UNTREF UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Sound Engineering

Practical Work 10: SOFTWARE DEVELOPMENT FOR RIRs PROCESSING

Cohen, Alejo (51134)
de la Vega, Luciano (51941)
Lockey, Ciro (51042)
Rapetti, Agustín (52033)

Abstract: This report depicts the process of development, trial and validation of an acoustic parameters calculation software written in Python. Taking a impulse response as an input the developed software calculates reverberation time, clarity, inter aural cross correlation, transition time and displays the results in a custom made graphical user interface. User can adjust processing options and choose whether to display the results in full-octave or third-octave bands, which can be exported as images or CSV/txt files if needed. The results were compared with commercial software, yielding very similar values, with differences below the just noticeable difference.

Keywords: impulse response, software development, acoustical parameters

1 Introduction

The impulse response is one of the main tools for the acoustic characterization of rooms. Through it, various parameters can be obtained that define the acoustic behavior of the space. There are several software programs available that calculate these parameters, with varying degrees of popularity within the field. This work presents the development of a software that calculates acoustical parameters such as reverberation time, clarity, inter aural cross correlation and transition time. All necessary code is written in Python language, taking advantage of its data processing capabilities and user friendly approach. All results are displayed in a custom made graphical user interface, using a Python compatible library.

2 Theoretical Framework

a Acoustical Parameters

- RT20 and RT30:
  Both parameters describe the elapsed time between the instant when the sound source turns off and the moment when the signal level drop in the room is 60 dB (from -5 dB to -65 dB [1]. The difference lies in the calculation method. For the RT20 parameter, a 20 dB level drop is taken starting from -5 dB and the time it would take for the level to decay to -65 dB is extrapolated, while the RT30 performs the same process but with a 30 dB drop. This is necessary because (with few exceptions) the information recorded in the final part of the decay may be unreliable when the signal level approaches the noise floor.

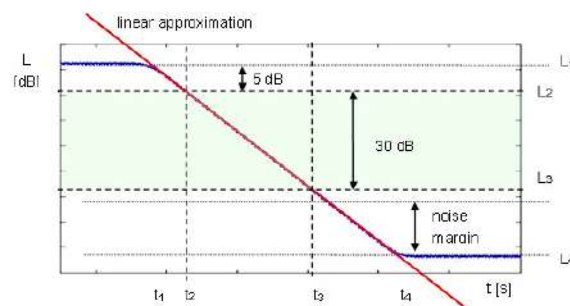


Figure 1: Caption

- Early Decay Time (EDT)
  In the same way as with the RT20 and RT30 parameters, the early decay time depict the level decay time for 60 dB but, in this case, extrapolated taking as reference the slope of the attenuation of the first 10 dB.

- Transition Time (Tt)
  The Transition Time represents the point at which the system behavior shifts from deterministic to stochastic. It is defined as the moment when the accumulated energy of the impulse response reaches 99% of its total value. This parameter can also be interpreted as the effective duration of the early sound field.

- Early Decay Transition Time (EDTt)
  Beyond the EDT calculated from the initial 10 dB decay, an alternative approach to assess early decay is to perform a linear regression between the peak of the impulse response and the transition time. This yields the parameter known as EDTt.

- $IACC_{Early}$:
  The Inter Aural Cross Correlation (IACC) is an psychoacoustic parameter that depict the correlation between perceived stimuli in each ear of the listener. A high IACC value indicate that the listener will not perceive a remarkable stereo sensation because of the high correlation between both signals. In this case, this parameter is used for the early field, thus $IACC_{Early}$ is obtained, quantifing the similarity between the left and right channels during the first 80 ms through cross-correlation within a range of interaural time delays, identifying the maximum value, as defined by ISO 3382 [1]. Its mathematical expression is:

$$IACC_e = \frac{\int_0^{80} h_L(t).h_R(t+\tau)d\tau}{(\int_0^{80} h_L^2(t)dt \int_0^{80} h_R^2 dt)^{\frac{1}{2}}} \tag{1}$$

- C50 y C80:
  The clarity parameters describe the energy ratio between early and late fields, seeking to quantify how distinguishable a sound or message is from the reverberant field generated. Usually, C50 is associated with music intelligibility and C80 with speech intelligibility. Mathematically, they are developed as shown:

$$C_{50} = 10log(\frac{\int_0^{50} p^2(t)dt}{\int_{50}^{\infty} p^2(t)dt})[dB] \tag{2}$$

$$C_{80} = 10log(\frac{\int_0^{80} p^2(t)dt}{\int_{80}^{\infty} p^2(t)dt})[dB] \tag{3}$$

b Reversed IR

To avoid ripple caused by the pass band filters on the acoustical analysis, the option of the reversed IR method exists. This simply consists of reversing the IR, applying the filters, and re-reversing the signal, so now it's forward again. With this technique, the band pass filters' ripple affect the most at the end of the signal instead of the beginning of the impulse, which is crucial to the parameters' calculation. This method, described in the literature, helps reduce distortion caused by narrow-band filtering and improves the reliability of the results [**reversed**].

c Moving Median Filter

The moving median filter (MMF) is a signal processing technique used to average data, removing outliers and smoothing the signal. It works by averaging signal points all over the signal length by the following equation:

$$y_i = \frac{1}{N} \sum_{j=0}^{N-1} x_{i-j} \tag{4}$$

where y_i is the average value in i point, x_{ij} is the signal value in $(i - j)$ and N is the number of averaged points. In IR processing, it is used with the aim of reducing local fluctuations and hence ease the generation of the decay curve.

d Schroeder's Inverted Time Integral

The Schroeder's inverted time integral is a formula that sums the energy from a time t to infinity, returning a descendant curve that reflects the energy fall of the IR. Its mathematical expression is:

$$R(t) = \int_t^{\infty} h^2(\tau)\, d\tau \tag{5}$$

where $R(t)$ is the inverse energy sum and $h(\tau)$ the signal.

e Just Noticeable Difference (JND)

The JND (Just Noticeable Difference) is the smallest variation in a magnitude or stimulus that an observer can detect and perceive as a real change. This threshold is determined through experimental tests and subjective studies, and its value may vary depending on the type of stimulus and the conditions of perception. Table 1 presents typical JND values for various parameters relevant to this study.

