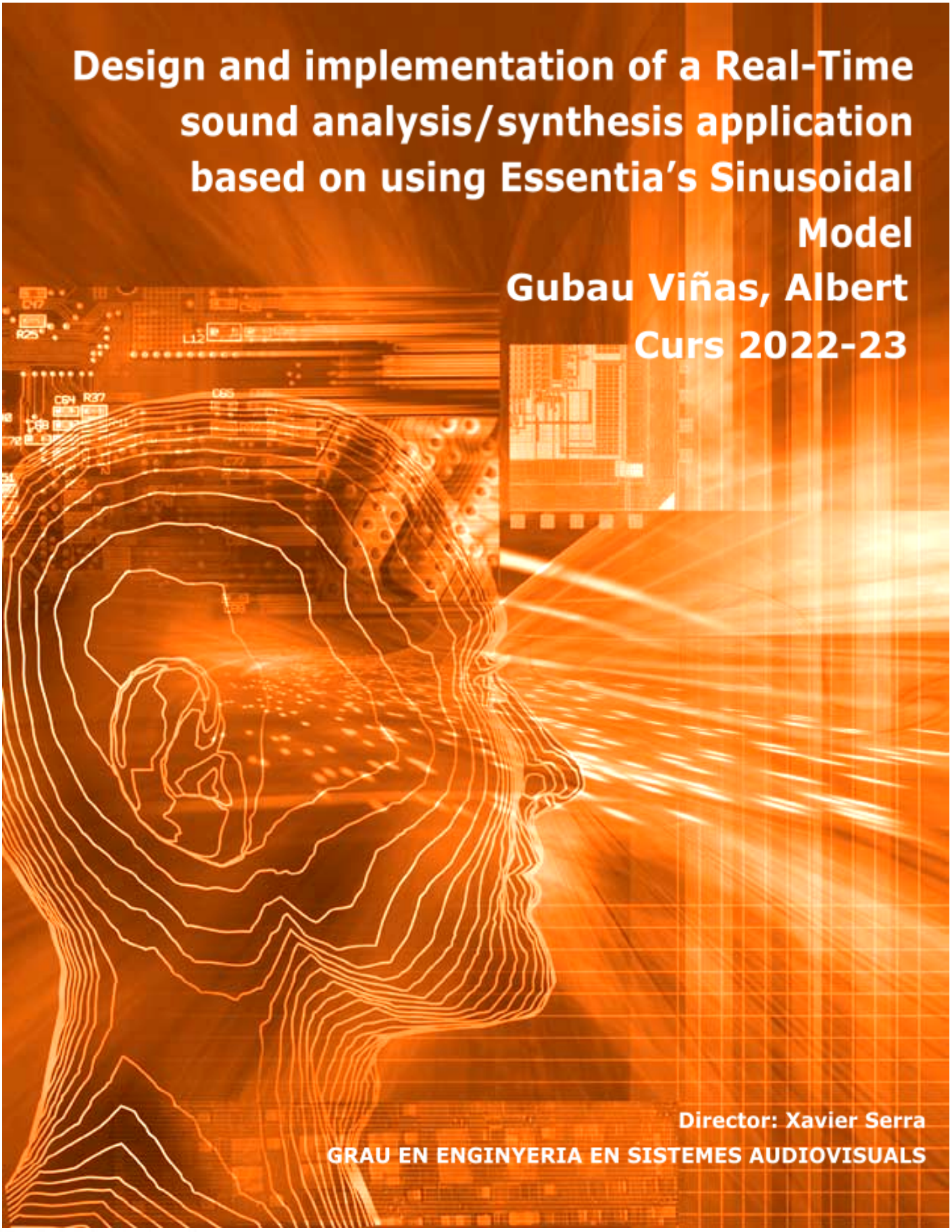
Design and implementation of a Real-Time sound analysis/synthesis application based on using Essentia's Sinusoidal Model

Gubau Viñas, Albert

Curs 2022-23



Director: Xavier Serra

GRAU EN ENGINYERIA EN SISTEMES AUDIOVISUALS

upf. Universitat Pompeu Fabra Barcelona
Escola d'Enginyeria

Treball de Fi de Grau

Design and implementation of a Real-Time sound analysis/synthesis application based on using Essentia's Sinusoidal Model

Albert Gubau Viñas

FINAL DEGREE THESIS – Audiovisual Systems Engineering

Publication date
June 16, 2023
Thesis Supervisor
Dr. Xavier Serra
Department
Information and Communication Technologies

Universitat Pompeu Fabra Barcelona
Escola d'Enginyeria

Per a la meva àvia Natàlia

Acknowledgements

I would like to express my gratitude to my thesis supervisor Dr. Xavier Serra, for the effort that has made helping me develop this idea, and also to the Essentia team that have assisted me in the pursuit of the software development goals.

I also want to thank all the teachers that have been instructing me for the last four years at Universitat Pompeu Fabra and all the YouTube creators that allow engineers to learn, grow and discover new concepts through funny and helpful audiovisual masterpieces.

And last but not least, I would not have been able to get to this point in my life without the support of my family, so, I will always be grateful to them for that.

Abstract

We may not have noticed, but sound processing is all around us, from being used to bring life to fictional characters by changing their voice to becoming a helpful tool for many artists around the globe. The aim of this project is based on the development of a simple sound analysis/synthesis application that can help educators and students in order to work with the *Sinusoidal Model*. I have developed a software that allows end users to use two different interfaces, one that permits analysis/synthesis of short audios by selecting a spectrogram region of interest, and another one that grants real-time sinusoidal synthesis of sound by enabling the microphone. The project basis leans on the usage of the *Sinusoidal Model* processing algorithms, which enables us to process the data coming from audio files and real-time sound recordings in an effective way. The software is coded with Python and uses the Music Technology Group (MTG) *Essentia* library in order to apply the different processing techniques required to accomplish the desired results and PyQt to get a satisfactory interface design.

Resum

Potser no ens n'haguem adonat, però el processament del so ens envolta, des de ser utilitzat per donar vida a personatges de ficció canviant la seva veu fins a convertir-se en una eina útil per a molts artistes d'arreu del món. L'objectiu d'aquest projecte es basa en el desenvolupament d'una aplicació senzilla d'anàlisi/síntesi de so que pugui ajudar a educadors i estudiants a treballar amb el *Model Sinusoïdal*. He desenvolupat un programari que permet als usuaris finals utilitzar dues interfícies, una que permet l'anàlisi i síntesi d'àudios curts seleccionant una regió d'interès a l'espectrograma d'aquests i una altra que permet la síntesi sinusoïdal en temps real de so capturat del micròfon. La base del projecte es basa en l'ús dels algoritmes de processament del *Model Sinusoïdal*, que ens permeten processar les dades procedents d'arxius d'àudio i enregistraments de so en temps real de manera efectiva. El programari està codificat amb Python i utilitza la biblioteca d'*Essentia* del MTG per aplicar les diferents tècniques de processament necessàries per aconseguir els resultats desitjats, combinades amb PyQt per tal d'obtenir un disseny d'interfície satisfactori.

Resumen

Es posible que no lo hayamos notado, pero el procesamiento del sonido nos rodea, desde ser utilizado para dar vida a personajes ficticios cambiando su voz hasta convertirse en una herramienta útil para muchos artistas en el mundo. El objetivo de este proyecto se basa en el desarrollo de una aplicación sencilla de análisis/síntesis de sonido que pueda ayudar a educadores y estudiantes a trabajar con el *Modelo Sinusoidal*. He desarrollado un software que permite a los usuarios finales utilizar dos interfaces, una que permite el análisis y la síntesis de audios cortos seleccionando una región de interés en sus espectrogramas y otra que otorga síntesis sinusoidal en tiempo real de sonido capturado del micrófono. La base del proyecto radica en el uso de los algoritmos de procesamiento del *Modelo Sinusoidal*, que nos permiten procesar los datos provenientes de archivos de audio y grabaciones de sonido en tiempo real. El software está codificado con Python y utiliza la biblioteca de *Essentia* del MTG para aplicar las diferentes técnicas de procesamiento requeridas para lograr los resultados deseados, combinados con PyQt para obtener un diseño de interfaz satisfactorio.

Contents

List of Figures..... xi

Chapter 1: Introduction.....1

1.1 Motivation..... 1
1.2 Planning.....1
1.3 Summary of contributions..... 2

Chapter 2: State of the Art..... 3

2.1 Signal Processing.....3
2.1.1 Spectral Analysis and Sinusoidal Model.....3

2.2 Software Tools.....5
2.2.1 Essentia.....6
2.2.2 PyQtGraph.....7
2.2.3 Real-Time processing with Python.....7

2.3 Software applications..... 8

Chapter 3: Materials and Methods..... 11

3.1 Technologies used.....11
3.1.1 Application interface (Front-End)..... 11
3.1.2 Application logic (Back-End).....18

3.2 Designed algorithms.....20
3.2.1 Sound analysis.....20
3.2.1.1 From file (non-real-time).....22
3.2.1.2 From microphone (real-time)..... 22
3.2.2 Sound transformation..... 24
3.2.2.1 Filtering..... 24
3.2.2.2 Pitch Shifting.....25
3.2.3 Sound Synthesis.....25

3.3 Software development.....27
3.3.1 Application software management.....27
3.3.2 Software tools used.....28

3.4 Software requirements....28
3.5 Software architecture....30

Chapter 4: Results....31

4.1 Final application....31
4.1.1 Spectrogram Sound Synthesizer....32
4.1.2 Real-Time Sinusoidal Transformation....35

4.2 Software evaluation....37

Chapter 5: Conclusions....39

5.1 Obtained results from first idea....39
5.2 Project drawbacks....39
5.3 Possible improvements....40
5.4 Next steps....40

Bibliography....41

Appendix A: Color Scheme of the Application....43

Appendix B: Results of the Software Evaluation....44

List of Figures

Figure 1.1: Gantt diagram of the project planning..... 2

Figure 2.1: DFT equation..... 3

Figure 2.2: sms-tools DFT for an oboe-A4 sound using a blackman window with.....4

Figure 2.3: Sinusoidal Model expression..... 4

Figure 2.4: Spectrogram of a bendir sound together with the sinusoidal tracks extracted from the Sinusoidal Model..... 5

Figure 2.5: Essentia documentation on the SineModelAnal method of their library..... 6

Figure 2.6: Audacity “Change Pitch” interface..... 8

Figure 2.7: Sonic Visualiser Spectrogram Analyzer..... 9

Figure 2.8: Kilohearts Pitch Shifter interface..... 10

Figure 3.1: Sketch of the main window of the application..... 12

Figure 3.2: Main window of the application in QtDesigner with default styling..... 12

Figure 3.3: Stylesheet of the application main window..... 13

Figure 3.4: Same button while hovering (right) and not hovering (left)..... 14

Figure 3.5: Widgets added to the main window and their names..... 14

Figure 3.6: Sketch of the Spectrogram Sound Synthesizer window..... 15

Figure 3.7: Spectrogram plot together with the Colorbar and the Region of Interest selector..... 16

Figure 3.8: Sketch of the Sinusoidal Model Transformation window..... 17

Figure 3.9: File organization of the application..... 18

Figure 3.10: Essentia SineModelAnal function parameters..... 20

Figure 3.11: Computation of auxiliary frames to apply Essentia computations using a 2048 samples FFT Size and a 512 samples Hop Size..... 23

Figure 3.12: Filtering applied in the Spectrogram Sound Synthesizer..... 24

Figure 3.13: Semitones pitch shifting.....25

Figure 3.14: Semitones pitch shifting applied in the software..... 25

Figure 3.15: Software architecture of the Nataly application.....30

Figure 4.1: Nataly application with the dark theme..... 31

Figure 4.2: Nataly application with an audio loaded in the Spectrogram Sound Synthesizer interface with the dark theme.....32

Figure 4.3: Input file scheme of the Spectrogram Sound Synthesizer interface..... 32

Figure 4.4: Slider of the Spectrogram Sound Synthesizer interface.....33

Figure 4.5: Spectrogram plot of the Spectrogram Sound Synthesizer interface..... 33

Figure 4.6: Colorbar widget of the Spectrogram Sound Synthesizer interface..... 34

Figure 4.7: Right-side widgets of the Spectrogram Sound Synthesizer interface..... 34

Figure 4.8: Nataly application with audio being recorded in the Real-Time Sinusoidal Transformation interface with the dark theme.....35

Figure 4.9: Waveform and spectrum plots in the Real-Time Sinusoidal Transformation interface..... 35

Figure 4.10: Slider of the Real-Time Sinusoidal Transformation interface..... 36

Figure 4.11: Recording label in the Real-Time Sinusoidal Transformation interface.....36

Figure A: Color Scheme of the Application with Dark Theme.....43

Figure B: Color Scheme of the Application with Light Theme..... 43

Figure C: Change theme button in the main window.....43

Chapter 1: Introduction

The aim of this chapter is to introduce a little bit the motivation of the project, the idea behind it and the purpose of developing it, as well as the planning that I scheduled to accomplish the desired results and a little abstract of the final outcome.

1.1 Motivation

The main purpose of this project is to build a sound processing application which can help students and educators in order to enhance their sound understanding from the two points of view. The idea is to work with the Music Technology Group (MTG) Essentia library in order to develop an application that can analyze spectrograms and synthesize audio in a real-time basis, allowing the two previously presented users to use it as a tool to improve their knowledge.

The motivation behind the development of this project comes from two main pillars. As one of the main purposes for me while developing the final thesis was to learn and one aspect that I have always wanted to learn was the software interface development, one of the main pillars was the restlessness to learn this kind of skills. The other main pillar came from the fact that I have studied Audiovisual Systems Engineering at Universitat Pompeu Fabra and our university is popular for having a reference research group in terms of audio processing as it is the MTG so, I thought that I could use my career knowledge about audio processing in order to develop something interesting and helpful with a great design. Therefore, combining these two main reasons and the fact that I have always wanted to develop some kind of tool for students and educators to upgrade their learning/teaching journey, I managed to bring this idea to life.

1.2 Planning

The project plan was structured following different steps. Firstly, I dedicated a couple of weeks to define the project purpose and idea, and then I wanted to ensure that I could use the Essentia library sound processing functions in a Python-based code and that all the tools needed to develop an application were available to do so (research stage). In order to do that, I planned to spend some weeks working on applying these algorithms in a known audio processing project such as sms-tools[1]. Following that methodology, I scheduled that at the moment at which I reached the point when I got the necessary skills to work with the library and that the functions could be used to develop a Python project, I would do research on Python-based Graphical User Interfaces (GUIs) and real-time applications. The plan for the research part was to focus some weeks in researching for example applications that tried to apply real-time based sound synthesis using Python in order to see the feasibility of doing so.

[1] https://www.upf.edu/web/mtg/sms-tools (accessed 4 May 2023)

Finally, the project plan was to develop and design the application within the time that I had left. All the tasks and time dedicated to them can be found in the following figure:

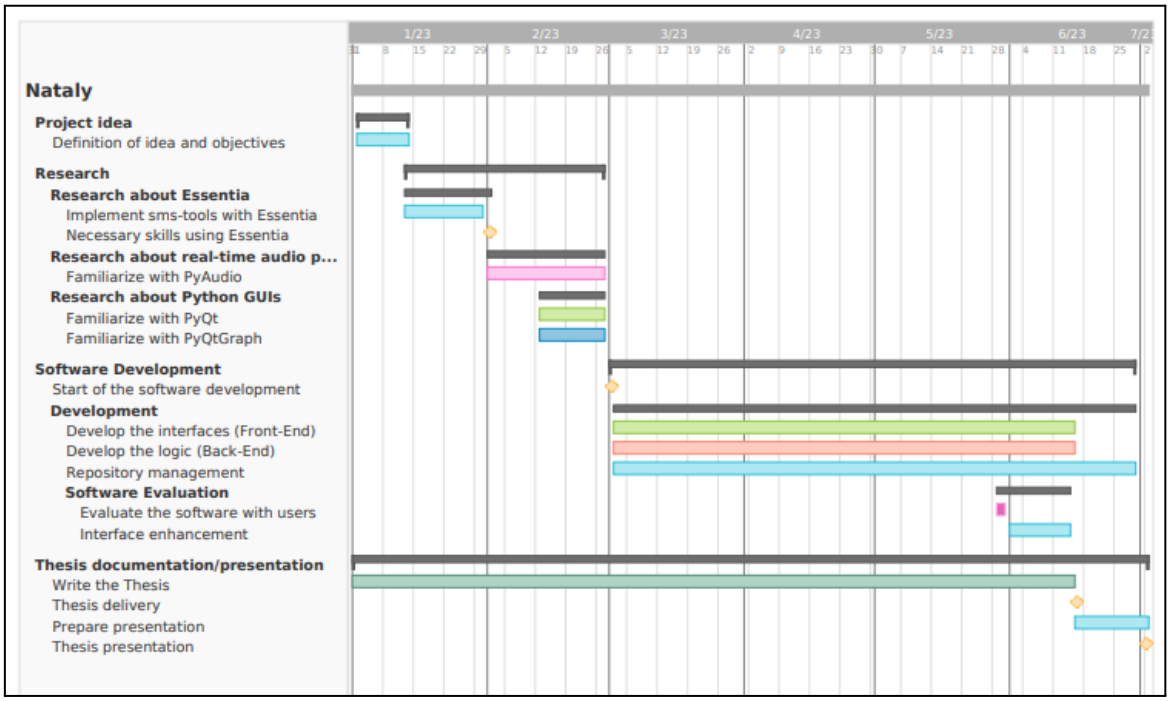


Figure 1.1: Gantt diagram of the project planning[2]

1.3 Summary of contributions

Taking into account the initial objectives, the results obtained for the software are more than satisfactory. I managed to develop an application that can help students to understand spectrograms and synthesize sound with them in a playful and interactive way. I have also managed to find a way to synthesize sound with the Sinusoidal Model on a real-time basis using Python combined with Essentia and PyQt.

Looking at the obtained results, we can also see that some of the initial objectives were not accomplished and therefore, they should be implemented in future versions of the project. As an example, the application is not multi-platform, it only works on Ubuntu based systems at the moment. Another aspect to consider is the fact that the application is not able to deal with large audios as it stores this information in arrays that occupy lots of memory and require a lot of disk usage while working with them. Thinking about the purpose of the application, I have also created a tool that actually can help students and teachers to enhance their journey while understanding audio processing, which was one of the main objectives of the project too, and considering the initial project planning, I have accomplished the objectives and dates scheduled to fulfill them. The software, a video demonstration and this thesis can be found in the GitHub repository[3].

[2] https://www.teamgantt.com
[3] https://github.com/albertgubau/Nataly

Chapter 2: State of the Art

The purpose of this chapter is to introduce the current context of the different mathematical models that we are going to use for the sound processing part of the software, as well as to analyze the current state of the art for the available applications in the market that allow users to apply sound processing algorithms in an efficient way.