Table 1: JND values for the presented parameters.

Parameter	JND
T_{20}	5%
T_{30}	5%
EDT	5%
C_{50}	$1\ dB$
C_{80}	$1\ dB$
$IACC_e$	0.075

3 State of Art

Impulse response (IR) analysis constitutes an essential tool in acoustics engineering, as it allows characterization of sound's behaviour in closed rooms through the study of their response to a controlled stimulus. Over time, various methodologies have been proposed to capture, process, and analyze IRs, including the use of pseudorandom noise and sinusoildal sweeps.

Among the most significant methodological advances, the technique introduced by Manfred Schroeder [2] stands out for estimating the reverberation time from the energy decay curve, which has become in a widely adopted standard.

Later, Angelo Farina [3] introduced a method based on sinusoidal sweeps, which not only allows to obtain the high resolution impulse response and sound to noise ratio (SNR), but also separate in an efficiently way the system's lineal response from the harmonic distortion generated by electronic components used in the measurement. Additionally, works as Lundeby's et al. [4] have adressed measurements uncertainties in rooms, establishing guidelines to increase accuracy and repeatability under different conditions.

More recently, application of artificial intelligence in this field has begun to be explored. Researches such as that of Richard, Dodds and Ithapu has shown that is possible to train deep neural networks to estimate impulse responses from partial or contaminated data, opening up new possibilities for scenarios where direct measurement is impractical or expensive [5].

Another interesting development is by Lan, Zheng, Zhen and Zhao, which introduce the Acoustic Volume Rendering (AVR) method for IRs synthesis in any point of a room through simulation, with results that improves IRs simulations by other programs [6].

In terms of digital tools development, Aurora stands out [7], a suite of plugins for Adobe Audition that allows acoustic analysis using techniques as inverse deconvolution and the Schroeder's curve, being widely used by recording and post-production studios. In contrast, although more limited in technical capabilities, Audacity [8] offers compatibility with convolution effects and external plugins, making it an affordable option for educational environments.

Specialized tools as Room EQ Wizard (REW) [9], ARTA [10] and EASERA [11] offer advanced functions for capturing, analyzing and reporting IRs, integrating octave filters, error compensation algorithms and compliance with international regulations, making them common in both academic and professional fields.

On the other hand, Texture Analyzer [12] software, developed by Alejandro Bidondo, allows the analysis of monaural and binaural IRs to calculate microscopic parameters such as cumulative number of early reflections (CNERs), Tt, EDTt, effective length of autocorrelation (ACd) and other indicators of acoustic texture.

4 Software Development

The programming language used for the development of the software is Python [13]. In addition, ChatGPT [14] was employed to implement various improvements to the code, including optimizations

and enhanced data visualization. Several Python libraries were also used: `numpy`, `pandas`, `scipy`, `tkinter`, `numba`, `matplotlib`, `csv`, and `pillow`.

The software's development process consisted of three major interconnected branches: the implementation of functions for parameter calculation, the development of the program's signal processing code, and the design of the graphical user interface (GUI). Special attention was also given to anticipating potential user or code errors by setting default values for input parameters, managing errors and ensuring that the results are presented clearly and neatly.

a IR Processing

The functions dedicated to the acoustic parameter calculations are located in a separate `.py` file, which is imported into the main file of the graphical user interface. A block diagram illustrating the processing steps for obtaining all the acoustic parameters is presented in Figure 2.

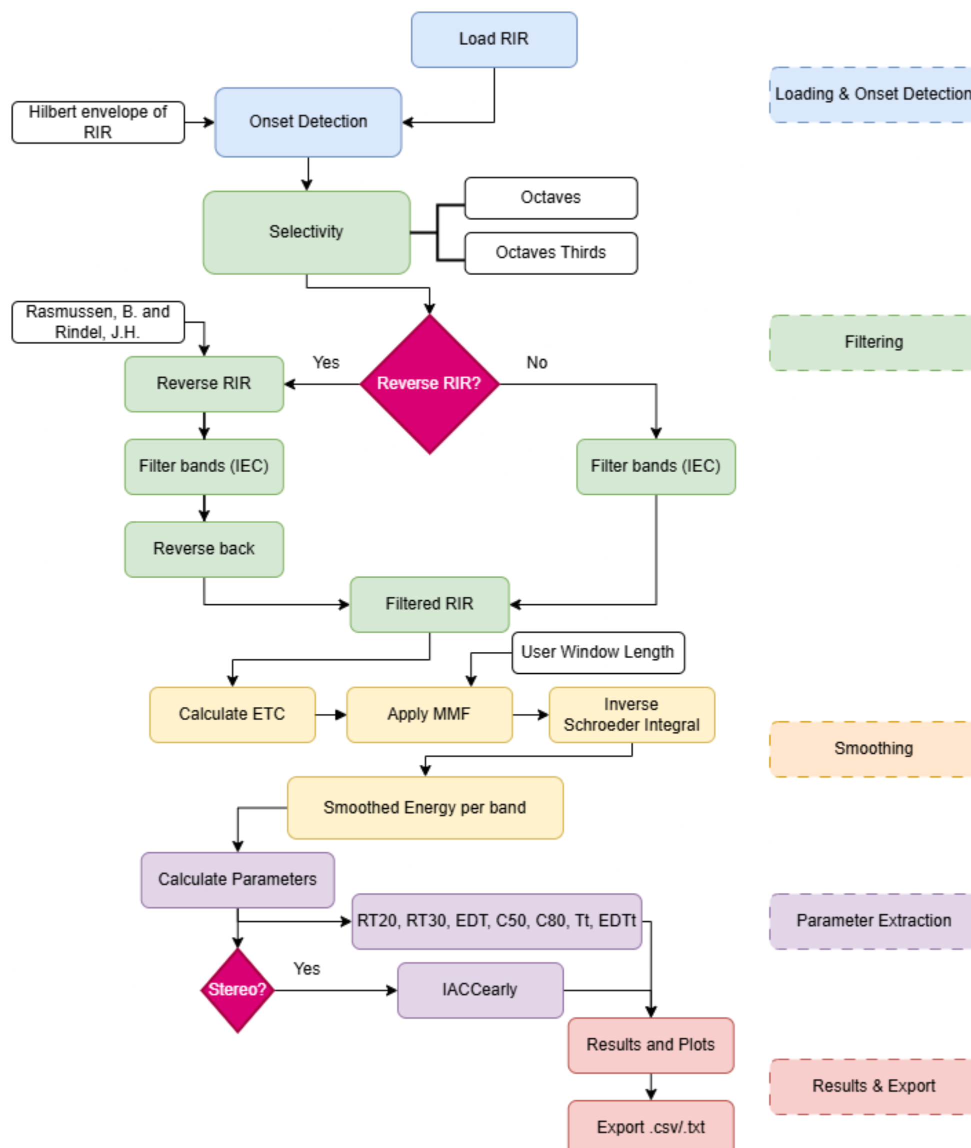


Figure 2: Processing diagram.

The developed code is intended for the analysis of room impulse responses (RIR) to extract standardized acoustic parameters. The processing is structured around a main code that coordinates all the necessary steps to obtain the acoustic parameters (from a function script), based on the input parameters entered by the user through the GUI and their subsequent extraction.

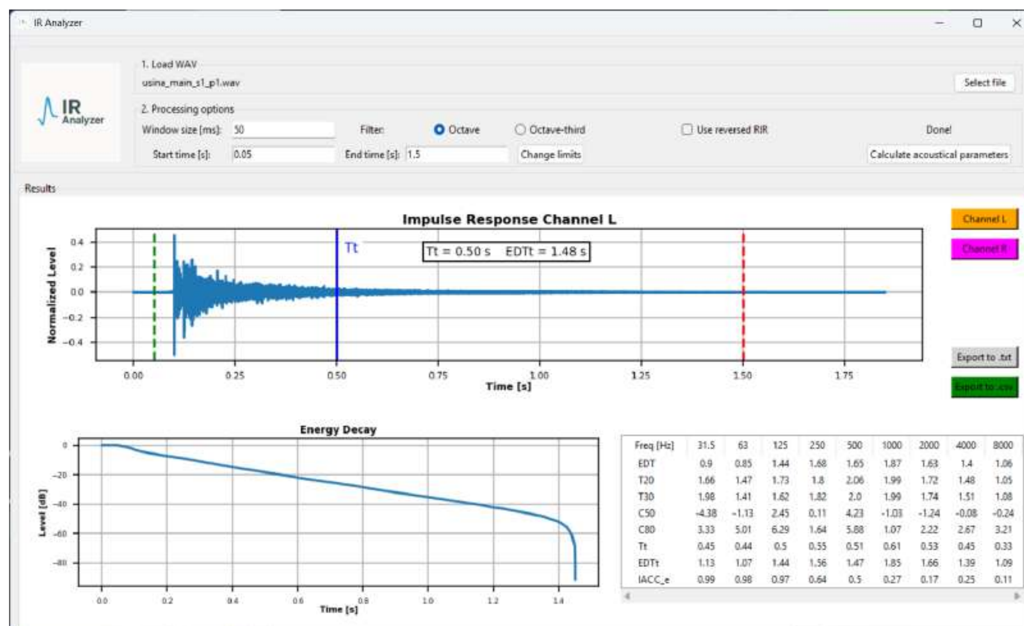
The first step involves identifying the actual onset of the RIR. To achieve this, the point of maximum energy in the signal's envelope —obtained using the Hilbert transform of `scipy`— is detected, and everything preceding is discarded. This ensures that the time origin is aligned with the true beginning of the acoustic excitation, enhancing the accuracy of subsequent temporal analyses.

Once the useful portion of the signal is isolated, it is split into standardized frequency bands —either octave or third-octave— using FIR filters from `scipy`. These filters are designed according to IEC 61260 specifications and applied using fast convolution [15]. In the case of third-octave filtering, a technique known as *'Reversed RIR'* is considered, where the signal is reversed in time before filtering and then flipped back afterwards.