2.1 Signal Processing

Signal processing can be described as a set of techniques applied to real world signals that allow us to operate through these types of information in a way that we can observe, analyze, transform and use them to accomplish different kinds of results. In this project we will focus our attention on a very particular set of signals which are audio signals, that can be transformed by using different mathematical models.

2.1.1 Spectral Analysis and Sinusoidal Model

One of the most used transformations that allows us to represent and analyze audio signals is the **Fast Fourier Transform (FFT)**[4] which is in fact an algorithm applied in order to optimize the **Discrete Fourier Transform (DFT)**. The Fourier Transform and most specifically, the DFT, is characterized by the particularity of allowing users to represent data using frequency representations, granting them the capacity to analyze the spectral representation of a signal by applying a mathematical expression.

The DFT expression is the following one [1] [2]:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi k \frac{n}{N}} \qquad k = 0, 1, ..., N-1$$

Figure 2.1: DFT equation

Where:

x[n] = input signal in the time domain (N samples)
X[k] = output signal in the frequency domain (N samples)
n = discrete time index
k = discrete frequency index

The expression can be described as a scalar product between a discrete signal x[n] and a collection of sinusoids, in fact, it can be seen as a projection that tells us how much of a sinusoid is in x[n].

[4] www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft (accessed 12 May 2023)

The frequency representation that we can obtain by using the above expression is called spectrum and it can have different shapes depending on the frequency content of the signal[5]. Here we can see an example of the DFT computation extracted from the sms-tools application software and using oboe sound as input:

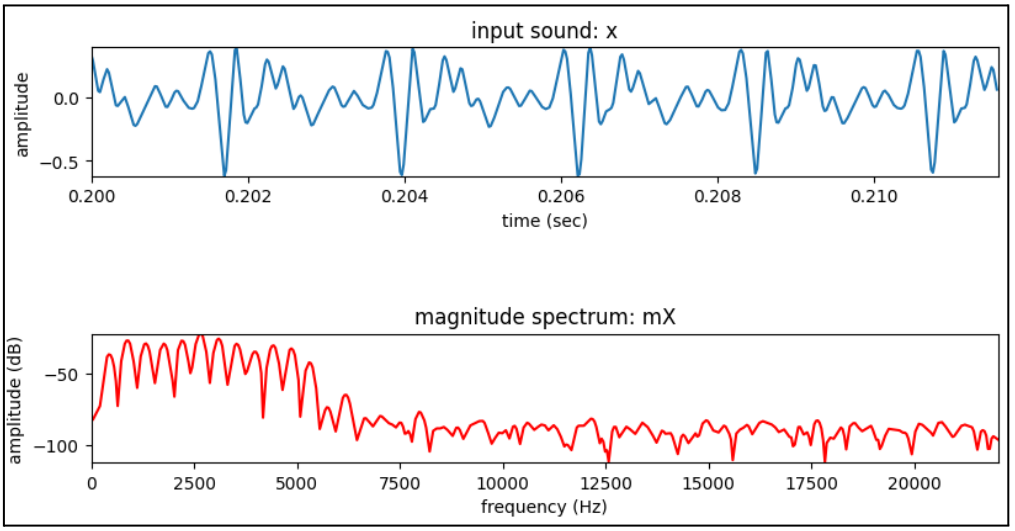


Figure 2.2: sms-tools DFT for an oboe-A4 sound using a blackman window with FFT size = 1024 and window size = 511

From it we can see that the spectrum allows us to see the frequency content of the signal in an intuitive way, allowing us to know the main frequency components of the data and permitting us to play with them to acquire different results and transformations using different models (spectral analysis/synthesis). For the purpose of this project, we are focusing our attention on one specific mathematical model that allows us to apply transformations to sound data by using the DFT explained basis, this model is called the Sinusoidal Model. The **Sinusoidal Model** is a higher level representation based on a mathematical model that grants us the capability to see the sound represented as a collection of sinusoids.

The expression that represents this model is the following one[6]:

$$y[n] = \sum_{r=1}^{R} = A_r[n] \cdot cos(2\pi f_r[n]n)$$

Figure 2.3: Sinusoidal Model expression

Where:

R = number of sine waves
$A_r[n]$ = instantaneous amplitude
$f_r[n]$ = instantaneous frequency

[5] www.coursera.org/learn/audio-signal-processing/lecture/EZRXC/dft-1 (accessed 12 May 2023)
[6] www.coursera.org/learn/audio-signal-processing/lecture/gjiP7/sinusoidal-model-1 (accessed 12 May 2023)

From the expression of the model we can see that it tries to typify the signal as a summatory of multiple sinusoids with time-varying amplitude and frequency. Then, the model defines the sinusoids as peaks in a magnitude spectrum which will be later identified with stable tracks with the notion of the spectrogram representation. The model allows us to extract the amplitude, frequency and phase data from each sinusoid present in our signal. A spectrogram representation allows us to represent a collection of spectrums taken from different frames and captured at different time instants of the sound in a way that we can identify the changes/tendencies of the signal frequency content through time[7]. An example of this type of representations combined with sinusoidal tracks obtained by applying the sinusoidal model is the following one:

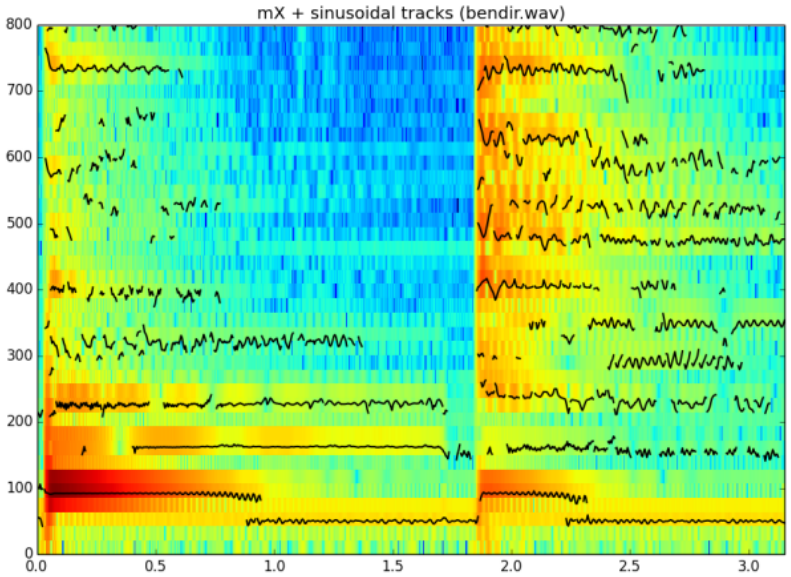


Figure 2.4: Spectrogram of a bendir sound together with the sinusoidal tracks extracted from the Sinusoidal Model

2.2 Software Tools

In order to bring this project to life we must ensure that we have the right processing tools to develop the main idea. As this application is based on sound processing, we need to use different programming libraries to ensure the right operation, visualization and computation time of the audio data. At the current software development context, and specifically taking into account the sound processing context, applications must ensure or at least try to apply their procedures on a real-time basis. Most of the currently available software in the market guarantee this type of computations and we must be concerned about the user desires at the development stage that we are living in. Taking into account the project to be developed, we must take into account the current software tools available to apply the procedures that we want to achieve. Some of the most important tools that we can find nowadays to develop sound processing/visualization python-based applications are the following ones.

[7] https://www.coursera.org/learn/audio-signal-processing/lecture/D4u8J/sinusoidal-model-2 (accessed 12 May 2023)

2.2.1 Essentia

Essentia is an open-source sound processing library developed by the Music Technology Group (MTG) at Universitat Pompeu Fabra in Barcelona. The library offers a variety of audio data models/functions to be applied that allow users to analyze, synthesize and transform sound together with many other applications[8] [3].

I have chosen this library for many reasons, firstly because it is a library developed at the university in which I have studied and I wanted to help in the research for new upgrades, uses and enhancements of the library. Secondly, I used Essentia as it is based on computations that are similar to the ones applied in the sms-tools application with which I worked in other projects and subjects of the career allowing me to be familiarized with its procedures. And last but not least, I have chosen Essentia because it has been developed to include Python as one of the programming languages that can be used with, allowing also to be applied on a real-time basis by making computations faster. This library allows us to apply a wide variety of algorithms and sound processing computations, but the ones that we will take into account are the Sinusoidal Model methods that include other different procedures such as the audio file loading, the FFT transformation, windowing, audio framing, and more.

All of these operations can be found within the Essentia library and are the ones that we are going to use. For example, as an overview of one concrete method of the functions that we are going to use, we can see the documentation of the sinusoidal model analysis method[9].

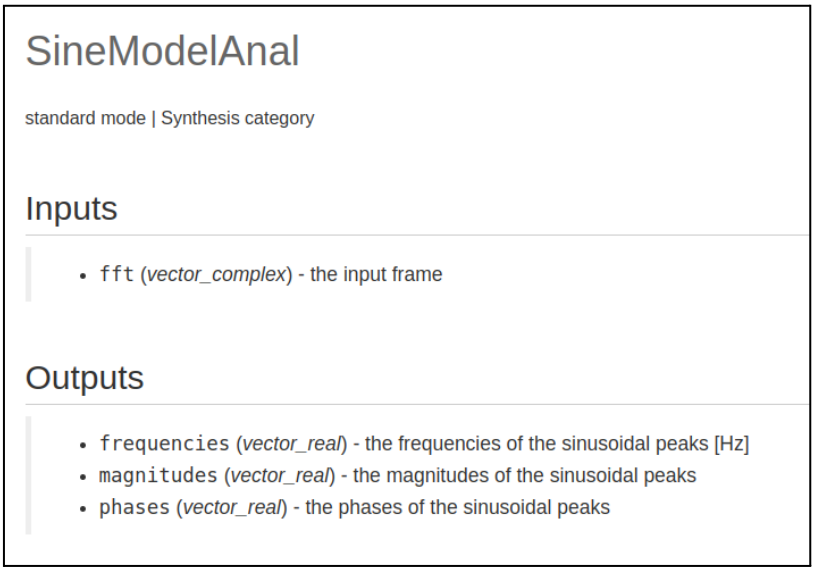


Figure 2.5: Essentia documentation on the SineModelAnal method of their library

[8] https://essentia.upf.edu
[9] https://essentia.upf.edu/reference/std_SineModelAnal.html (accessed 15 May 2023)

From the figure above, we can see the documentation style of the Essentia library for one specific function which is the SineModelAnal method. One of the key strengths of Essentia's Sinusoidal Model is that it has a high accuracy while tracking the sinusoidal components over time even if there is presence of noise/interference in the audio track. As we have said, we are not only going to use the Sinusoidal Model function of the library, as Essentia is designed to be modular and flexible, we can combine different functions to achieve the analysis/synthesis of an audio signal. The other functions that we will be mainly using will be the MonoLoader method (allows us to load audio tracks and convert them to arrays of data), the FFT method[10] (allows us to apply the Fast Fourier Transform to an frame of audio data), the Windowing method[11] (allows us to window a given audio frame) and more.

2.2.2 PyQtGraph

As we know, our project is aimed at building an efficient data visualization and interaction application, and for this, we are going to use the PyQtGraph library[12], which is a Python library that can be applied in order to build data visualizations and interactions in an efficient way. Nowadays, we have a clear reference library that allows users to build data plots which is Matplotlib. In comparison with the more widely known Matplotlib library, PyQtGraph computes plotting algorithms faster, allowing applications to display real-time changing plots with minimum delay, which is a feature that we want to guarantee in our software. Another advantage of using PyQtGraph is that this library also has some great features such as the Region of Interest (ROI) selector of graphs, which will enable us to build the spectrogram synthesizer of the application with great performance.

2.2.3 Real-Time processing with Python

The current context of real-time computations using Python depends on many different factors. First of all, we must take into account that at the current time at which this project is developed, Python is an interpreted language, and from this fact we can easily know that it can be slower than compiled languages such as C and C++[13]. We will see that, for example, Audacity is coded using this type of languages and taking into account also the Essentia library that we are going to use for the development of the project, we can notice that they are also based on those languages but can be used in many cases with Python. Nevertheless, from the Python starting version till now, we can see that the language has been improving computation time, although it is an interpreted language, and in some cases it can reach the computation speed needed to be applied in real-time applications.

---

[10] https://essentia.upf.edu/reference/std_FFT.html (accessed 15 May 2023)
[11] https://essentia.upf.edu/reference/std_Windowing.html (accessed 17 May 2023)
[12] https://www.pyqtgraph.org
[13] https://towardsdatascience.com/how-fast-is-c-compared-to-python-978f18f474c7 (accessed 21 May 2023)

Even if the language has been improving their code interpreting/computation speed, the developers can also use some tools to ensure that their Python-based projects increase their performance such as multi-threading[14], multi-processing, optimization of algorithms, framework using, and more[15]. To sum up, we can state that Python can be suitable for developing real-time applications but we need to take into consideration the context of the application, the framework (if needed) that we are going to use, the software architecture and the libraries that will be used in the project.

2.3 Software applications

Nowadays, there are plenty of applications that enable users in order to apply sound analysis, synthesis and transformations through interactive interfaces, including sound processing libraries that help the development of such softwares. For the purpose of this thesis, the applications that we are going to analyze are Audacity, Sonic Visualiser, and Kilohearts Pitch Shifter.

Audacity

It is a free open-source cross-platform software that allows users to record and manipulate multi-track audio streams of data[16]. The application is developed using C and C++ and has some interesting features that we want to take into consideration for the implementation of our project. We want to analyze how Audacity applies their pitch transformations to audio signals. In the software initial interface, we can quickly identify the Effects option at the upper menu. If we load any sound and then we proceed by clicking on the Effects option we can see that it appears an unfolded menu presenting us all the possible effects to be applied to the loaded signal. For the purpose of this project, we will focus on the implementation of the pitch shifting effect that can be found in the menu as “Change Pitch”. When we open this option, a modal appears:

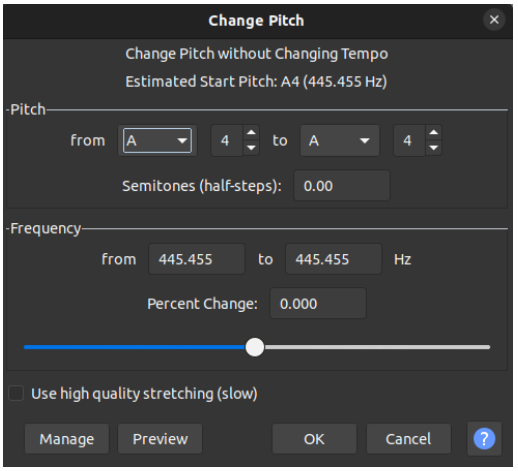


Figure 2.6: Audacity “Change Pitch” interface

[14] https://www.geeksforgeeks.org/multithreading-python-set-1 (accessed 22 May 2023)
[15] https://thechief.io/c/editorial/how-python-is-becoming-faster (accessed 22 May 2023)
[16] https://www.audacityteam.org

As we can see in the previous figure, the interface is based on two main parts, one of them refers to the Pitch and the other refers to the Frequency. Basically, the first one manages the tones of the audio and the second one manages the multiplication factor of the frequency content (in this case in percentage) so you can slide the selector to get high or low pitched audios depending on the sliding direction. You can also preview the resulting sound and change the steps of the pitch changer. This interface helps us to get an idea of what a frequency change involves and that we can manage it through sliders.

Sonic Visualiser

This software is also developed as a free, open source and cross platform project in which users can interact with sounds in a way that they can get a detailed visualization, annotation and analysis of audio data[17]. For the purpose of this thesis, we will analyze how this software grants the detailed visualization of audio signal spectrograms to take it as an example implementation of what we could apply in our software.If we open the software, we can easily load any audio file that we want to analyze and then add a spectrogram layer in which they allow us to see the spectrogram of the sound and change some of the analysis/synthesis parameters like the window size, the amount of overlap applied, the spectrogram colors, the range of the sound we are looking at, the frequency scale and the amount of bins. We can look at the interface that Sonic Visualiser presents to analyze spectrograms in the following figure:

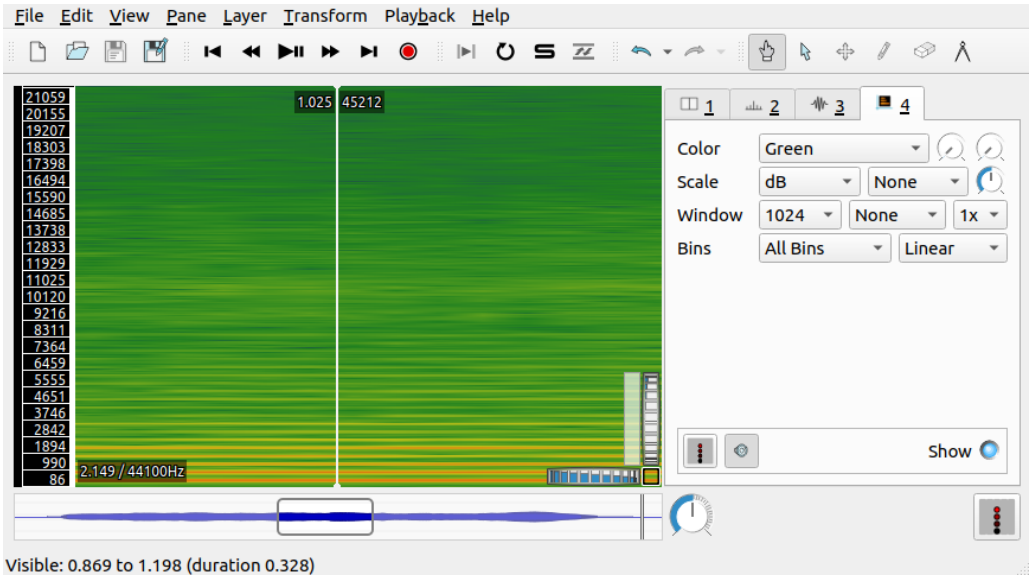


Figure 2.7: Sonic Visualiser Spectrogram Analyzer

As we can see, we have control over the visualization of the spectrogram which is what we will need to ensure in the final application, and we have also control over the analysis parameters that generate the spectrogram, which we can change and see the resulting plot in real time.

[17] https://www.sonicvisualiser.org

Kilohearts Pitch Shifter

The Kilohearts Pitch Shifter is a plugin that allows users to shift the pitch of audio signals while preserving their quality by using sophisticated algorithms. The plugin also grants users the possibility to apply this shifting by using semitones or cents (like Audacity) which can be very suitable for those who want to use the software for musical applications in which you may need to adjust the key of a track or create new harmonies[18]. Moreover, the Kilohearts software has more utilities such as time stretching and formant-shifting which can be applied for advanced modifications of our track.