For each filtered band, the Energy-Time Curve (ETC) is computed by squaring the envelope of the signal. This curve is normalized to its peak value to allow relative comparisons of its temporal evolution. A smoothing step is then applied using `scipy`'s Moving Median Filter (MMF), whose window length is user-defined (often set as the period of the lowest frequency of interest; by default 50 ms, which its 20 Hz). The smoothed ETC is integrated backwards using the Schroeder integral, generating a decay curve in decibels. This decay profile is used to estimate the main reverberation times. Thereby linear regressions are performed, with `linregress` from `scipy.stats`, over specific intervals to extrapolate the time required for a 60 dB drop, yielding the T_{20}, T_{30}, and EDT values respectively. Clarity parameters, such as $C50$ and C_{80}, are also calculated by comparing the ratio of energy within the first 50 ms or 80 ms to the remaining energy, obtained by applying `numpy` functions. The transition time is obtained from the cumulative energy of the RIR, where it reaches 99% of its total. Within this interval, a linear decay is estimated in the dB domain, from which EDT_t is obtained. For each frequency band, the $IACC_{early}$ is computed by filtering the truncated stereo IR and then evaluating the maximum cross-correlation between the left and right channels within the first 80 ms, considering interaural delays of 1 ms. All calculated parameters are organized by frequency band and compiled into a structured matrix, with rounded values to ease interpretation.

b Graphical User Interface (GUI)

In Figure 3 is shown the window of the IR Analyzer software.



Freq [Hz]	31.5	63	125	250	500	1000	2000	4000	8000
EDT	0.9	0.85	1.44	1.68	1.65	1.87	1.63	1.4	1.06
T20	1.66	1.47	1.73	1.8	2.06	1.99	1.72	1.48	1.05
T30	1.98	1.41	1.62	1.82	2.0	1.99	1.74	1.51	1.08
C50	-4.38	-1.13	2.45	0.11	4.23	-1.03	-1.24	-0.08	-0.24
C80	3.33	5.01	6.29	1.64	5.88	1.07	2.22	2.67	3.21
Tt	0.45	0.44	0.5	0.55	0.51	0.61	0.53	0.45	0.33
EDTt	1.13	1.07	1.44	1.56	1.47	1.85	1.66	1.39	1.09
IACC_e	0.99	0.98	0.97	0.64	0.5	0.27	0.17	0.25	0.11

Figure 3: Software's graphical interface.

By clicking the *'Load File'* button, a `.wav` file containing the impulse response (IR) is loaded by user. The program automatically detects whether the file is mono or stereo. A plot of the loaded audio is

displayed below (if the file is stereo, the left channel is shown by default).

The software includes various processing options. The user can specify the MMF window size in milliseconds, choose between octave or third-octave bands, enable or disable the use of a reversed IR, and select the time limits of the audio file. This is particularly useful for files that may contain multiple responses or noise segments that could affect the results.

When pressing *'Calculate Parameters'*, the corresponding calculations are performed according to the specified conditions. A message is displayed indicating that the calculation is in progress, and once completed, a success message confirms that the parameters have been calculated.

At the bottom left, the signal after applying Schroeder's inverse integration can be visualized, while on the right side, a table displays all parameters by frequency band. Channel left or right can be switched using the associated buttons. If a parameter row in the table is clicked, the graph on the left updates to show a bar plot of that parameter as a function of frequency. On the IR graph, a blue vertical line is drawn to indicate the signal's transition time, and the Tt and EDTt global values can be visualized.

By right-clicking on one of the plots, a context menu appears. It allows the graph to be saved as a `.PNG` file. To save the lower plot, the user must click on *Save Energy Decay curve or Parameter graph*, and to save the impulse response plot, on *Save IR graph*.

Lastly, the results can be exported to a `.txt` or `.csv` file for documentation, visualization, or further comparison of the acoustic characteristics of the analyzed space.

c Error Messages

To prevent the application from failure or crashing due to incorrect input data, it displays several alerts when any of the input parameters are invalid. If a negative window size is entered, the program notifies the user of the error and urges them to change its value. If the selected impulse limits exceed those of the original signal (such as a negative start time or an end time longer than the audio length), an error message is displayed. Additionally, if a very short impulse duration is selected, a warning is shown indicating that the results may not be representative of the signal, although the calculation is still performed. If the user clicks *'Save Energy Decay curve or Parameter graph'* and there is none, a warning appears. Lastly, if the *'Calculate Parameters'* button is pressed without having loaded a file, a message appears informing the user that no file has been loaded.

d Optimization

Early code versions took a considerate amount of time to process, around 100 seconds. Following several optimization improvements, processing time was significantly reduced to around 10 seconds. Main optimization techniques include the utilization of Numba and ThreadPool libraries. Numba allows for low-level language execution hence reducing each commands processing time, while ThreadPool enables multi-thread processing which means several orders can be executed in parallel.

e Validation - Comparison with Alternative Softwares

In order to validate the obtained results, a comparison was made between them and two widely available acoustic processing softwares: Aurora and Easera. For this purpose, a stereo room impulse response from the OpenAIR database was used —specifically, the file *usina_main_s1_p1*, which corresponds to a measurement taken at Usina del Arte Auditorium [16]. The RIR was processed using a 50 ms window and without applying the reversed RIR technique.

5 Results and Analysis

a Acoustical Parameters

For the Usina del Arte's impulse response, comparisons between C50, C80, EDT, T20, T30, and IACC values are shown in Figures 4, 5, 6, 7, 8, and 9, along with their corresponding percentage differences.