Taking into consideration the strengths of the plugin, we can notice that the performance of the computations looking at the quality of the transients preservation is notable, which is an important aspect to be considered while applying these kinds of transformations to audio signals. We can take a look at the simplicity of the application interface in the following figure:

Figure 2.8: Kilohearts Pitch Shifter interface

From looking at the figure above, we can see the different parts of the software such as the pitch display at top, showing us the pitch adjustment of the sound. Moreover, we can see the Jitter, Grain Size and Mix knobs at the middle which are used to add randomness to the pitch, chop the audio into small snippets called grains and apply a dry/wet mix to the pitch shifting respectively. We can also notice that we have two options at the bottom which are the Correlate and Compensation selectors that allow us to get a clear effect on many inputs and also to compensate for any latency introduced by the plugin respectively.

As a conclusion, we can see that there is plenty of software available out there on the market but we will focus our attention on the ones that are applying the similar procedures that we want to implement in our project. That is why I have chosen these three applications to be analyzed. They fit many of the requirements that my application needs to implement and we can take ideas of how this is applied in those softwares.

[18] https://kilohearts.com/products/pitch_shifter (accessed 23 May 2023)

Chapter 3: Materials and Methods

The aim of this chapter is to introduce the main materials and methodologies applied to accomplish the objectives of this project. We will dive into the technologies used to develop the application (front-end, back-end and software management) and we can also find information of the designed algorithms to make the two interfaces work (the spectrogram sound synthesizer and the real-time sinusoidal transformation). Finally, we will take a look at the software development, software requirements and software architecture parts, in which we can find the methodologies applied to develop the software, the functional and non-functional requirements of the application and the file organization of the project.

3.1 Technologies used

For the technologies applied to develop the application, we can distinguish between different areas of the software development. Firstly, we will take a look at the application interface design (the front-end). Secondly, we will see how the logic of the application is implemented (the back-end). And finally, we will consider the main issues about software management.

3.1.1 Application interface (Front-End)

The application interface is one of the most important aspects of this project, together with the sound processing algorithms applied. We want to ensure that the application looks simple and satisfactory to the users that are going to use it. When talking about audio processing applications, we must take into consideration that the software must be intuitive and simple, taking into account the user perspective and giving them the facilities to apply the different procedures available in the application. The main objective of the project is to give the end users (mostly students and educators) an application that can help them in two stages: the learning process and the teaching process, so we must take that into consideration while designing the software.

The application interface has been developed using Qt, but more specifically, PyQt5[19] through the Qt Designer[20] software that enables users to design with Qt based interfaces through a drag & drop interface that is very simple to use. The usage of this library is pretty simple, you have different widgets available that you can use to design the interface of your application, there are buttons, labels, sliders, and more. You can download the program and start the design process of a window, using the widgets available in Qt and personalizing them. When we have the right interface designed, we can export it as a .ui file that is used later in the Back-End to load the window design [4] [5].

[19] https://www.qt.io/qt-for-python (accessed 23 May 2023)
[20] https://doc.qt.io/qt-6/qtdesigner-manual.html (accessed 25 May 2023)

For this project, I designed three different windows using this procedure: the application window (Main window), the Spectrogram Sound Synthesizer and the Real-Time Sinusoidal Model Transformation windows, therefore, we will have three .ui files (main.ui, sinusoidal_spec_synth.ui and rt_sine_transformation.ui).

Application window (main.ui)

This is the window that should take care of changing to the two other interfaces while presenting them in a given space, changing the theme (light or dark) and presenting the application logo as well as the Essentia Logo. As I believe that an image says more than a thousand words let's see the first sketch[21] of how the application should look like:

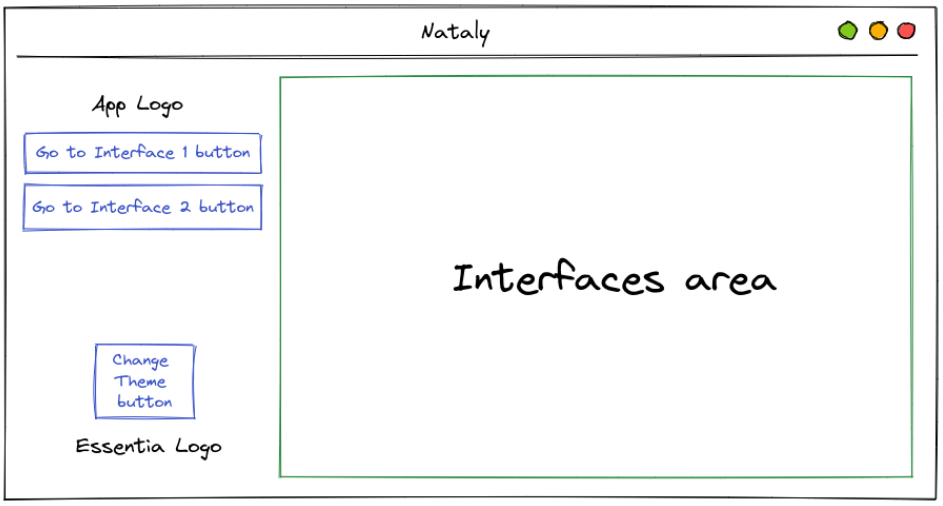


Figure 3.1: Sketch of the main window of the application

As we can see in the figure above, the main interface is composed of the shown parts that we can add using the Qt Designer Software:

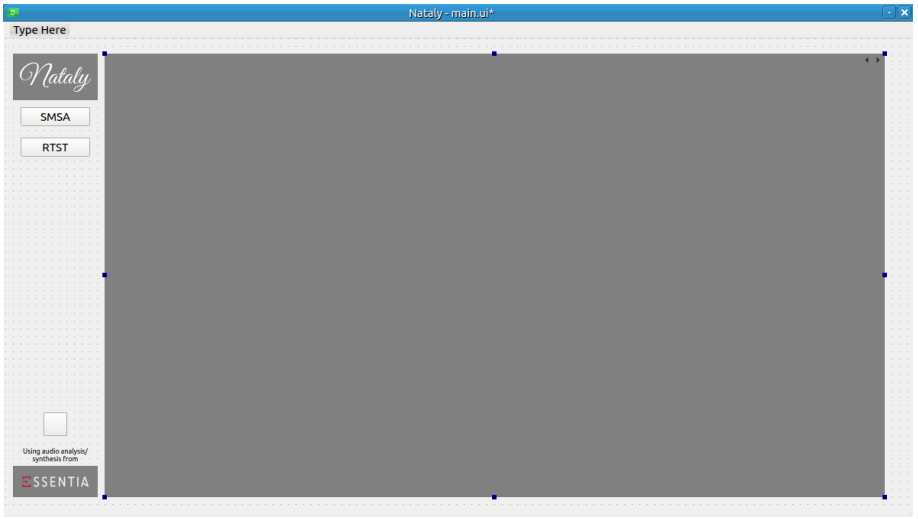


Figure 3.2: Main window of the application in QtDesigner with default styling

[21] https://excalidraw.com

One important aspect of the interface design is the appearance, we can leave the styles of the widgets as default but the application would look kind of simple and old, like in the previous image, so I decided to dive into the stylesheets of the widgets to know more about the design procedures that are applied to them.

It is an interesting fact to notice that the stylesheets applied to the widgets use the Cascading Style Sheets (CSS)[22] codes that are applied in today's web page designs, so let's take a look at a possible implementation of such designing techniques in a window. For example, we can see that the stylesheet associated with the main window is the following one:

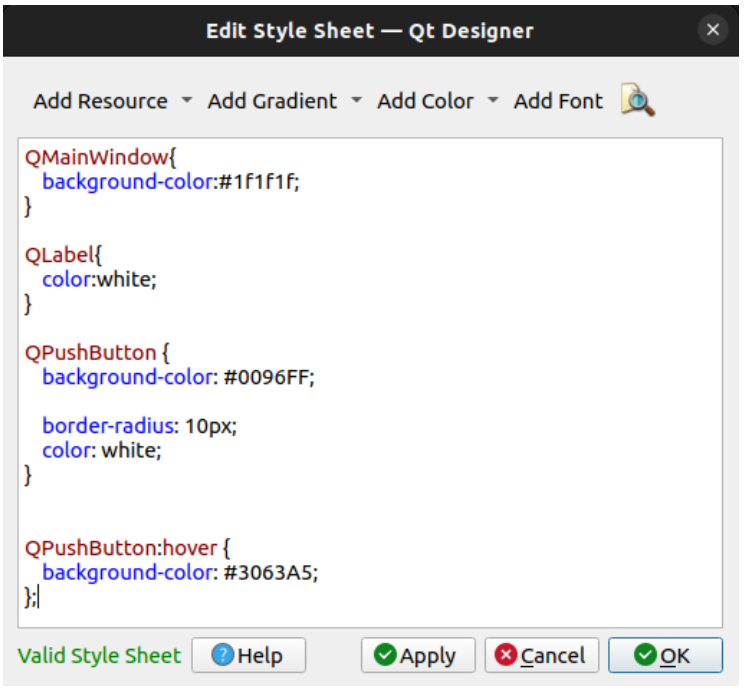


Figure 3.3: Stylesheet of the application main window

As we can see, in this stylesheet we set different constraints to the design of the widgets that appear in the main window. Firstly, we can see that we set the background color of the interface to #1f1f1f, which is in fact a dark color to accomplish the dark theme of the application (**see Appendix A** for the different color schemes information). Following the next lines of the stylesheet, we can see that we can access the different widgets properties through their names (QMainWindow, QLabel and QPushButton), and we can see that the color of the QLabel is set to white, which means that the text inside a label will be white. Finally, we can see the properties assigned to the button which are that the color will be blue (#0096FF), the appearance of it would be rounded, the text inside white and when we hover it we will see that the background color gets darker to acquire a satisfactory hover effect [6].

[22] https://developer.mozilla.org/en-US/docs/Web/CSS (accessed 27 May 2023)

Here we can see an image of the final application in which we can see all the above properties (background color, button color, button color on hover and text colors):

Figure 3.4: Same button while hovering (right) and not hovering (left)

The interface that you can see in **Figure 3.2** is an example of the usage of the QtDesigner application to design an interface dragging and dropping the different elements needed such as buttons, labels that contain images or not, and the interfaces area which is a QStackedWidget[23] that comes from the Qt library and allows us to mount the two other windows (the spectrogram sound synthesizer and the real-time sinusoidal transformation interfaces). The two other windows will be also created like the main window using the Qt Designer software and creating a new window. In the back-end we will take a look at how we link all the windows in one application.

While dragging the different widgets into the windows we can also see a menu on the right side of the Qt Designer software in which we can easily distinguish what widgets are added and their names. We can modify their names to be able to access them through the Back-End easily with Python.

As an example, in the main window we have this widgets and this names applied to them:

Object	Class
MainWindow	QMainWindow
centralwidget	QWidget
change_theme_btn	QPushButton
essentia_label	QLabel
essentia_logo	QLabel
nataly_logo	QLabel
page1_btn	QPushButton
page2_btn	QPushButton
stackedWidget	QStackedWidget

Figure 3.5: Widgets added to the main window and their names

Taking now into account the “interfaces area” of the main window, we can talk about two other windows that will be displayed at that widget (QStackedWidget), these two windows are also created with the QtDesigner software and we can see more in detail each of them.

[23] https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QStackedWidget.html (accessed 27 May 2023)

Spectrogram Sound Synthesizer (sinusoidal_spec_synth.ui)

For this window, I took into consideration the state of the art analysis made with the Sonic Visualiser application. In that case, we can see that we have an interface that allows users to have a spectrogram plot in the center and then the analysis parameters at the right side of the visualization. We can also notice that in that software the file loading is applied at the top of the application, therefore, users must be familiar with this kind of procedure and that is why I decided to put the file loading options on top.

As we also apply sinusoidal synthesis with a multiplication factor on the sinusoids (pitch shifting without timbre preservation) on this interface, we also need to have a slider to select the amount of semitones that we want to change from the selected region of the spectrogram pitch. The idea of using a slider to change the pitch comes from the analysis made on Audacity, where we can see that the frequency change is formed by this type of widget. I also thought that a change in pitch using a slider must be oriented vertically, as we also define the pitch as high or low.

One interesting part of the design of this interface was also looking for a way to let the user know that the changes made on the right side of the plot (analysis/synthesis parameters) were not applied and that it must recompute the synthesis to apply them. This was accomplished by surrounding the parameters area with an orange color, which gives the user the "Caution" meaning [7] whenever the user changes any of the inputs, and also by giving them a message on top which says "Not applied changes" to let the user know it. Taking into consideration the analysis made and the different considerations, the sketch of the interface looks like this:

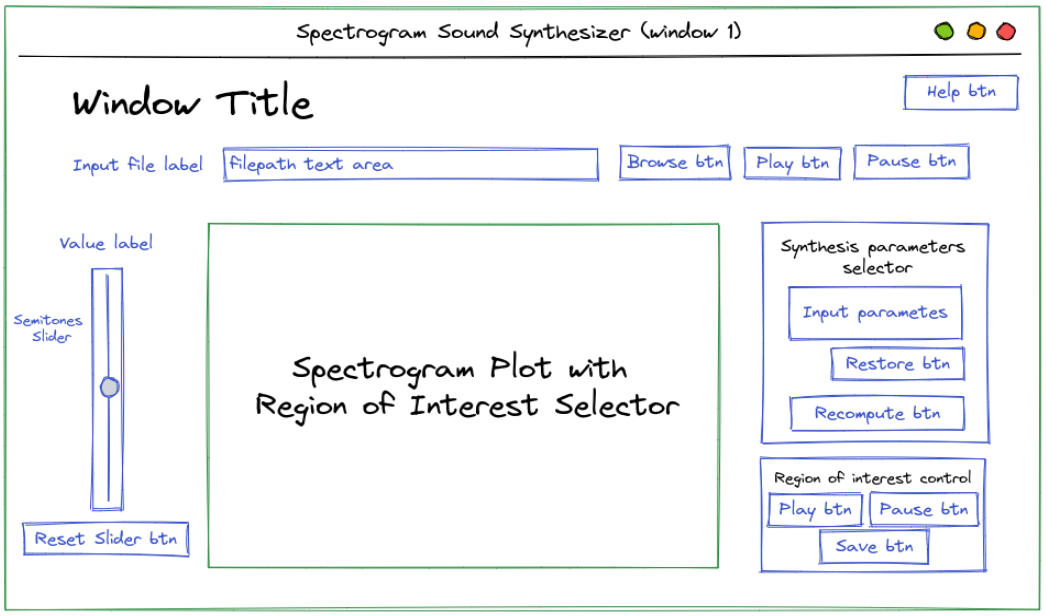


Figure 3.6: Sketch of the Spectrogram Sound Synthesizer window

The most important functionality of this window leans on the usage of the spectrogram plot together with a Region of Interest (ROI) in order to be able to select the appropriate area of the sound to analyze and then be able to synthesize it in a correct way using the chosen parameters and options. To acquire this behavior I used different widgets present in PyQtGraph library.

Firstly, to get the plot area I used the GraphicsLayoutWidget[24] of the library, this allows us to have a plotting window inserted in the interface. Secondly, in order to plot the Spectrogram in that area I used the ImageItem widget in combination with the HistogramLutItem, the first one helps us in order to plot the spectrogram and the second one allows us to have a colorbar at the the right-hand side of the plot to change the color intensities of it. Finally, for the region of interest I used the ROI[25] widget present in the same library which grants us the usage of a square region in the spectrogram plot from which we can map the selected area with values in our spectrogram array, which is very useful in order to synthesize one given specific area of a spectrogram without worrying about the other audio data around.

All combined together allows us to have the spectrogram plot as shown in this figure:

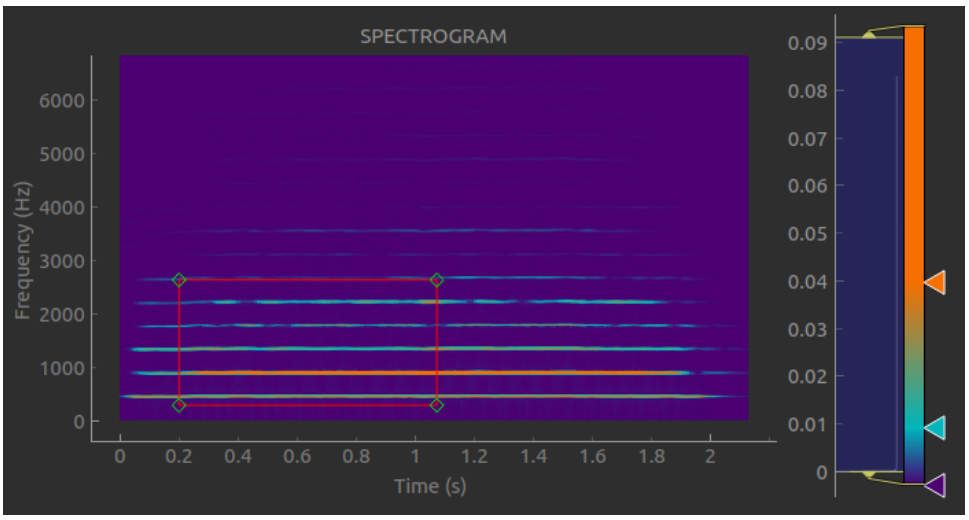


Figure 3.7: Spectrogram plot together with the Colorbar and the Region of Interest selector

All of these widgets are not available in the QtDesigner software as they are part of a plotting library developed by Qt (PyQtGraph). Therefore, the widgets need to be added programmatically in the interface in order for them to be used. All the other widgets around the interface can be added using the QtDesigner software, which helped a lot with the Front-end development of the project.

[24] https://pyqtgraph.readthedocs.io/en/latest/api_reference/widgets/graphicslayoutwidget.html (accessed 29 May 2023)
[25] https://pyqtgraph.readthedocs.io/en/latest/api_reference/graphicsItems/roi.html (accessed 29 May 2023)

Real-Time Sine Model Transformation (rt_sine_transformation.ui)

For the development of the second window interface I applied similar procedures as in the previous one, considering also that this is a simpler interface in which we do not want to overload the user with many parameters and information. We do not want to do that to ensure that the basic knowledge about what is being applied is fully understood so that the user can learn easily from it and the teacher can explain it in an easy way.

Taking into account the previously mentioned designing techniques we also add the change of pitch slider as a vertical slider and we let the user know the value of it and reset it whenever they want using a label and a button respectively.

In this interface the user will be able to record sounds captured from their microphone so this is why we must add a recording button. To let the user know when it is recording we add a red border whenever the recording button is pressed. In that way the user will be able to distinguish when the interface is recording or not, we can also add a label that says “Recording” to let them know.

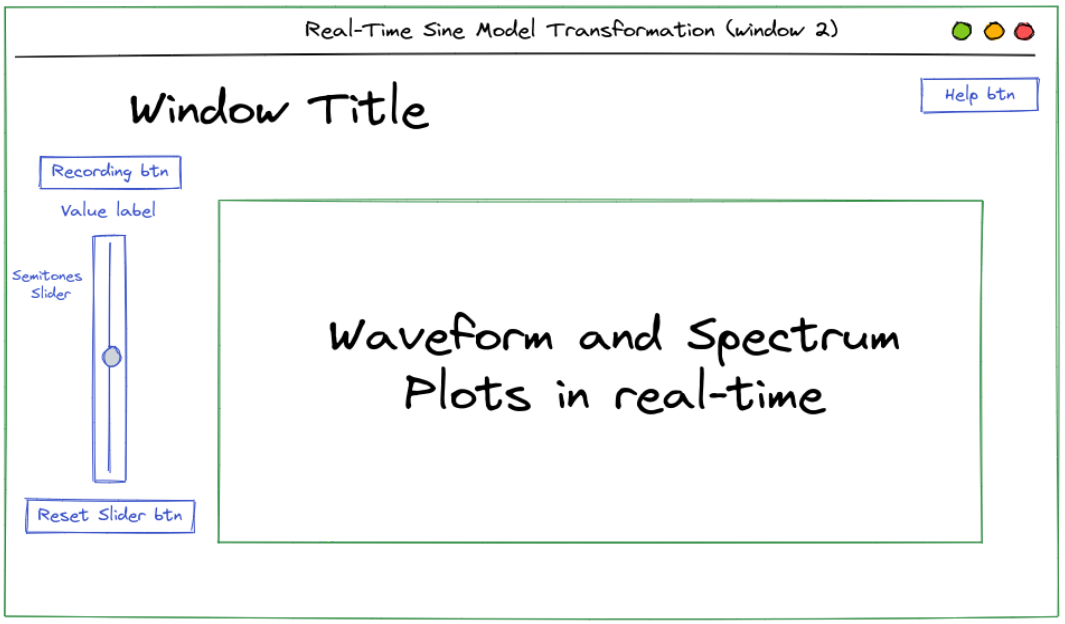


Figure 3.8: Sketch of the Sinusoidal Model Transformation window

As well as in the previously presented window, the plots in this interface are accomplished by using the GraphicsLayoutWidget from the PyQtGraph library, which enables us to graphically visualize our data and does so with high computation speed, therefore, it helps us in order to visualize plot changes in real-time. Nevertheless, in this case we did not apply the ImageItem to plot the data because we did not want to map the plot data with a Region of Interest selector to extract data from the plot.

3.1.2 Application logic (Back-End)

As we have seen in the previous section, whenever I had designed the three interfaces needed for the application, I ended up with three different .ui files exported from the Qt Designer application. Then, with those three files I needed to accomplish a methodology in order to link them with three other Python files in order to load the designed interfaces and start the development of the logic behind the different widgets added to the windows.

To do so, I organized the project with three Python files called main.py, sinusoidal_spec_synth.py and rt_sine_transformation.py. With those files created, I needed a way to link them to be presented in a single application with the different interfaces on it. The main idea to organize the files of the software was to have a principal file (main.py) with its interface loaded (main.ui), which was capable of changing between the two other interfaces (sinusoidal_spec_synth.ui and rt_sine_transformation.ui) linked with their Python files (sinusoidal_spec_synth.py and rt_sine_transformation.py), while showing them and also changing the software theme. PyQt5 does not explicitly tell you about a methodology to create this kind of project organization taking into account hierarchical links.

Therefore, I had to research a method to assemble the application with my idea using the created interfaces. Searching for a solution, I found out that a methodology that could be applied with Python using a class basis for all the .py files in the program[26]. Using that approach, I was able to link the code files and also separate their logic from the others in order to have the application distributed by the three different windows as I initially desired.

Thus, the project organization would be the following one:

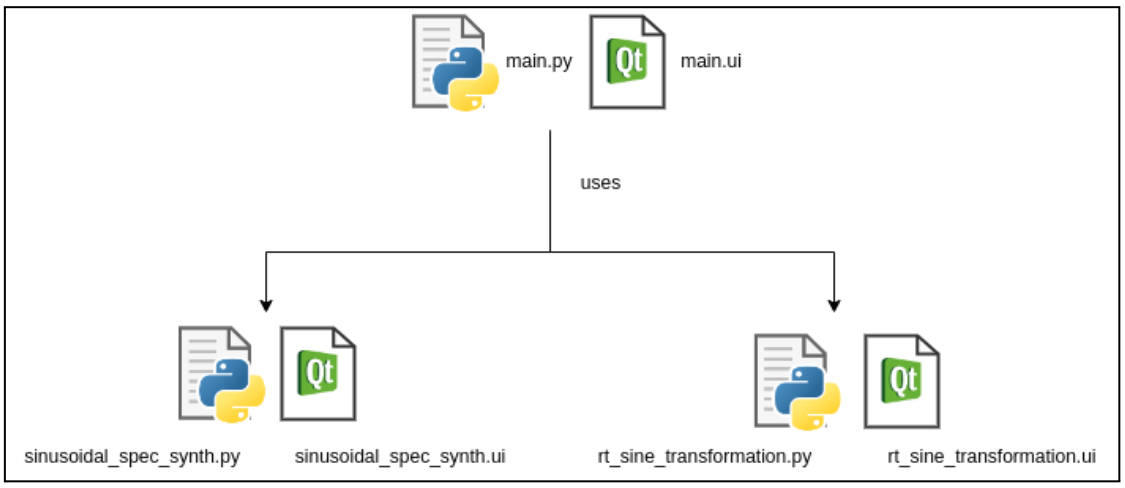


Figure 3.9: File organization of the application

[26] https://stackoverflow.com/questions/60904814 (accessed 30 May 2023)

As we can see in the previous figure, we have a main.py file which is basically a class that implements all the logic required for the main window of the application, that is, loading the main.ui file to see the designed interface and use its widgets while also operating with the two other classes with their corresponding interfaces (sinusoidal_spec_synth.py with sinusoidal_spec_synth.ui and rt_sine_transformation.py with rt_sine_transformation.ui).

Using the class design approach, we will have a class for each of the windows of the application and therefore, each class will have its own methods that can be developed and triggered whenever the user interacts with the widgets added in the interface. As an example, if we have a button added to the main.ui file, we can access it through the main.py file which is linked to the previous one and connect a callback method of the main class whenever the button is pressed. Taking into account the simplicity of this methodology I have developed all the logic behind the designed interfaces.

To sum up, we can see an example of how the procedures are being implemented, to do so, we can see how the main.py file works for the main window:

Firstly, we import all the necessary libraries for the logic implementation of the file (that includes sound processing libraries, visualization libraries and the necessary widgets from PyQt5). Secondly, we also import the two other classes from the corresponding Python files for the different windows inside the main one (sinusoidal_spec_synth and rt_sine_transformation). Whenever we have all this structure ready, we can create the class for the main.py file which is a PyQt5 widget based class, more specifically based on the QMainWindow widget. Then, we can define the methods to be applied in that class, the init method and all the others needed to accomplish the different functionalities of the software in that window. Taking into account the init method, we can see that the first aspect that we need to take into consideration is the import of the interface file (main.ui), whenever we apply that charge we have the two files linked and we can access the widgets of the previously designed interface. When we have the interface and the logic linked, we can access the widgets attributes and design the methods that implement their connected callbacks functionalities. Those functions will also be part of the class methods and will be separated from the other classes logic.

As an example, for the change theme button inside the main window interface, we can access it through the main.py file and connect a callback function that enables us to access the main window stylesheet and change it so, whenever the button is pressed the method is executed and the theme of the application changes from light to dark or vice versa. With that approach we can ensure the theme change for all the software because the stylesheet hierarchy is applied through the other classes that load the other interfaces with other widgets. This is a simple example and we will see in the Designed Algorithms section that we are implementing more sophisticated procedures for other widget on the interface such as the plotting areas.

3.2 Designed algorithms

The aim of this section is to explain the different algorithms developed to accomplish the correct link between the frontal and back end implementations of the application as well as the correct features desired for the software, the synthesis of short sounds using a spectrogram region of interest for the first one and the real-time synthesis of the sounds coming from the user microphone for the second one.

3.2.1 Sound analysis

As we have explained previously, for the purpose of this application we are going to apply the Sinusoidal Model. In terms of the analysis, we can distinguish different parameters to take into account for this algorithm. As the Essentia algorithm reference states[27], the parameters are the following ones:

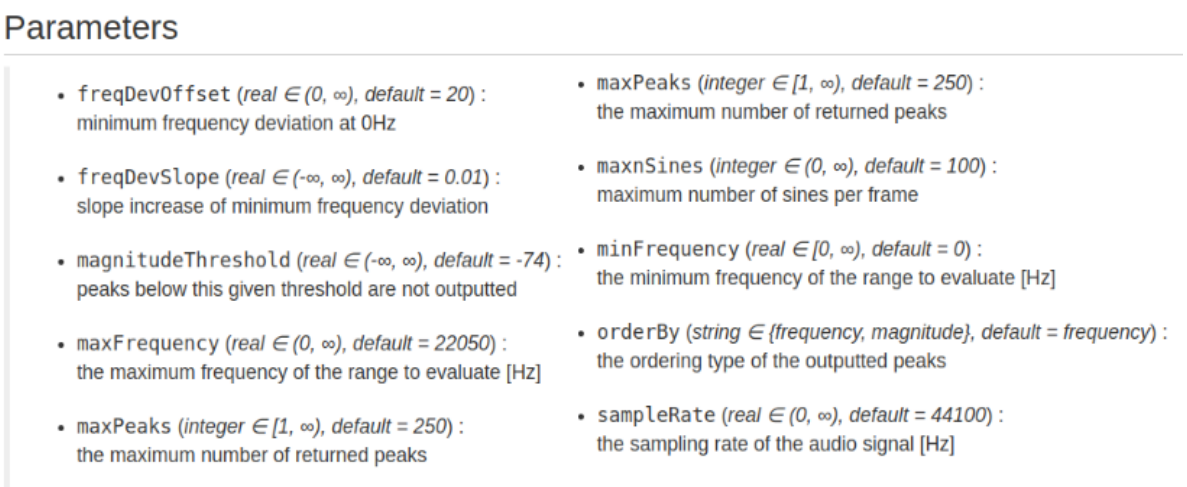


Figure 3.10: Essentia SineModelAnal function parameters

For the analysis applied to the audio signals that we are treating in this application, the parameters that we will change, i.e, not leaving them with their default value, are the following ones (with their corresponding new values):

maxnSines = 150
magnitudeThreshold = -80
freqDevOffset = 10
freqDevSlope = 0.001

The choice of these parameters are based on the quality and type of audio signals that we are treating while using the software. For example, for the maximum number of sines per frame, we choose 150 because we want to play it safe.

[27] https://essentia.upf.edu/reference/std_SineModelAnal.html (accessed 31 May 2023)

For the kind of audio signals that we will deal with, it could be sufficient by choosing the default value (100), but we choose 150 to encompass more cases. For the magnitude threshold chosen we can also notice that this value can be lower if we increase the audio quality, for 16 bits audio signals (the ones that we are handling) it would be sufficient to have approximately a -90 dB magnitude threshold[28]. In our case, we choose the value to be -80 dB, because Essentia's default value is -74 dB and can be related to algorithm performance specifications, and also because of the previous explanation for the kind of audio signals that we are analyzing. Therefore, the chosen value for this parameter is set to -80 dB.

Getting to the next chosen parameter, we can see that we choose a minimum frequency deviation offset at 0 Hz of 10. This parameter tells us what will be the chosen minimum deviation of the frequency tracking at 0 Hz, the offset will be modified linearly when increasing the frequency, this change in the deviation comes from the frequency deviation slope parameter. The frequency deviation slope is set to 0.001, this parameter tells us the slope that we are having in the frequency deviation when increasing the frequency, when we have higher frequencies to track, the deviation will be increased by 0.001 linearly. Taking into consideration the previously mentioned analysis parameters chosen for the Sinusoidal Model Analysis function implemented with Essentia. We also need to define the analysis parameters for the FFT, IFFT, Windowing and Synthesis functions. The parameters that we choose for those algorithms are the following ones:

Window type = hamming
M (Window Size) = 2001
N (FFT Size) = 2048
H (Hop Size) = 512

The decision of choosing these parameters comes from the fact that in the real-time sinusoidal transformation interface, we are using frames of 2048 samples in order to ensure the correct results and the speed needed in order to accomplish the real-time transformation. The Hop Size is chosen to be a quarter part of the FFT size [8] which is in fact a power of 2 bigger than M[29]. For the window type, we have chosen a hamming window which allows us to have a good trade-off between the main-lobe width and the side-lobe levels [9] in such a way that we can analyze and synthesize the sounds we tried with good frequency/time resolution[30]. The window chosen comes also from different tests implemented with the sounds that we use, we could also use a Blackman window which could give us good results too. In the case of the Spectrogram Sound Synthesizer interface, the user has the option to change these analysis parameters.

[28] https://www.soundguys.com/audio-bit-depth-explained-23706 (accessed 31 May 2023)
[29] https://www.coursera.org/learn/audio-signal-processing/lecture/tjEQe/stft-2 (accessed 31 May 2023)
[30] https://www.coursera.org/learn/audio-signal-processing/lecture/HFALS/stft-1 (accessed 31 May 2023)

We can distinguish between the analysis applied when we load the audio file (case of the Spectrogram Sound Synthesizer) and the analysis applied when we are getting audio frames from the microphone stream. The first one is identified as non-real-time analysis and the second is identified as real-time analysis.

3.2.1.1 From file (non-real-time)

For the analysis of audio tracks coming from loaded files, we can see that the procedure applied would be the following one:

First, when we load the audio file we call the MonoLoader Essentia function[31], which is capable of loading raw audio data from an audio file, downmixing it to mono and resampling it if the sampling rate does not match with the defined in the application (44100 Hz). Then, if the loader worked properly we call the compute method present in the sinusoidal_spec_synth.py file, which is capable of slicing the audio array returned by the MonoLoader into frames by using the FrameGenerator Essentia functionality[32], ensuring the specified Hop Size between slices. With those slices we iteratively compute the analysis functions (Windowing, Spectrum, FFT, sineAnal) and we stack the results into arrays that we will use in the synthesis of the spectrogram region selection.

3.2.1.2 From microphone (real-time)

For the analysis of audio signals coming from the microphone, i.e for the real-time sinusoidal transformation interface, we can see that the procedure that we apply is a little bit different.

In our software, we deal with the microphone capture by using the stream functionality present in the PyAudio library[33]. Taking into account this method, we start a microphone capture stream with 2048 samples per frame (CHUNK parameter of the stream). As we may recall, the parameters defined for the sinusoidal model in both interfaces took into consideration that the FFT size is 2048 and that the Hop Size in that case should be 512. Therefore, we must ensure that this is accomplished in the real-time sinusoidal transformation interface.

However, we have a problem with the usage of microphone streams through PyAudio. Those streams are capturing frames of 2048 samples sequentially from the microphone, but the hop size applied to that frame-by-frame sequences is 2048, and therefore, we have to design a methodology to take into account the frames that should be captured between the two microphone captured frames if the library applied the 512 samples Hop Size.

---

[31] https://essentia.upf.edu/reference/std_MonoLoader.html (accessed 1 June 2023)
[32] https://essentia.upf.edu/reference/std_FrameGenerator.html (accessed 1 June 2023)
[33] https://people.csail.mit.edu/hubert/pyaudio/docs/#class-pyaudio-stream (accessed 2 June 2023)

In order to accomplish that, I designed the following scheme to recover the frames in between that give us the 512 samples Hop Size to be able to apply the sinusoidal model algorithms defined before. The diagram below shows the methodology applied to get those auxiliary frames from the microphone captured frames:

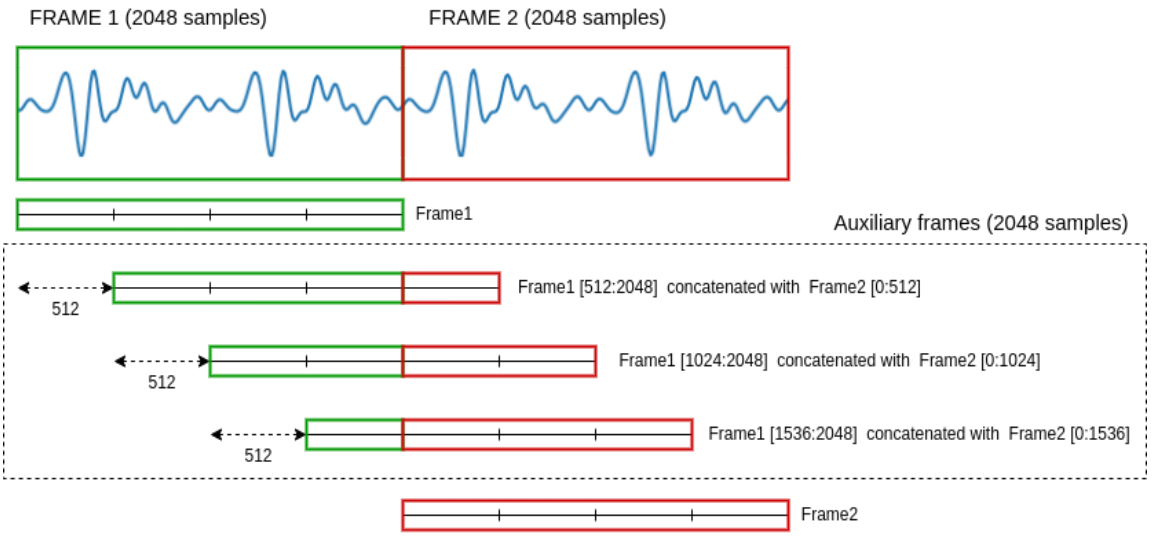


Figure 3.11: Computation of auxiliary frames to apply Essentia computations using a 2048 samples FFT Size and a 512 samples Hop Size[34]

As we can see in the figure above, the FRAME 1 and FRAME 2 are the frames that are being captured from the microphone in real-time and the auxiliary frames are computed using the stored data from previous frames iteratively, using this approach, we can ensure the desired hop size and apply the Essentia functions correctly.

The idea was simple, for the first captured frame (FRAME 1) we do not need to get any auxiliar frames, but when we capture the following one (FRAME 2), we need to get three auxiliar frames using the previously captured frame (FRAME 1) to ensure that the 512 samples hop size is accomplished and we have those frames to compute the Essentia functions. This approach is given by the neediness of acquiring the 512 samples hop size using 2048 samples frames, by using any other sizes we should change the methodology so, that is why this interface doesn't let the user change the analysis/synthesis parameters in the first place.