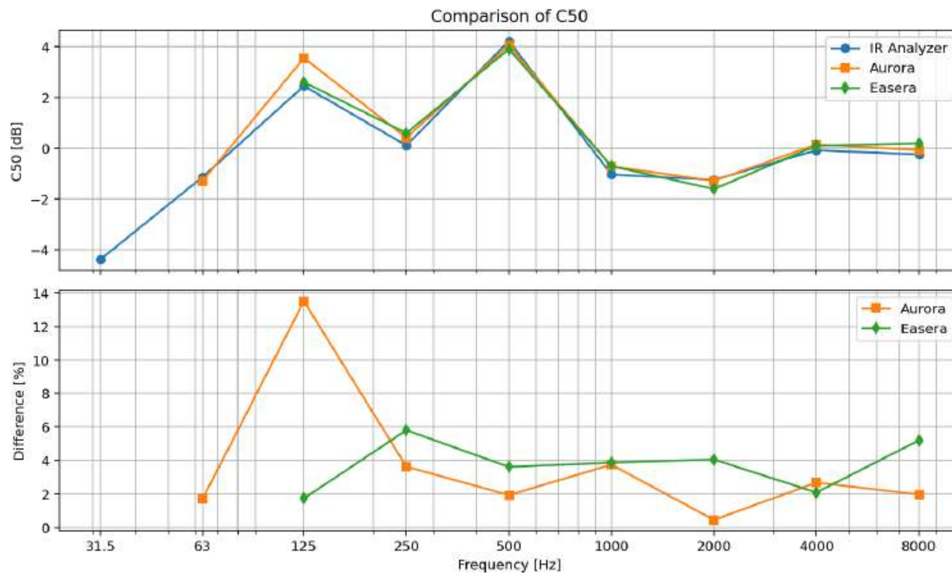


Figure 4: C50 comparison.

For the parameter C_{50} (Figure 4), differences of less than 6% were obtained for almost all bands, except for the 125 Hz band (13.4%). Overall, the results show that the percentage difference tends to stabilize with frequency.

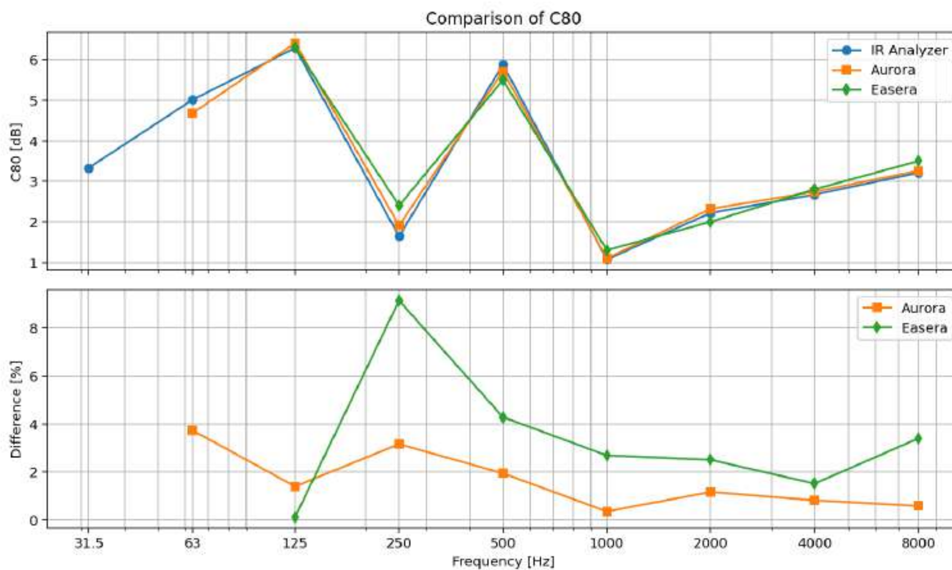


Figure 5: C80 comparison.

For the C_{80} parameter (Figure 5), the percentage differences remain below 5% in all cases, except for the 250 Hz band when compared with Easera (9.1%). Similar to the C_{50} results, a stabilization trend is observed with increasing frequency. In general, for the Clarity parameters the results show smaller differences when compared with Auro ra than with Easera. Moreover, the two tested software tools also show differences between each other in the mentioned bands.

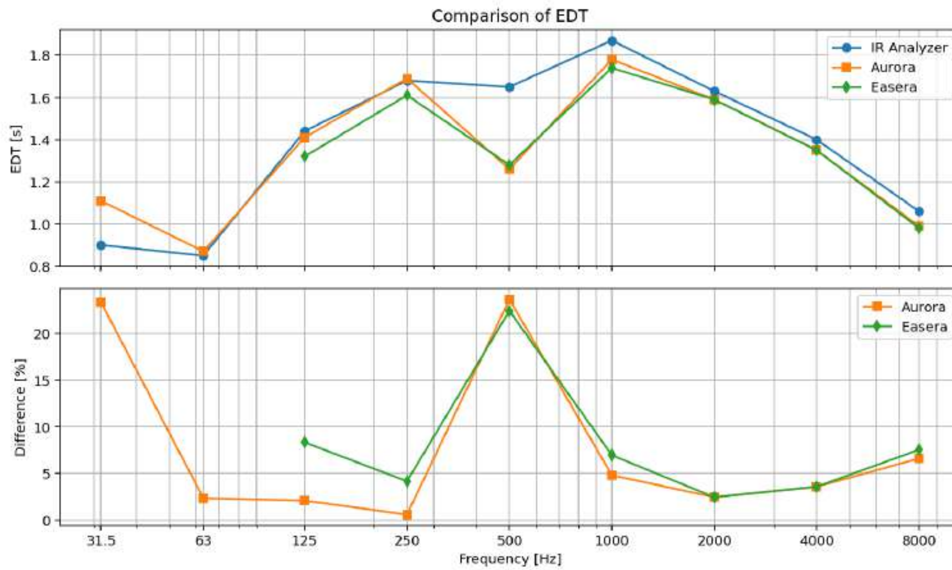


Figure 6: EDT comparison.

In the case of the EDT parameter (Figure 6), the differences with both reference software packages are generally similar. However, a significant rise in deviation is observed in the 500 Hz band, with values reaching around 30%, which clearly differs from the behavior observed in the other bands.

Using the same signal with a much smaller window size (5 ms instead of 50 ms), a value of $1.25\ s$ was obtained in that frequency band, with a difference of less than 3% for both softwares. Therefore, this outlier can be attributed to the use of the moving median filter. Regarding the use of filtering in the other apps, the Aurora manual does not mention any smoothing stage [7] while Easera includes one, although it is not easily accessible to the user. As a result, it is likely that the average user leaves the window size at its default value (35 ms) [11].

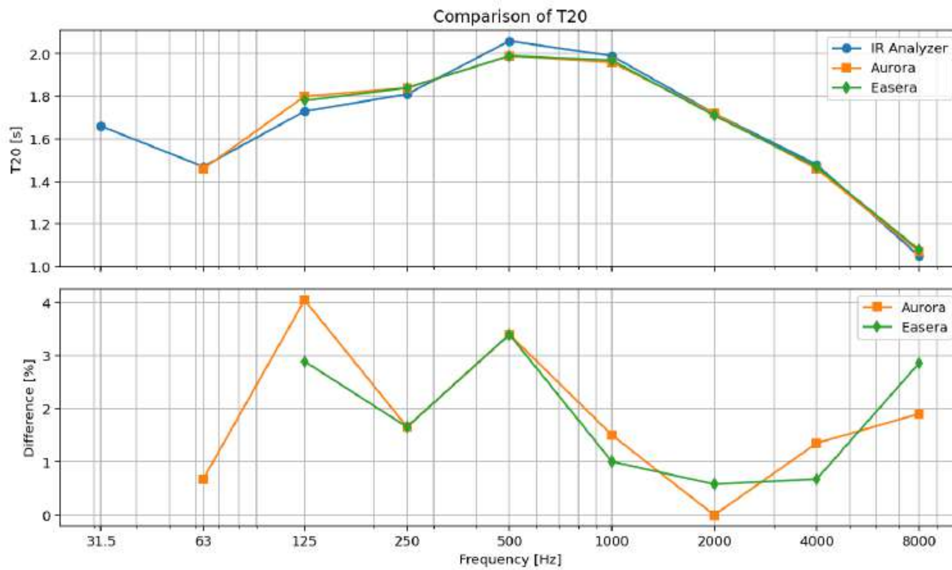


Figure 7: T20 comparison.

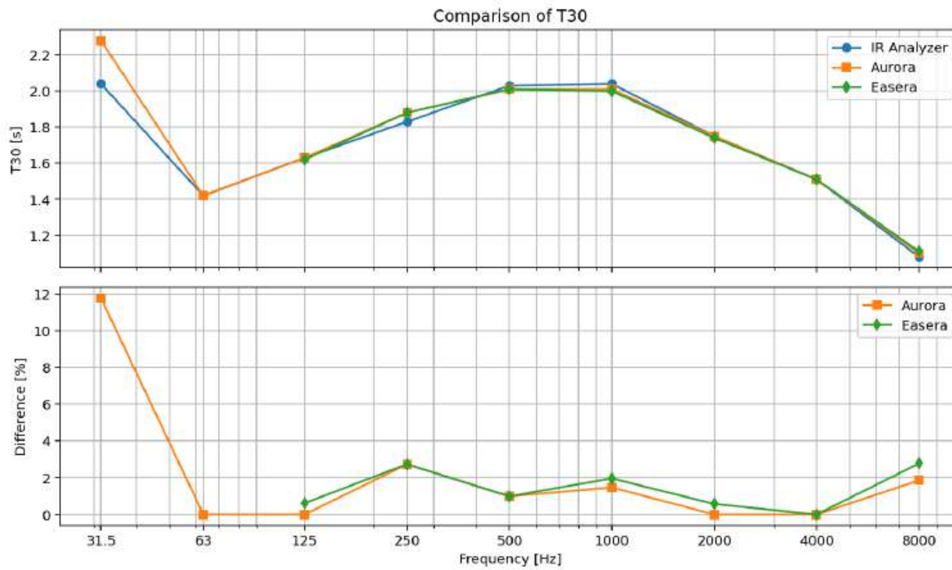


Figure 8: T30 comparison.

For the T_{20} and T_{30} parameters (Figures 7 and 8), the differences do not exceed 4% in general, except for the 31.5 Hz band, where the T_{30} shows a deviation of 10.4%. It can be observed that results obtained for each parameter are similar, although greater deviations are evident at low frequencies. In the case of T_{20}, the comparison with Aurora showed a deviation greater than 40%, but this value was excluded from the graph to improve its readability. This difference may be due to differences in decay onset detection or to the limited effectiveness of the octave-band filtering at that frequency, both of which can significantly affect the accuracy of T_{20} and T_{30} estimations.