In this case, we already have the frames to compute the analysis, we do not need to apply the FrameGenerator functionality and we can proceed with the FFT, SineAnal and the SineSynth methods with the computed frames.

[34] https://draw.io

3.2.2 Sound transformation

For the sound transformations applied in the two interfaces, we can distinguish between the transformations applied. First, we can see that we apply a pitch shifting transformation in both interfaces, in order to accomplish that, we apply a multiplication factor to the analyzed sinusoids in the synthesis. The difference between the two applied synthesis in the two interfaces is that in the Spectrogram Sound Synthesizer we first filter the sinusoids by means of the Region of Interest selected inside the Spectrogram.

3.2.2.1 Filtering

In the case of the Spectrogram Sound Synthesizer interface, we must ensure that the synthesized region of the sound is the one that we select on its spectrogram, therefore, we need to keep track of the region of interest (ROI) position and map the coordinates to the corresponding index values in the spectrogram array, which will help us later in order to decide which sinusoids we use in the synthesis part. When we move the ROI inside the Spectrogram, the widget has a connected callback function that is called when we finish moving it around the plot, and then, the SelectedRegion method is applied, which is capable of mapping the position of the ROI with the indexes of the array that contains the spectrogram data. Therefore, by having those indexes, we can set all the sinusoids extracted from the sinusoidal analysis and living outside the limits defined by the indexes to 0, ensuring the synthesis of the region of interest in the Spectrogram. The following diagram tries to explain the algorithm visually:

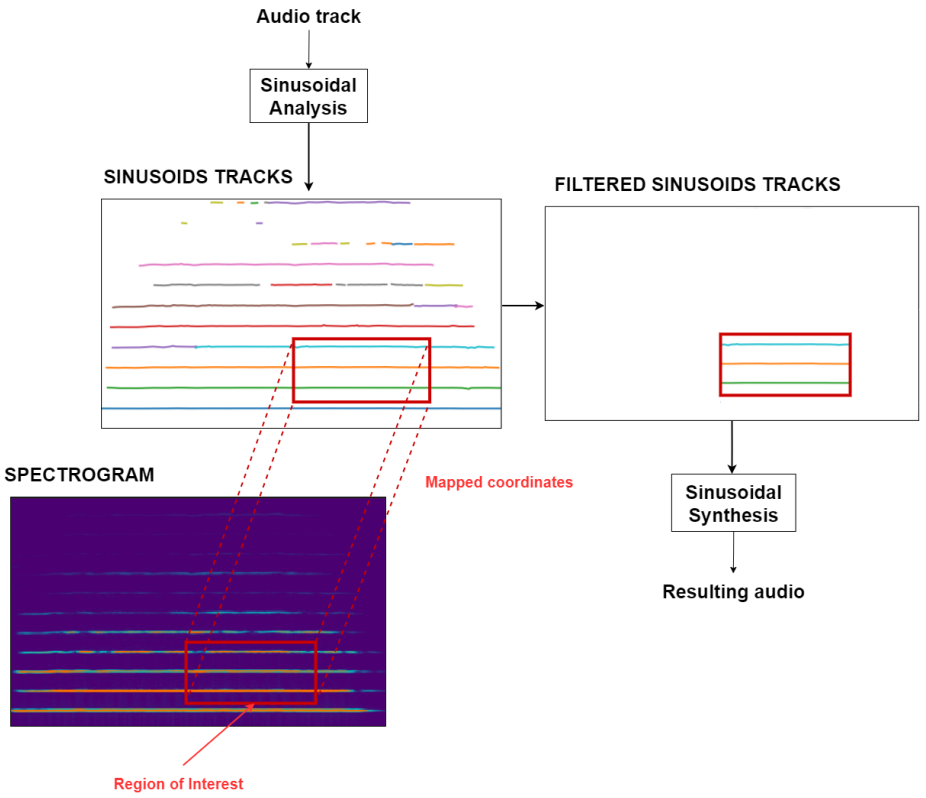


Figure 3.12: Filtering applied in the Spectrogram Sound Synthesizer

3.2.2.2 Pitch Shifting

For the case of the pitch shifting transformation applied to both of the interfaces, we only need to apply a multiplication factor to the sinusoids extracted from the Sinusoidal Analysis of the sound. In that process we extract the sinusoids information by applying the SineAnal Essentia function, which we can later multiply by a given factor. The given factor comes from the slider value selected by the user. The slider values go from -12 to 12 Semitones[35], which is currently one octave lower or higher[36].

Then, the pitch shifting is applied by following this expression:

$$f_{final} = 2^{(semitones/12)} \cdot f_{initial}$$

Figure 3.13: Semitones pitch shifting

In the code, we use the following one, which is exactly the same:

$$r = 2^{(1/12)}$$

$$f_{final} = r^{semitones} \cdot f_{initial}$$

Figure 3.14: Semitones pitch shifting applied in the software

Following the previous scheme to compute the multiplication factor, we can multiply the captured sinusoids in the analysis by this factor, which gives us the desired pitch shifting. This procedure is applied in the two interfaces in the synthesis.

3.2.3 Sound Synthesis

For the sound synthesis, we make use of the SineModelSynth Essentia function[37], which allows us to apply the sine model synthesis given a sine model analysis/transformation, i.e, the magnitudes, the frequencies and the phases of the sinusoidal peaks. The result of applying such functionality is a magnitude and phase spectra, which it can then be converted to a sound by applying the IFFT function[38] and the OverlapAdd function[39]. These two algorithms allow us to ensure the correct synthesis output for the frames that we are using in the synthesis and the transformations. The synthesis is independent from the analysis, we could choose the parameters that we wanted, but, in this case we use the same FFT Size and Hop Size as described in the analysis section.

[35] https://mastering.com/pitch-shifter (accessed 2 June 2023)
[36] https://www.guitarpitchshifter.com/pitchshifting.html (accessed 3 june 2023)
[37] https://essentia.upf.edu/reference/std_SineModelSynth.html (accessed 3 June 2023)
[38] https://essentia.upf.edu/reference/std_IFFT.html (accessed 3 June 2023)
[39] https://essentia.upf.edu/reference/std_OverlapAdd.html (accessed 3 June 2023)

Why the Sinusoidal Model?

As we have seen in the State of the Art section for the sinusoidal model, we know that it is a higher level representation based on a mathematical model that allows us to represent sound signals as a collection of sinusoids. Taking into consideration this model, we can say that “it is a quite general technique that can be used in a wide range of sounds and offers a gain in flexibility compared with the direct STFT implementation.” [10]. Also, in favor of the usage of such model, we can state that “a large class of acoustical waveforms including speech, music, biological, and mechanical impact sounds can be represented in terms of estimated amplitudes, frequencies and phases of a sum of time-varying sine waves” [11], which is in fact, what the model does.

As stated before, the sinusoidal model allows us to represent the sound as a collection of sinusoids, using that approach, we can analyze the audio signals and extract a collection of different sines from it, which can be transformed to give us the results that we want. The sinusoidal model is one of the models that gives us a more flexible representation while getting a good compromise between two important factors, the sound fidelity and the model computing time [10].

One of the effects that we want to accomplish in this software is the pitch shifting, but it is interesting to notice that our pitch shifting implementation does not take into account the timbre preservation, what we are applying is a multiplication of the partials by a scaling factor, determined by the slider and converted from semitones to linear [10]. That transformation is simplified by the usage of the sinusoidal model, which brings us the magnitudes, frequencies and phases of the sinusoidal tracks, and we are only multiplying their frequencies to get the pitch shifting transformation.

We have also chosen this model because it has been proved that it can be actually fitted to acquire good results with speech signals [12] [13], which are not our main focus of attention for the purpose of the application, but as we are capturing audio signals from the users microphone, the most common signals that we will receive could be speech signals too.

3.3 Software development

In this section we can take an overview of the software development process strategies that I chose to take for the different creation and control of the project. First, we will see where the code is stored and published (code repository), and then we will take a look at the software tools used to implement the coding strategy.

3.3.1 Application software management

For the software management and development I used two different tools: GitHub and PyCharm. For the software version management I created a **GitHub** repository[40], that helped me in order to keep track of the software modifications and allowed me to publish the project efficiently.

In the repository we can identify different aspects to take into account, first of all, the software license is specified in it, which is an GNU Affero General Public License (v3.0), this kind of open license allows users to use it for any purpose but requiring to release the modified versions under a compatible license. Therefore, we can state that the Nataly software is based upon open source characteristics, granting the advantages of those types of softwares [14] and facilitating community development.

In the repository, we can also check the Readme file which contains all the necessary instructions to use the application (install the corresponding libraries) and some documentation images to let the user take an overview of the software without needing to install it. I have been the only person developing it so, I did not need to use branches for the different features that were added to develop the application. In the future, these kinds of methodologies should be applied if there are more developers involved in the application development.

Taking into account the repository organization, as I have stated in different sections, the project is based on a principal file called main.py, which is capable of executing all the application by using the different Python files available in the project such as the sine_spec_synth.py, the rt_sine_transformation.py and more.

We can also see that we have two different folders in the repository called assets and documentation_images, these two folders are the ones in charge of storing the assets of the application (logos and helping diagrams) and the documentation images needed for the Readme to be filled with images respectively. And last but not least, we also have a sounds folder which contains a list of different short sounds to be used in the Spectrogram Sound Synthesizer interface.

[40] https://github.com/albertgubau/Nataly

Finally, to indicate the modifications that have been made or have to be made, I use the issues of the repository. Using that tool I can know if there is any problem with the software or one of the implementations that it has. Using the issues we can modify the project taking into account the different suggestions or bugs and then, when we modify it, we create a pull request in order to make a petition to change the project to the owner of it. As in this project I am the only developer involved, these procedures were not applied, but in the future we will have to take into account this plan of action to modify the software and enhance it.

3.3.2 Software tools used

For the software development process I used the **PyCharm** Integrated Development Environment (IDE), an integrated development environment used to code with Python, which helped me in order to organize the software files, execute them and also add the necessary libraries to it. For the purpose of this project and the software that was developed, I used the following Python modules:

- **Numpy**[41]: Used for array computing using Python (version 1.21.5)
- **Essentia Standard**[42]: Sound processing library (version 2.1b6.dev858)
- **PyAudio**[43]: Audio streaming library (version 0.2.13)
- **SoundDevice**[44]: To play and record sound with Python (version 0.4.5)
- **PyQt5**[45]: Used for interface design with Python (version 5.15.7)
- **PyqtGraph**[46]: Graphics and GUI library for Python (version 0.13.1)

3.4 Software requirements

For the software requirements, we can distinguish between two types, the functional requirements and the non-functional requirements. The functional requirements are the ones that allow us in order to specify the behavior, functionalities and the different features that the software must include in order to accomplish its desired purpose.

On the other hand, the non-functional requirements are the ones that describe the characteristics and qualities that the software must have, in which devices the application must run, which characteristics must those terminals have, what should be the software performance and more.

[41] https://numpy.org
[42] https://essentia.upf.edu/essentia_python_tutorial.html (accessed 4 June 2023)
[43] https://people.csail.mit.edu/hubert/pyaudio/
[44] https://python-sounddevice.readthedocs.io/en/0.4.6/
[45] https://www.qt.io/qt-for-python
[46] https://www.pyqtgraph.org

For the case of this application, the **Functional Requirements** are the following ones:

- The application must be simple and intuitive, the interface must not be overloaded with lots of information giving the user confusion and not allowing them in order to use the software correctly.

- The software must bring help options to the user, allowing them to look at it to have an explanation of how the application works and how the different plots are computed.

- The interfaces should be visually appealing and must allow the user in order to change the theme between light and dark.

- The software must be able to load short audio files correctly, giving the user the right information whenever the audio file cannot be loaded.

- The application must handle the different errors that may appear during the user interaction with it, giving them explicit explanations that help them in order to solve the problems properly.

- The computation speed of the software needs to be fast, allowing the user to work with it without taking a lot of time with the algorithm procedures.

For the **Non-Functional Requirements** we should consider that:

- The software should be able to execute under the following software specific requirements:

    - Operating Systems

    - Ubuntu 20.04 LTS (or greater)

    - Processor

    - x86 64-bit CPU (Intel/AMD architecture)

    - RAM

    - 2 GB RAM (minimum) and 2 GHz processor (minimum)
    - 4 GB RAM (recommended) and 4 GHz processor (recommended)
    - 2 GB free disk space (minimum)
    - 5 GB free disk space (recommended)

    - **Screen resolution**

    - Minimum resolution of 1165x625

3.5 Software architecture

The application architecture is designed to have a main file (main.py) that takes charge of the application, using different files in order to load the interfaces, the different assets and the helping windows of the software.

We can see a diagram of the software architecture in the following figure:

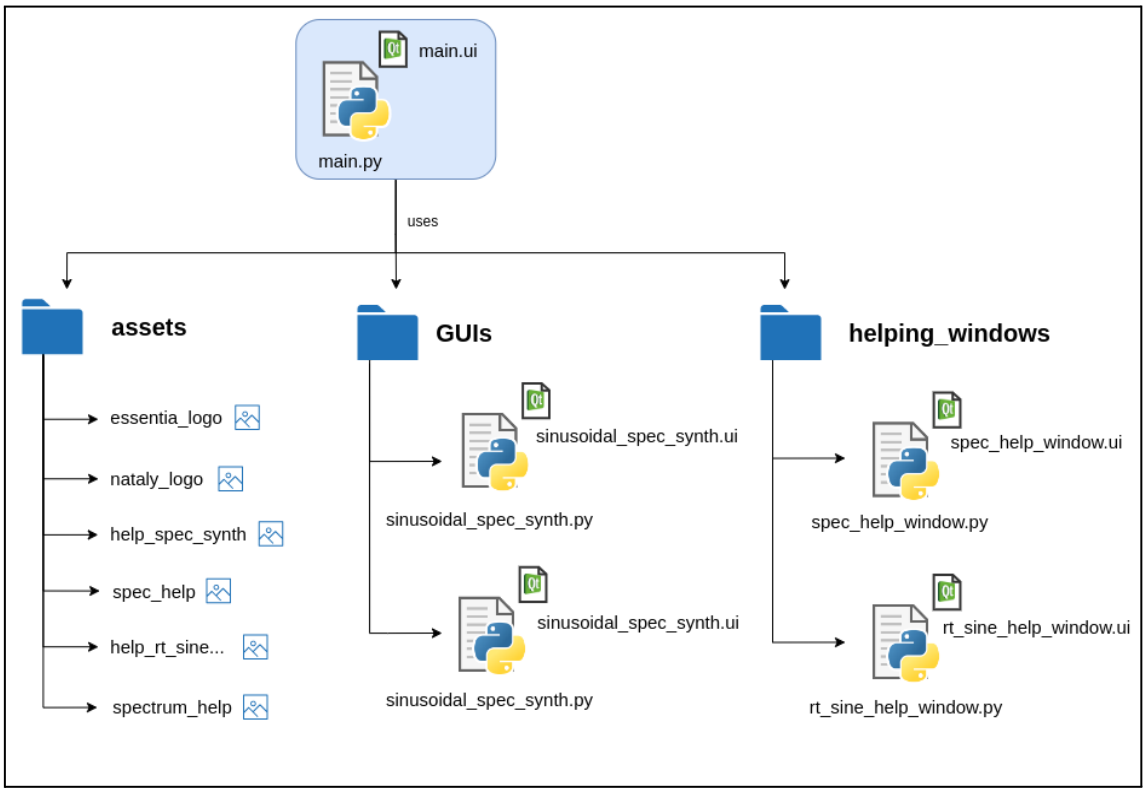


Figure 3.15: Software architecture of the Nataly application

As we can see in the previous figure, the main file loads the interface from the main.ui Qt file and uses different files stored in different folders. Firstly, we have the assets folder in which we store all the different images needed in the application (logos and helping images for the helping windows). Secondly, we have the GUIs folder in which we have the two different Python files corresponding to the two different interfaces present in the application together with their interface files. And last but not least, we have the helping_windows folder in which we store the two Python files in charge of the helping windows interface together with their interface files too, in which we load the images from the assets folder. Therefore, the code files inside the GUIs use the helping_windows folder in order to initialize those windows properly.

Chapter 4: Results

This chapter aims to explain the final results obtained for the application development and also the outcomes of the application evaluation with users that tested the program. Here we can also take a look at the resulting interfaces of the application from the Front-End software development methodologies explained in previous sections as well as the resulting outputs from the algorithms and interactions that the user may have with the application.

4.1 Final application

For the resulting final application we can distinguish two different sections, the one dedicated to the Spectrogram Sound Synthesizer interface and the one focused on the results obtained from the Real-Time Sinusoidal Transformation frame. Before looking at the two interfaces results, we can see an overview of the main interface of the application, the one implemented at the main.py file and linked to the main.ui interface file as explained in both the Front-End and the Back-End sections. As we have stated before, the final application can be found at the software repository[47].

The final application looks like this:

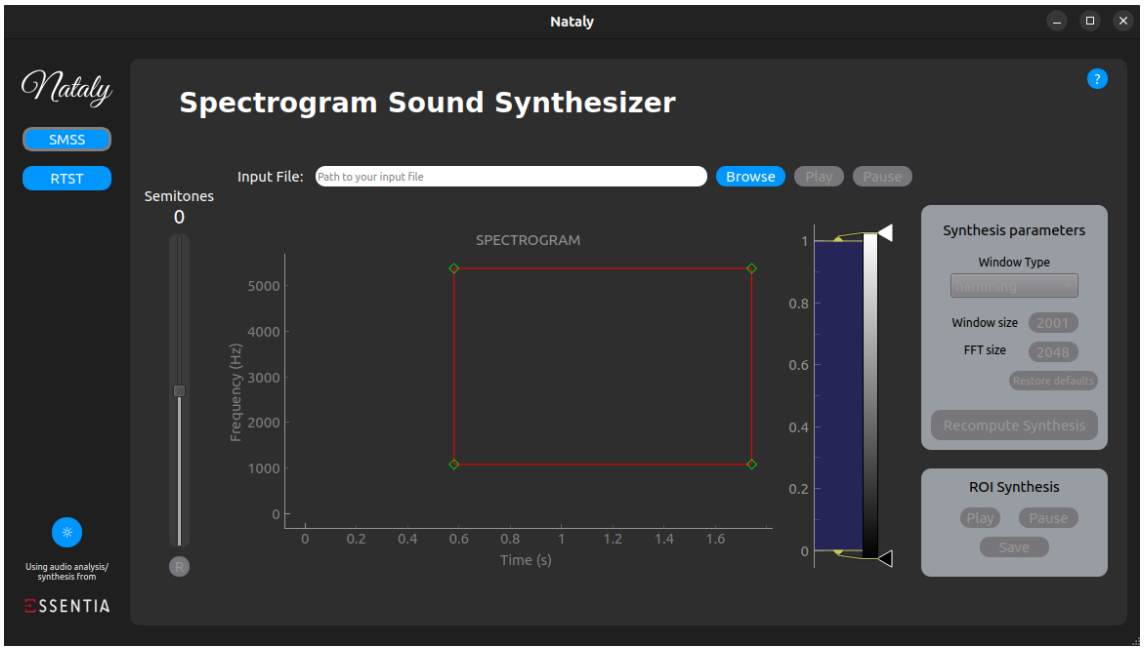


Figure 4.1: Nataly application with the dark theme

As we can see in the previous figure, the implementation takes into account all the sketches made for the main interface. We have a menu on the left that helps us to select the interface that we want to show and that also allows us to change the color theme of the application. As we have stated in previous sections, inside the main interface we have the interfaces area, capable of showing us the two main windows of the software.

[47] https://github.com/albertgubau/Nataly

The main interface of the application is launched by executing the main.py Python file present in the project, which is capable of encapsulating all the application by using the other files present in the software and linking them as stated in the Back-End section. As we can see in the previous figure, the buttons to select the interfaces are surrounded by a gray border whenever we are seeing the window corresponding to that button. And the change color theme button changes the color scheme of the application between the two themes presented in the appendices (**see Appendix A**).

4.1.1 Spectrogram Sound Synthesizer

For the Spectrogram Sound Synthesizer interface, we have the following result:

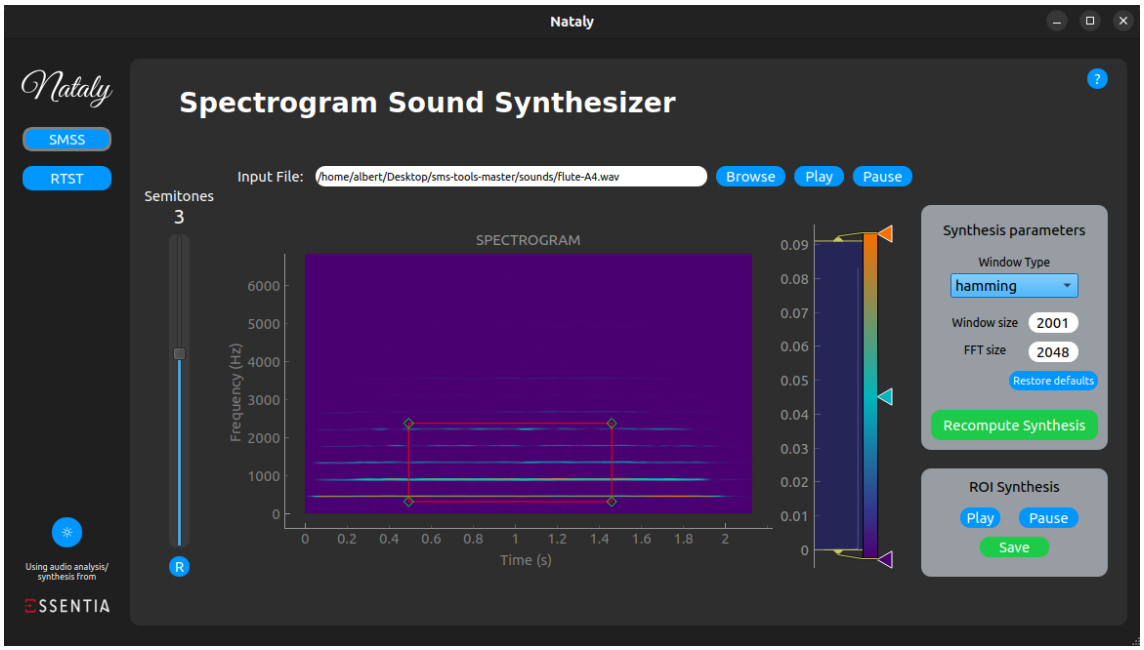


Figure 4.2: Nataly application with an audio loaded in the Spectrogram Sound Synthesizer interface with the dark theme

As we can see in this figure, the interface has different parts that need to be taken into consideration. Firstly, we can see that we have a loading audio scheme that will help us in order to do so. We can see the parts in more detail in the following figure:

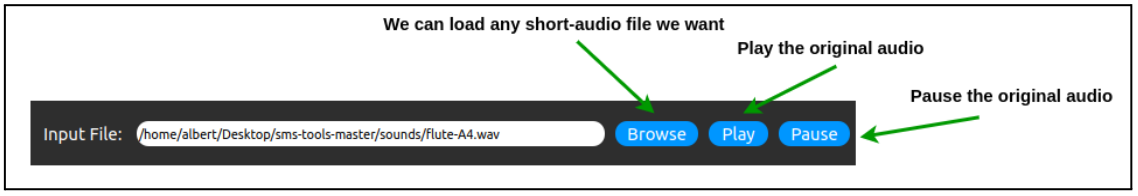


Figure 4.3: Input file scheme of the Spectrogram Sound Synthesizer interface

Then, we can also see that the interface presents the slider capable of changing the pitch of the synthesized audio by using it. In this slider we can differentiate some aspects that need to be taken into account and we can see in the following figure:

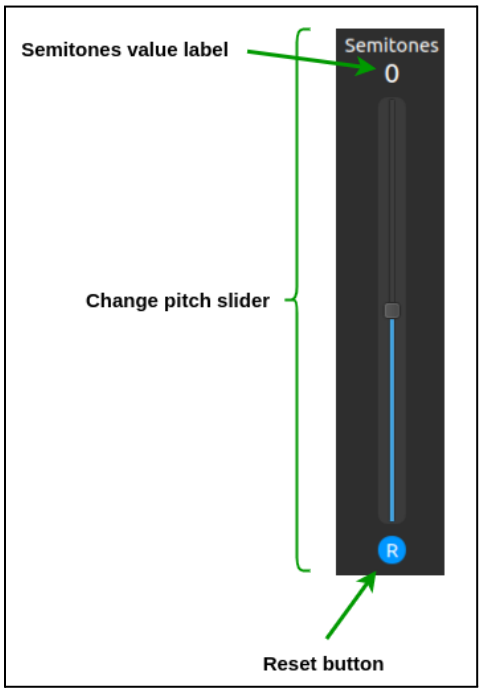


Figure 4.4: Slider of the Spectrogram Sound Synthesizer interface

Following the window composition, we can also take a look at the spectrogram plot, which presents the following structure:

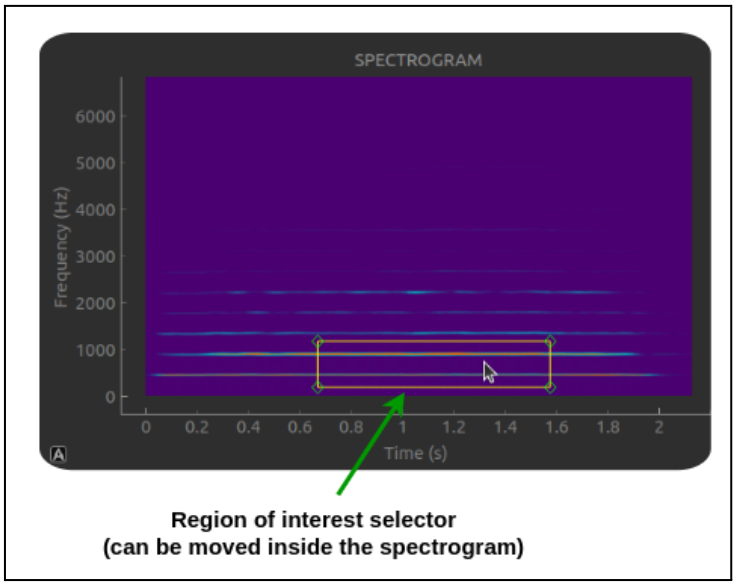


Figure 4.5: Spectrogram plot of the Spectrogram Sound Synthesizer interface

As the figure states, the region of interest selector can be moved inside the spectrogram plot, allowing the user to synthesize the piece of the audio track that it wants by using the region of interest player described in the next explanations.

Looking at the right side of the spectrogram plot, we can see a widget that allows us to modify the plot color scheme. This plot is the histogram colorbar and identifies the minimum and maximum values present in the spectrogram allowing to decide which color identifies each magnitude. By changing the color options we can see different plots of the spectrogram with different intensities for each frequency track. The colorbar presents the following structure:

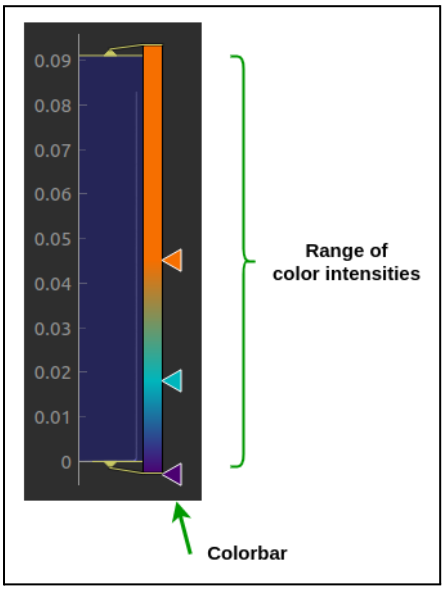


Figure 4.6: Colorbar widget of the Spectrogram Sound Synthesizer interface

Finally, we can take a look at the two widgets capable of changing the analysis/synthesis parameters and playing the resulting synthesized piece of sound from the selected region of interest in the spectrogram, these two are:

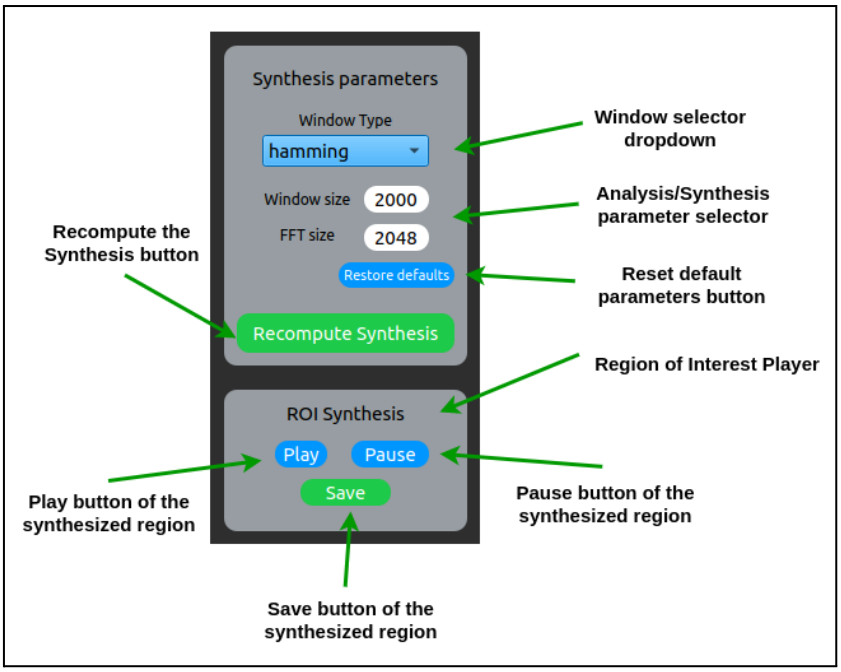


Figure 4.7: Right-side widgets of the Spectrogram Sound Synthesizer interface

4.1.2 Real-Time Sinusoidal Transformation

For the Real-Time Sinusoidal Transformation interface, we have the following result:

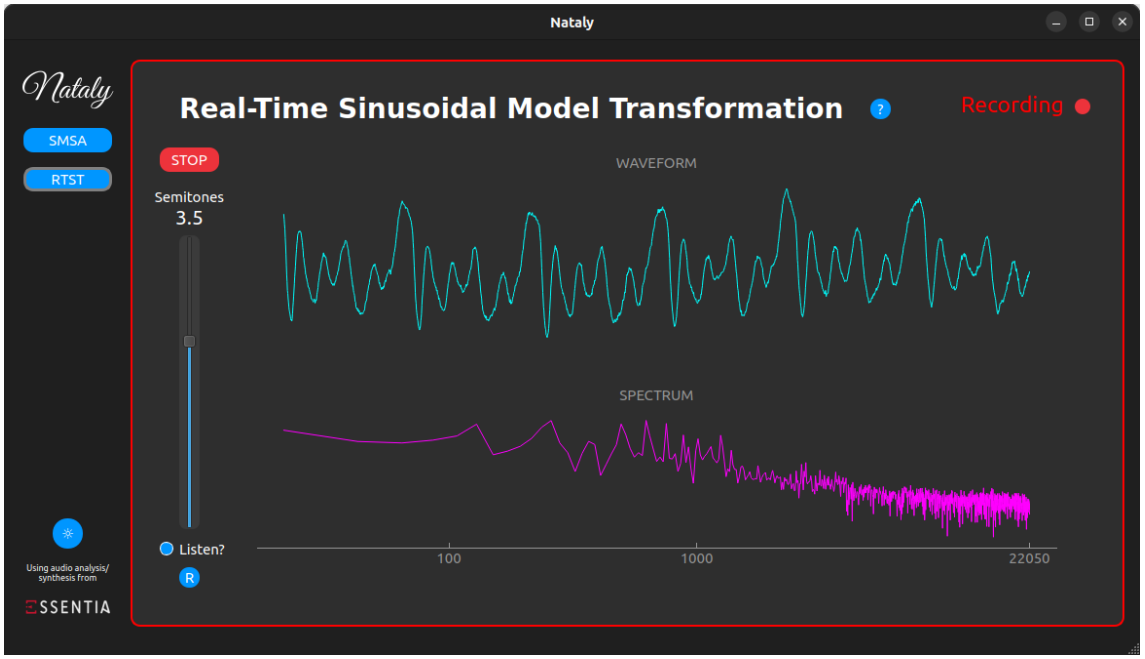


Figure 4.8: Nataly application with audio being recorded in the Real-Time Sinusoidal Transformation interface with the dark theme

As we can see in this figure, the interface has different parts that need to be taken into consideration. Firstly, we can take a look at the main plots of the interface. Those plots are updated in real-time taking into account the input sounds captured from the user microphone. The two plots show the following data:

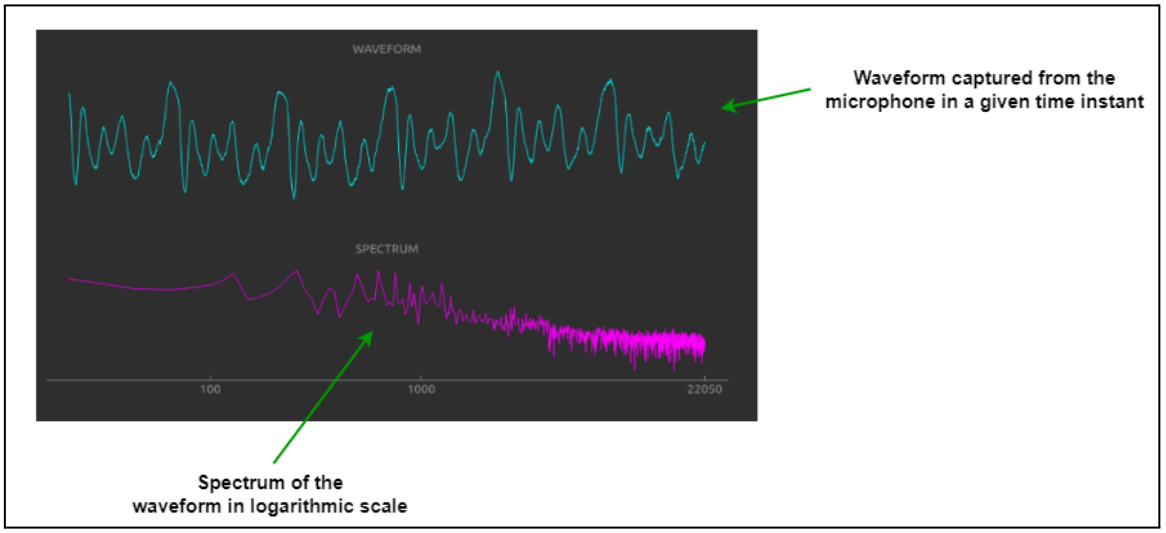


Figure 4.9: Waveform and spectrum plots in the Real-Time Sinusoidal Transformation interface

At the left side of the plots widget we can find the pitch change slider that works the same way as explained before in the Spectrogram Sound Synthesizer. We can differentiate more components in the slider part that are explained in the following figure:

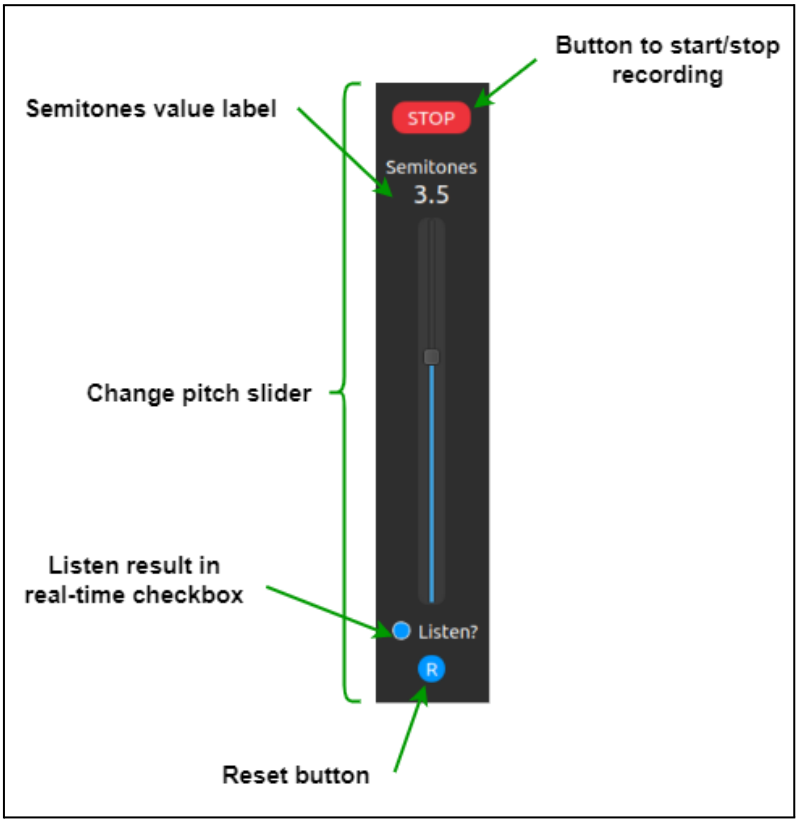


Figure 4.10: Slider of the Real-Time Sinusoidal Transformation interface

Finally, at the top right position of the interface, we can see that a label pops up whenever the user is recording, this label shows the user that the audio captured from the microphone is being recorded and we also border the interface with a red line to let the user know it as we can see in **Figure 4.8** :

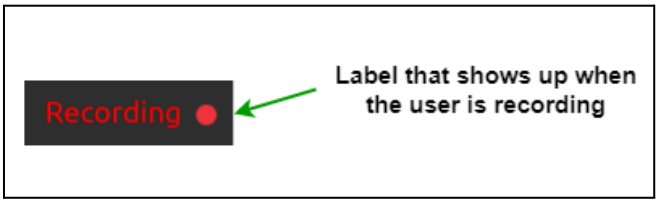


Figure 4.11: Recording label in the Real-Time Sinusoidal Transformation interface

4.2 Software evaluation

For the software evaluation, I conducted a usability questionnaire to different audiovisual systems engineering students in order to know their perception about the application and the possible enhancements that could be made in order to improve the different functionalities/capabilities of the software. I focus my attention on those types of users because they would be the main target of the application. I want to ensure that the software can help students in their learning process. Knowing that, I can infer that the teachers could use it too for their lessons. The questionnaire was based on different tasks that I defined and believed that were capable of testing the majority of functionalities of the application as well as the interactions between the interfaces and the users. The three tasks conducted by the users were the following ones:

Task 1: Navigate freely through the application and try the interfaces.