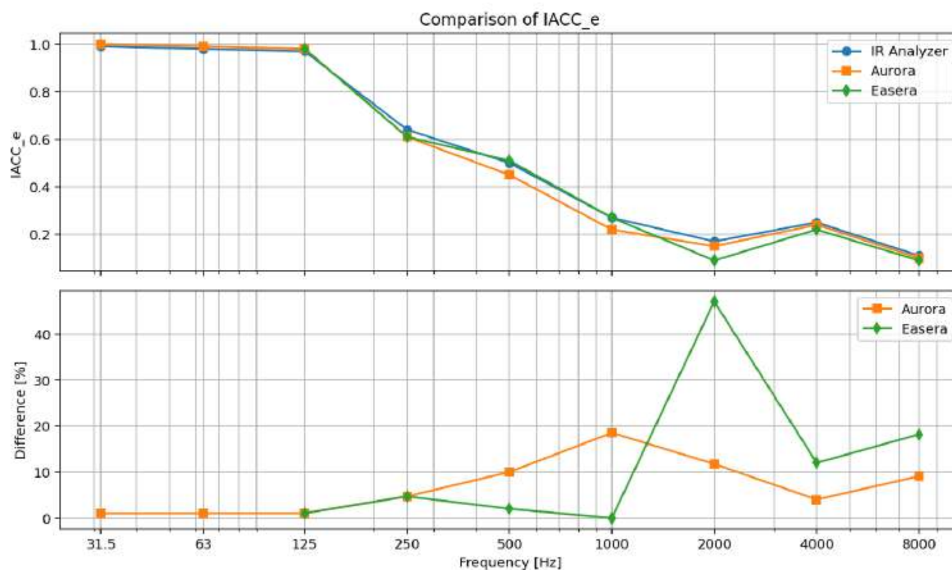


Figure 9: IACCearly comparison.

For the $IACC_e$ parameter (Figure 9), contrary to the others, the results show smaller differences at low frequencies. This can be explained by the sound source behavior being more omnidirectional in this range, resulting in minimal differences between the left and right channels, due to the longer wavelengths. Since all of the software perform a similar procedure regarding the correlation between channels, it is logical that the obtained differences are minimal. Regarding the behavior at high frequencies, a significant deviation is observed at 2 kHz with Easera. Nevertheless, the high percentage

difference can be explained by the low IACC values in this band, whereby a small variation results in a large relative difference.

In Figure 10 is shown the T_t and EDT_t by frequency bands. There is no software available to compare these frequency-specific values. Results were evaluated by comparison with the Texture Analyzer software, which calculates a global value.

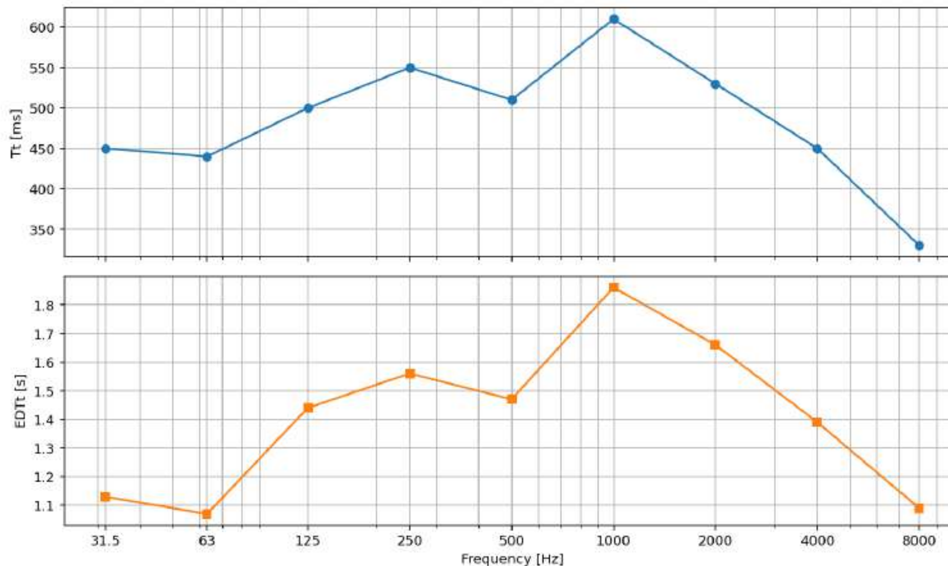


Figure 10: Tt and EDTt values.

Table 2 presents all parameters global values compared with the reference software. EDT, T_{20}, T_{30} and $IACC_e$ are compared against Aurora (because its inclusion of the lower frequency bands) , while T_t and EDT_t are compared against Texture Analyzer.

Table 2: Acoustic parameters comparison.

Parameter	EDT [s]	T20 [s]	T30 [s]	C50 [dB]	C80 [dB]	Tt [ms]	EDTt [s]	IACCe
IR Analyzer	1.39	1.66	1.70	-0.15	3.48	500	1.48	0.54
Other	1.34	1.66	1.73	0.60	3.52	451	1.49	0.53
Difference	3.58%	0.1%	1.65%	0.75	0.04	10.77%	0.34%	0.016
JND	5%	5%	5%	1 dB	1 dB	-	5%	0.075

It can be observed that the difference in all parameters is smaller than the Just Noticeable Difference (JND). This perceptual threshold evaluates, as its name suggests, the minimum change noticeable by a person, and it is particularly useful when comparing results [1]. IR Analyzer provides results that do not exceed the JND, which means the obtained values can be considered very close to those of commercial software.

In the case of T_t, no specific JND study is known. However, center time (T_s) can be used instead, as this parameter represents the energy center of gravity of the impulse response and is often taken as an approximation of the transition time. Therefore, it is the most appropriate JND for this comparison, with a value of 10 ms [17]. In this case, the difference exceeds this value by approximately a factor of five. However, it's important to note that this is a different parameter JND, and there is no other available software that calculates the transition time for its comparison.

b Reversed RIR and Window Size

The RIR from the Usina, analyzed in the previous section, was processed in third-octave bands both with and without time reversal. The results were virtually identical in both cases. Subsequently, a shorter

RIR from the OpenAIR database -specifically, the recording from Koli National Park (Winter) [18]- was tested under the same conditions, yielding no observable differences either.