Task 3: Use the Spectrogram Sound Synthesizer interface to load a flute sound and synthesize the end region of its spectrogram to make it sound with a higher pitch.

Task 2: Use the Real-Time Sinusoidal Transformation interface to listen to your own voice with a pitch shift.

The results of the test can be seen in the appendices section (**see Appendix B**). From the results we can see that the majority of users find the software intuitive and simple, most of them choose the second interface (Real-Time Sinusoidal Transformation) as the most intuitive one, maybe because it is simpler and not as much overloaded as the other interface. There are a lot of users who think that they like the aesthetics of the application, which was one of my main objectives from the software design part. From the questionnaire results can also extract that the interfaces work properly and that the majority of users do not encounter problems while developing the previously defined tasks, which means that the software allows the users in order to load short audio files correctly.

For the quality of the resulting audios, most of the testers ensure that the quality is good, although it could be enhanced, and that the execution time varies between fast and normal, which are good results taking into consideration the software limitations. Taking into account the helping options that the users have in both interfaces, most of the users state that they find those options helpful, which is an important point for those that try to learn how the application works. Taking into consideration the error handling capability of the software, most of the testers did not find any problem, and the ones that find an issue, are correctly informed about the error by the application. Finally, for the questions asking the users if they think that the application can be used for the learning/teaching process, there are lots of them who think that the software would fit those necessities, which again, is an important point taking into consideration the initial objectives of this project development.

Chapter 5: Conclusions

This chapter summarizes the conclusions that I have about the project development. In this section we are going to take a look at the obtained results from the first idea, the different drawbacks of the project from my point of view, the possible improvements and the next steps that should be taken in order to enhance it or keep developing it.

5.1 Obtained results from first idea

The obtained results from the perspective of the first idea that I had are very satisfactory. I managed to develop a simple sound processing application that, as the evaluation results show, could help students in order to work with the Sinusoidal Model and learn from different sound processing techniques and procedures. From my point of view, the software looks intuitive and the interfaces have a good design quality. The frontal part is well combined with the back role and the application covers the initially desired functionalities. Obviously, the project has some drawbacks that need to be taken into account, but to summarize, the obtained results are more than acceptable.

5.2 Project drawbacks

Some of the different project drawbacks that I encountered while developing the application are the following ones:

Firstly, the Spectrogram Sound Synthesizer interface does not allow the users to load large audio files. This problem comes from the fact that in our software we are treating the signals as arrays, which occupy a given memory space and when we have large items stored in them, the computations become slower. Moreover, when we implement some of the Essentia functions such as the FrameGenerator with a large array, the computation speed decreases and the application tends to not respond properly. This issue could be treated by applying multi-threading/multi-processing procedures in our code, whenever we load or process an audio track.

Secondly, taking into consideration the previous issue, in the Real-Time Sinusoidal Transformation interface, the application tends to run slower whenever we are recording a large audio. This problem is strictly related to the previously mentioned issue, taking into account the drawbacks of dealing with large arrays that occupy lots of memory space. One of the other drawbacks that I have encountered is that the application can not be used either on Windows or macOS operating systems. This issue comes from the fact that at the current date, Essentia is not compatible to be used on Windows systems and PyAudio has some limitations on macOS based terminals. Also, the application is not responsive in terms of the window sizes, it should be considered that the software could run in different screens and manage the available viewports to ensure that all the different widgets are seen and that the software functionalities can be applied properly.

Finally, the software can not be executed as a standalone application at the moment of writing this thesis. The initial idea was to bring this capability to improve the simplicity of the application usage, but the processing speed that we get by creating the standalone application is much lower than executing it using Python from the Ubuntu terminal.

5.3 Possible improvements

Looking at the different project drawbacks presented above, we can easily identify some of the possible improvements that could be made in the application. The ones that should be taken more into consideration should be the ones related with bringing the application execution to the most used operating systems, i.e Windows, macOS and Ubuntu. It should also be taken into consideration the computing speed improvement in the case of loading large audio files, which would enhance the software a lot and could increase the number of final users that could use it in order to apply their different sound transformations.

At last, one of the possible improvements could be applying multithreading or multiprocessing techniques to the software that could bring a huge upgrade to the application usage and capabilities, we could manage to separate the different processing algorithms asynchronously ensuring a higher computational efficiency and a better interaction between the user and the application.

Looking at the last question of the usability questionnaire, users left some possible enhancements that the application could implement, starting from adding a feature which allows the users to decide where they want to save the resulting audio files and choose the name of those stored files, adding a plot of the audio signal together with the spectrogram one or changing the tab names to make the application more understandable.

5.4 Next steps

The next steps considering the software development are strictly related to the project drawbacks and the possible improvements sections. The first next step should be to find a way to compile Essentia on Windows and also solve the different errors from the PyAudio library execution on macOS, bringing the application capabilities to those operating systems and allowing the developers to enhance the software from all the terminals perspectives. Secondly, developers should find a way to make the application responsive, allowing it to execute the interfaces from different viewports and enhancing the design of the program.

And last but not least, the application could add more windows in order to treat different transformations or audio processing models, enhancing the learning/teaching experience through the usage of the application.

Bibliography

[1] Smith, J. O. (2008). Mathematics of the discrete Fourier transform (DFT): with audio applications. Julius Smith.

[2] Zölzer, U., Amatriain, X., Arfib, D., Bonada, J., De Poli, G., Dutilleux, P., ... & Todoroff, T. (2002). DAFX-Digital audio effects. John Wiley & Sons.

[3] Bogdanov, D., Wack, N., Gómez, E., Gulati, S., Herrera, P., Mayor, O., Roma, G., Salamon, J., Zapata, JR., & Serra, X. (2013). Essentia: an open-source library for sound and music analysis. In Proceedings of the 21st ACM international conference on Multimedia, 855-858. DOI: 10.1145/2502081.2502229

[4] Summerfield, M. (2007). Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming. Pearson Education.

[5] Harwani, B. M. (2018). Qt5 Python GUI Programming Cookbook: Building responsive and powerful cross-platform applications with PyQt. Packt Publishing Ltd.

[6] Meyer, E. A. (2006). CSS: The Definitive Guide: The Definitive Guide. O'Reilly Media, Inc.

[7] Sun, Y., & Vu, K. P. L. (2018). Population stereotypes for color associations. In Engineering Psychology and Cognitive Ergonomics: 15th International Conference, EPCE 2018, Held as Part of HCI International 2018, Las Vegas, NV, USA, July 15-20, 2018, Proceedings 15, 480-489. Springer International Publishing. DOI: 10.1007/978-3-319-91122-9_39

[8] Serra, X., & Smith, J. (1990). Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. Computer Music Journal, 14(4), 12-24. DOI: 10.2307/3680788

[9] Nuttall, A. (1981). Some windows with very good sidelobe behavior. IEEE Transactions on Acoustics, Speech, and Signal Processing, 29(1), 84-91. DOI: 10.1109/TASSP.1981.1163506

[10] Amatriain, X., Bonada, J., Loscos, A., & Serra, X. (2002). Spectral processing. DAFX: Digital Audio Effects, 373-438. DOI: 10.1002/9781119991298.ch10

[11] Quatieri, T. F., & McAulay, R. J. (2002). Audio signal processing based on sinusoidal analysis/synthesis. Applications of digital signal processing to audio and acoustics, 343-416. DOI: 10.1007/0-306-47042-X_9

[12] McAulay, R., & Quatieri, T. (1986). Speech analysis/synthesis based on a sinusoidal representation. IEEE Transactions on Acoustics, Speech, and Signal Processing, 34(4), 744-754. DOI: 10.1109/TASSP.1986.1164910

[13] Macon, M. W. (1996). Speech synthesis based on sinusoidal modeling. Georgia Institute of Technology.

[14] Pankaja, N., & Mukund Raj, P. K. (2013). Proprietary software versus open source software for education. American Journal of Engineering Research, 2(7), 124-130.

[15] Savavibool, N., Gatersleben, B., & Moorapun, C. (2018). The effects of colour in work environment: a systematic review. Asian Journal of Behavioural Studies, 3(13), 149. DOI: 10.21834/ajbes.v3i13.152

Appendix A: Color Scheme of the Application

For the color scheme of the application I decided to use two different palettes, one for the Light theme and another for the Dark theme. The two color schemes are the following ones:

Dark Theme:

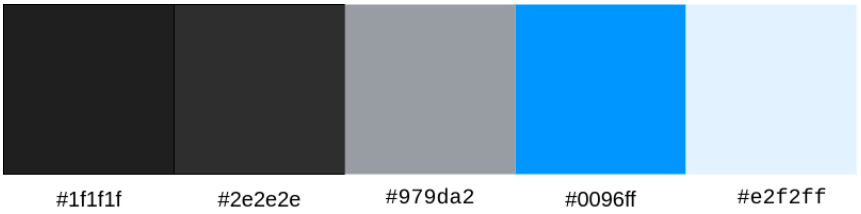


Figure A: Color Scheme of the Application with Dark Theme

Light Theme:

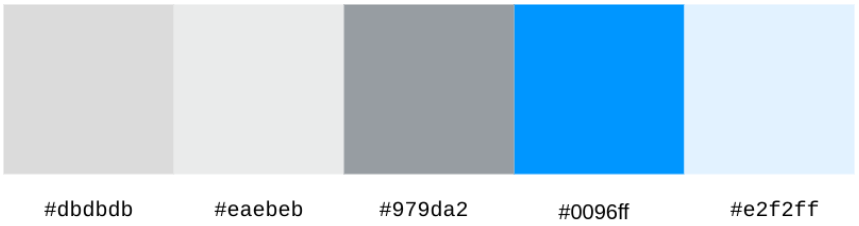


Figure B: Color Scheme of the Application with Light Theme

The reasons why I have chosen these two schemes are simple. Firstly, the colors combined together give a good contrast which can lead to a more satisfactory sensation while working with the software. Secondly, I found that the main tendencies in software development of sound processing applications were applying this kind of style so I decided to recreate some of the current applications in the market. And last but not least, I found a study that ensures that the blue color is related to productivity [15], so the decision of the button's color was not difficult.

As shown on the previous sketches, the theme of the application can be changed without the need of restarting the software. The user can change the color scheme of the application to the one that can fit their interests. This is done by having a simple button at the left bottom of the main window in which we can choose between the light (sun icon) and the dark (moon icon) themes:

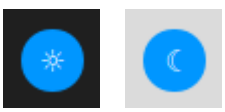
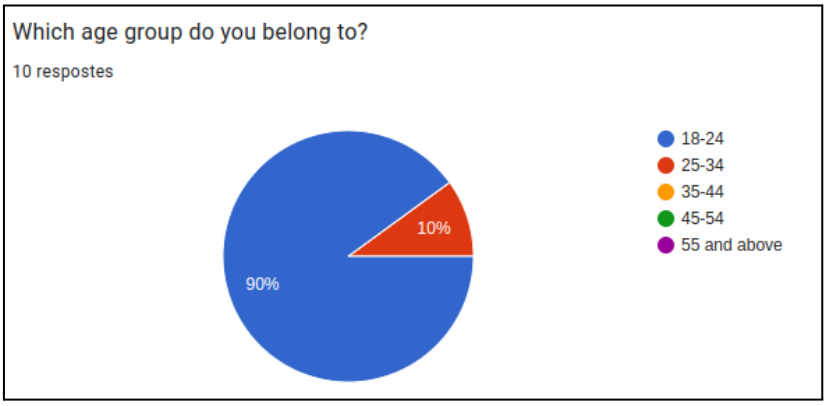
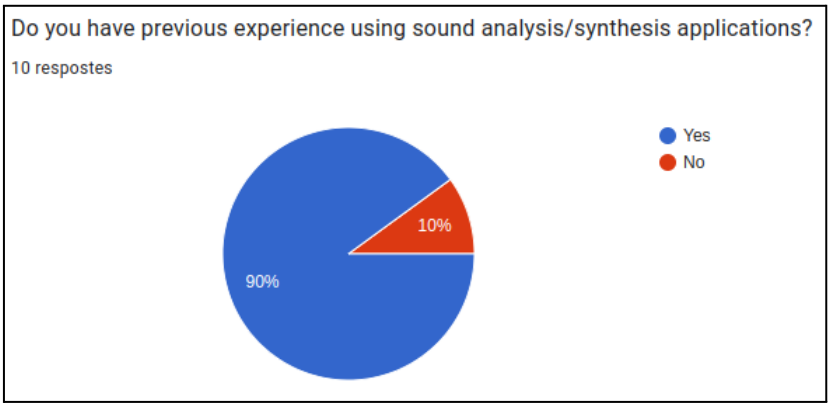


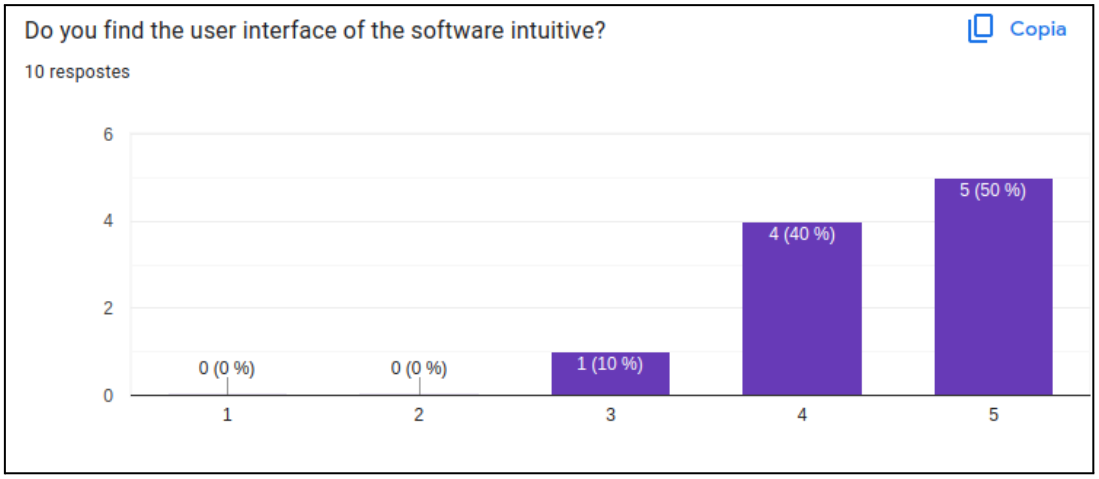
Figure C: Change theme button in the main window

Appendix B: Results of the Software Evaluation

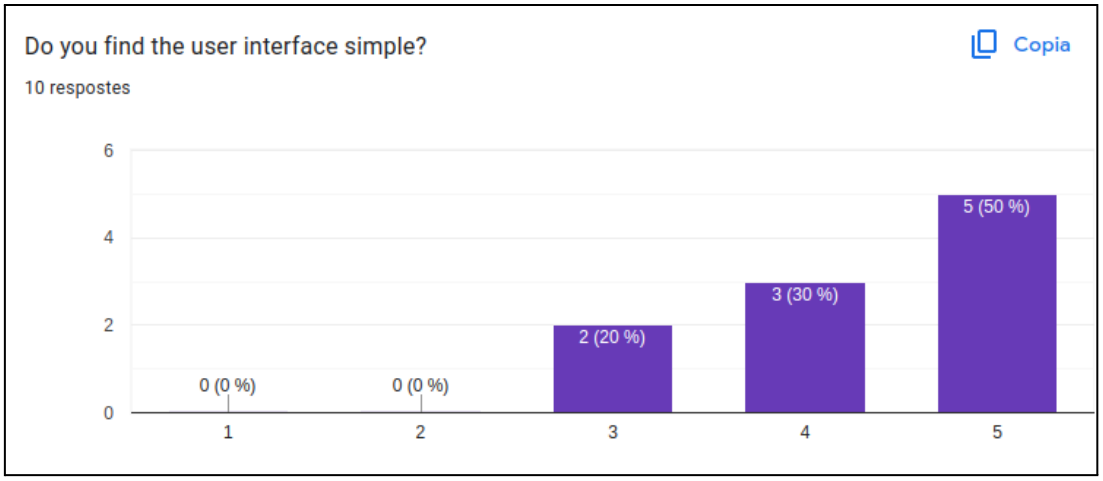
The results obtained in the software evaluation process conducted with different Audiovisual Systems Engineering students are the following ones:



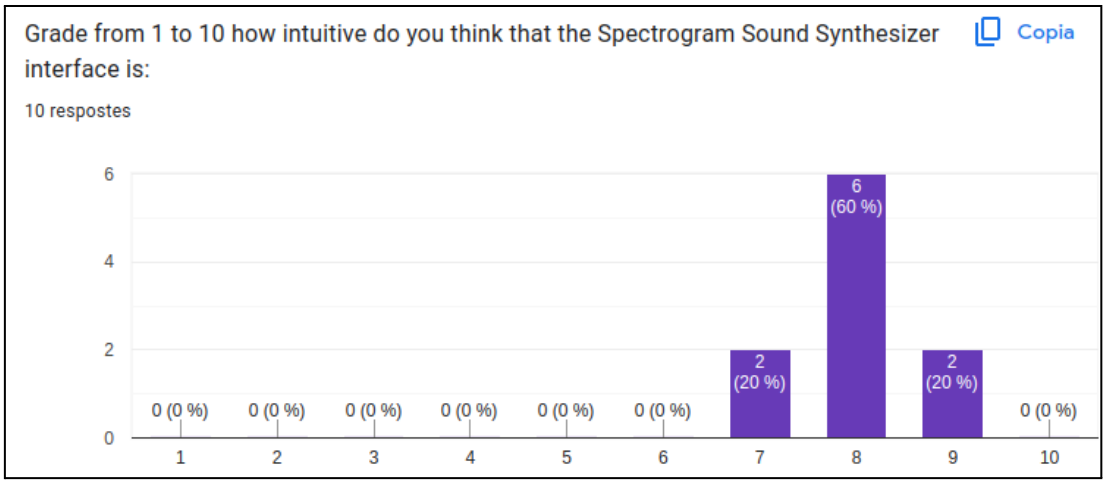


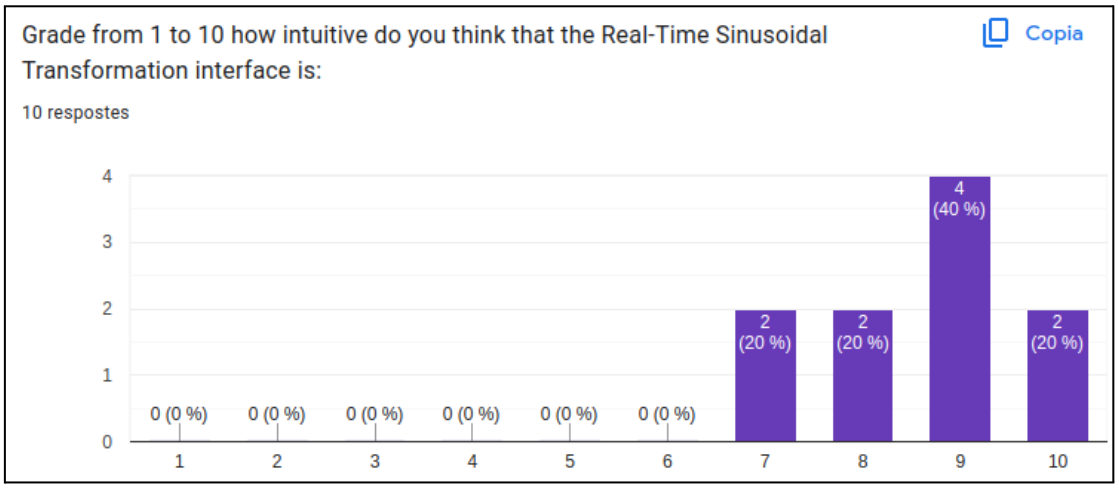


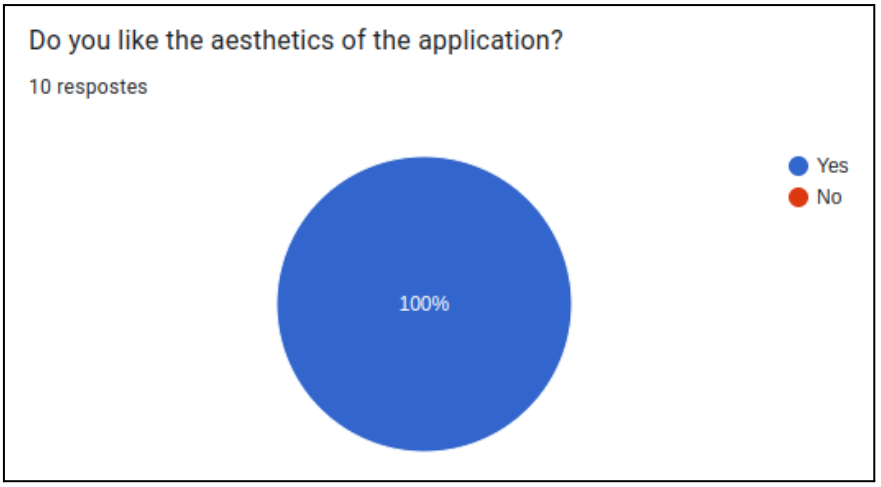
1 - Totally disagree 5 - Totally agree

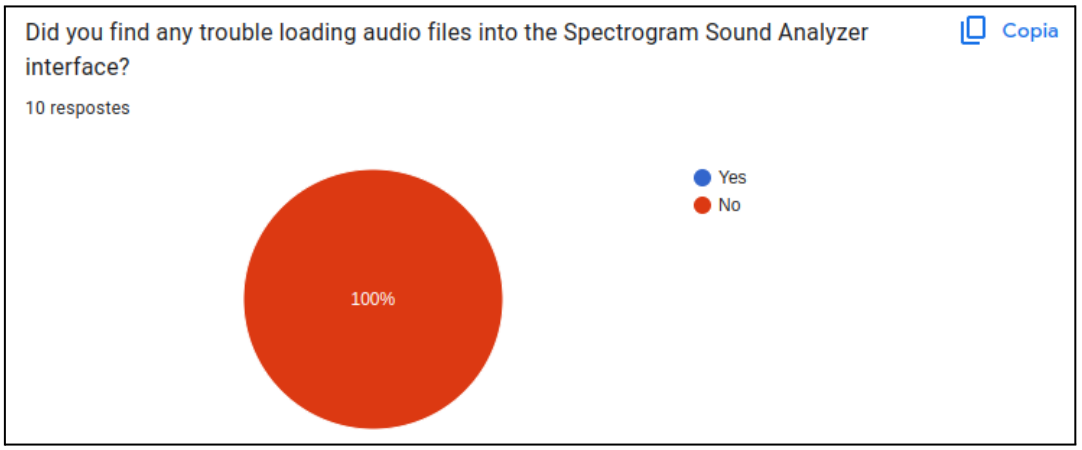


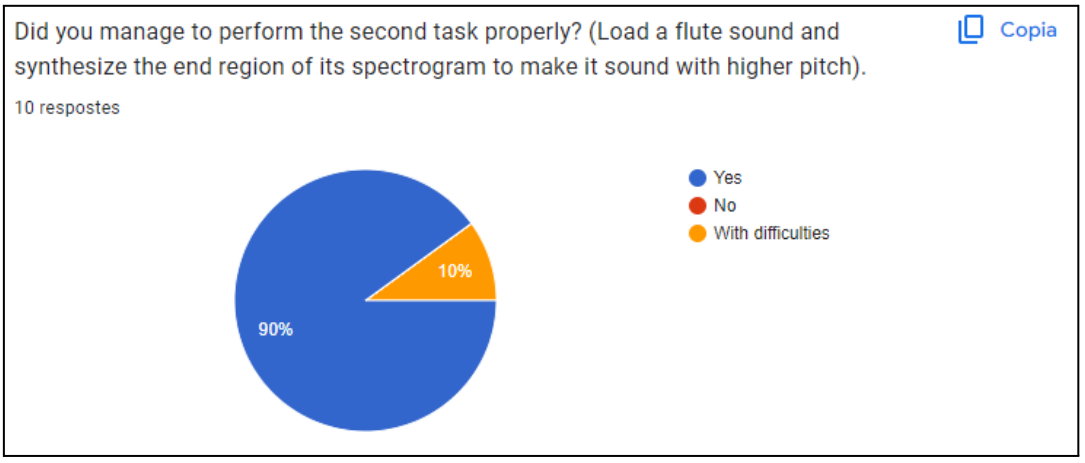
1 - Totally disagree 5 - Totally agree

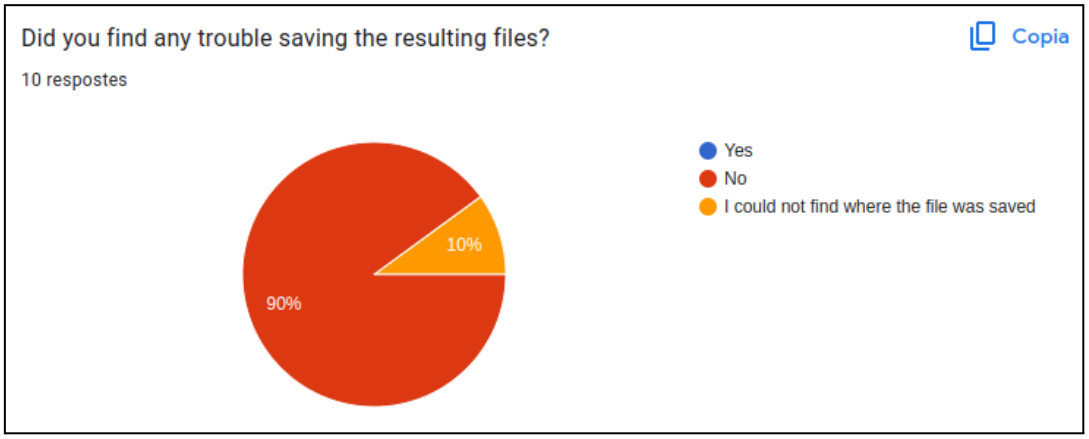


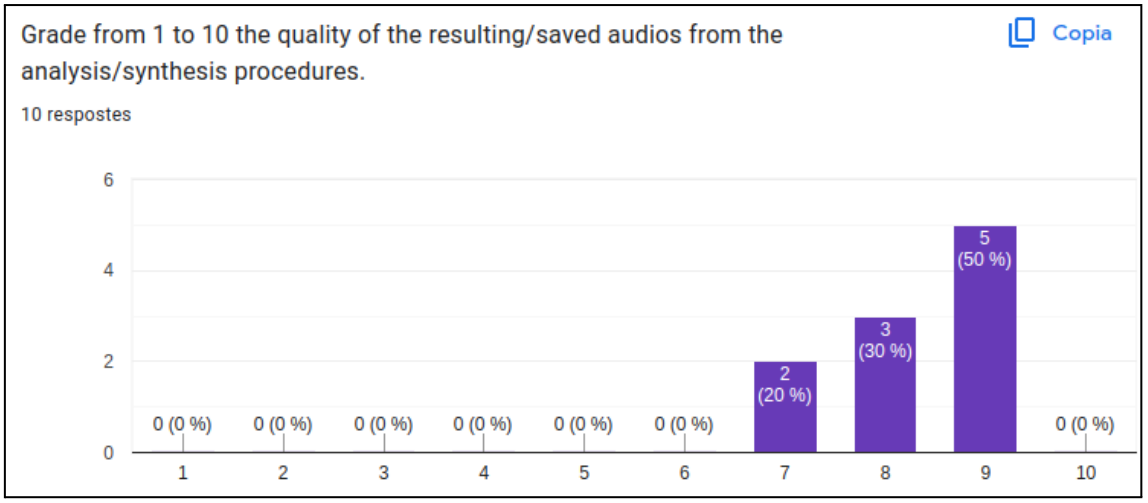


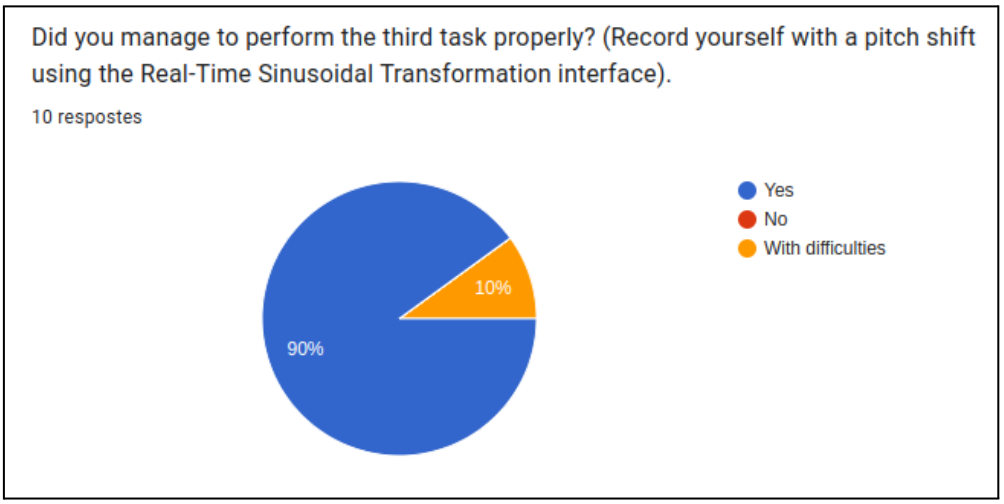

Do you like the aesthetics of the application?
10 respostes
Yes
No
100%

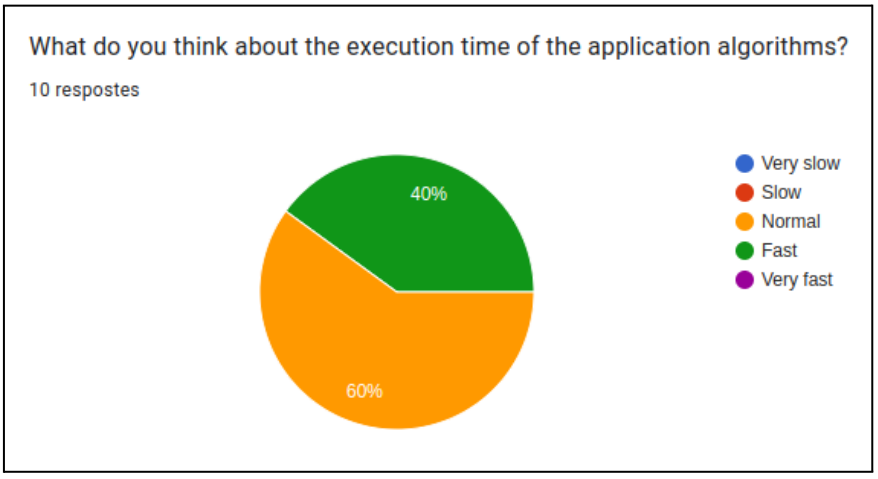

Did you find any trouble loading audio files into the Spectrogram Sound Analyzer interface?
Copia
10 respostes
Yes
No
100%


Did you manage to perform the second task properly? (Load a flute sound and synthesize the end region of its spectrogram to make it sound with higher pitch).
Copia
10 respostes
Yes
No
With difficulties
10%
90%


Did you find any trouble saving the resulting files?
Copia
10 respostes
Yes
No
I could not find where the file was saved
10%
90%

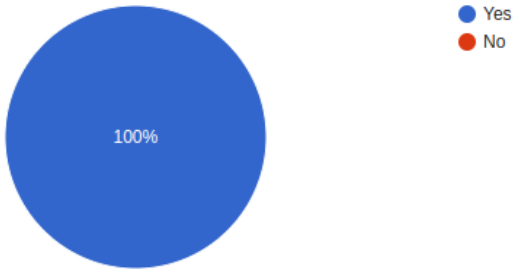

Grade from 1 to 10 the quality of the resulting/saved audios from the analysis/synthesis procedures.
Copia
10 respostes
6
4
2
0
0 (0 %)
0 (0 %)
0 (0 %)
0 (0 %)
0 (0 %)
0 (0 %)
2 (20 %)
3 (30 %)
5 (50 %)
0 (0 %)
1
2
3
4
5
6
7
8
9
10


Did you manage to perform the third task properly? (Record yourself with a pitch shift using the Real-Time Sinusoidal Transformation interface).
10 respostes
Yes
No
With difficulties
10%
90%


What do you think about the execution time of the application algorithms?
10 respostes
Very slow
Slow
Normal
Fast
Very fast
40%
60%

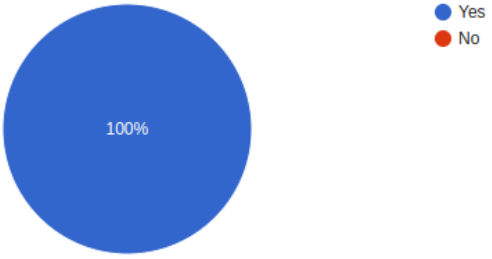
Do you think that the application can help students in order to learn about the Sinusoidal Model in a sound processing basis?

10 respostes



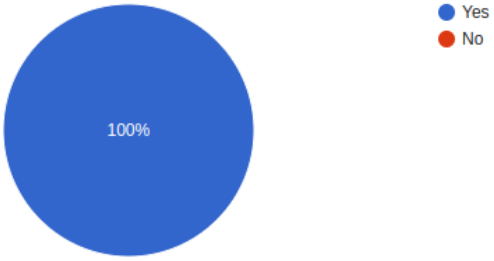
Did you find the different help options useful?

10 respostes



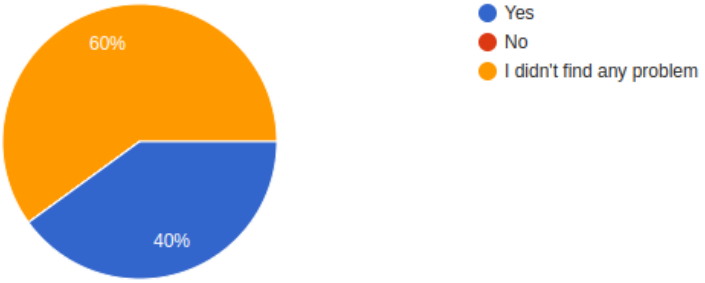
Do you think that the application can help teachers in order to teach about the Sinusoidal Model in a sound processing basis?

10 respostes



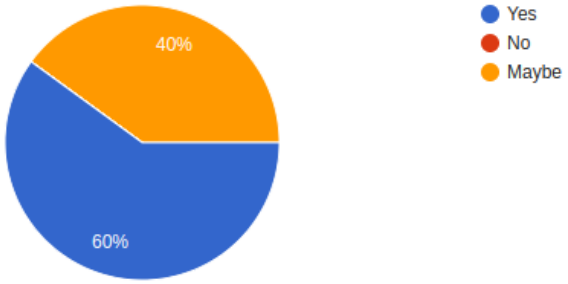
If you have encountered errors/issues, was there any notification from the application to handle it?

10 respostes



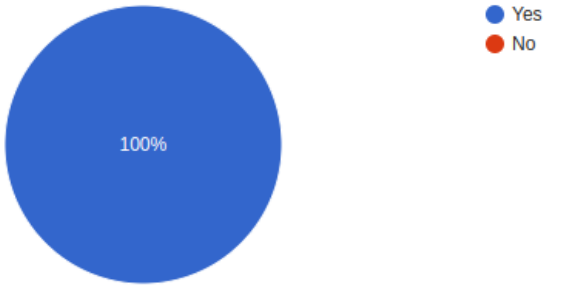
In general terms, would you use the software to learn about the Sinusoidal Model in a sound processing basis?

10 respostes



Would you recommend the software to other students/teachers to learn more about sound processing?

10 respostes



Comment us what would you add in the application to improve it.

10 respostes

I would add the functionality of loading long audios to the Spectrogram Sound Synthesizer

I would allow the users to choose the name of the saved audio files and also select the folder in which they want to store them.

The App runs slower with large sounds, it would be great to improve that.

I don't know where the files are saved, I would like a message or something from the software.

Maybe a plot of the audio together with the spectrogram would help the users

I would add a feature that allows users to change the themes of the application choosing the colors, I would like to change the colors of the plots in the second interface (light theme).

Maybe I would add some information of what is SMSA and RTST for people who aren't used to this terminology

Nothing

Maybe a link to video tutorial or something but everything was pretty clear anyways

State where the file is saved