Step-by-step plotting of the processing pipeline confirms that the RIR is indeed reversed; however, no evidence of filter ringing extending towards the right (i.e., forward in time) is observed. This suggests that the reversal may only become relevant in environments with very short reverberation times—such as control rooms or anechoic chambers—where filter ringing could become significant in relation to the overall decay, particularly at low frequencies.

Figure 11 shows EDT, T_{20}, and T_{30} parameters for different moving median filter window sizes.

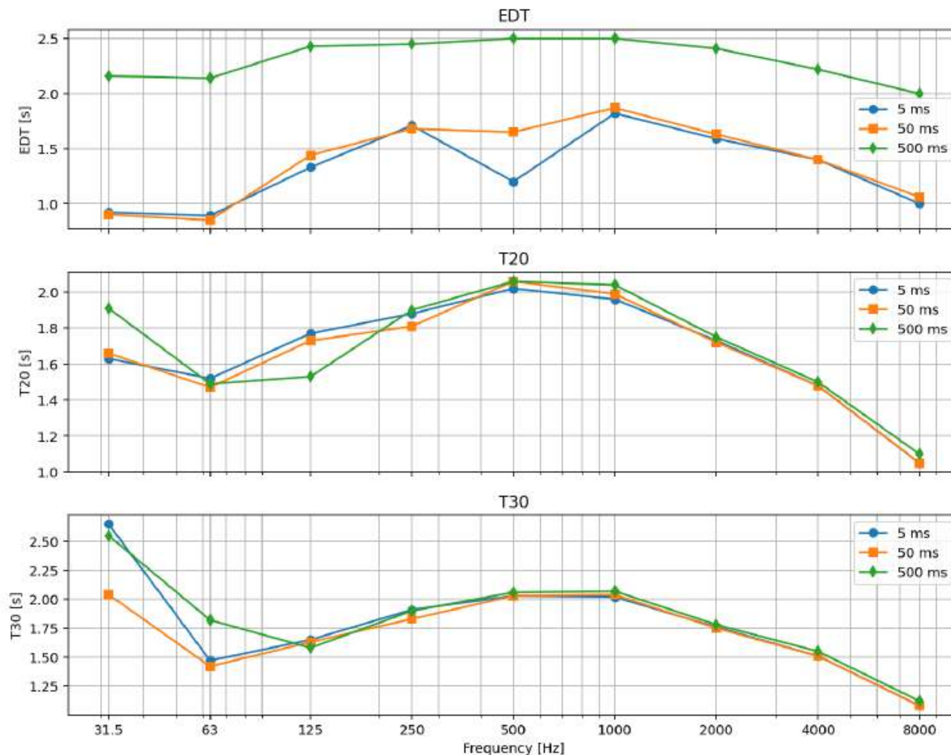


Figure 11: EDT, T20 and T30 for different MMF window sizes.

The results show that the size of the median filter window has little to no significant impact on the T20 and T30 parameters. In general, values remain consistent across the three tested window sizes, with some localized differences at low frequencies—most notably in the 31.5 Hz band—where estimates show greater variability.

In contrast, the EDT parameter exhibits a clear dependence on the window size. In particular, the largest window (500 ms) produces consistently higher values across the spectrum. This could be explained by excessive smoothing masking the initial decay in the energy curve, resulting in a linear regression over a shallower slope, and therefore, a higher EDT value.

Notably, the EDT at 500 Hz using a $5_m s$ window closely matches the result obtained previously with Aurora and Easera (approximately 1.2 s), suggesting that those software tools may be applying relatively short smoothing windows in their EDT estimation.

6 Conclusions

The developed application yields results very close to those of industry-standard software, with differences below the just noticeable difference. It was found that the window size of the moving median filter can significantly influence the results on low frequencies and when outliers are present in the signal. This also suggests that the compared software tools either do not apply smoothing or use a much shorter window than the one implemented in IR Analyzer for the analysis carried out in this report.

The analysis of the transition time presents a particular case, as only one software tool estimates it, and only as a global value. Therefore, it is not possible to perform a precise frequency-based comparison, nor to compare across multiple applications, which may also differ from one another. This limitation also applies to EDT_t.

Looking ahead, there is the possibility that users may not need to have Python installed on their computers to run the application. Among the options being considered are the use of an installer that packages all the necessary files and creates a desktop shortcut, as well as an online implementation hosted on a web platform such as Streamlit [19].

Future improvements could also include extending the code to allow loading a logarithmic sine sweep (LSS) together with its corresponding inverse filter, enabling deconvolution and impulse response extraction directly within the application. Additionally, implementing noise compensation algorithms such as the method proposed by Lundeby et al. could enhance the robustness and reliability of the estimated acoustic parameters, particularly in low signal-to-noise ratio conditions.

References

1. INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Acoustics - Measurement of room acoustic parameters*. 2009. ISO 3382.
2. SCHROEDER, M. *New Method of Measuring Reverberation Time*. Bell Telephone Laboratories Inc., 1964.
3. FARINA, A. Advancements in impulse response measurement by sine sweeps. *AES*. 2007.
4. LUNDEBY, A. et al. Uncertainties of Measurements in Room Acoustics. *Acustica*. 1995, vol. 81.
5. RICHARD, A. et al. *Deep impulse responses: estimating and parameterizing filters with deep networks* [https://arxiv.org/pdf/2202.03416]. 2022. IEEE.
6. LAN, X. et al. *Acoustic Volume Rendering for Neural Impulse Response Fields* [https://www.researchgate.net/publication/385722061_Acoustic_Volume_Rendering_for_Neural_Impulse_Response_Fields]. 2024.
7. AURORA PLUGINS. *Aurora Plugins Website* [http://pcfarina.eng.unipr.it/Aurora_XP/index.htm]. [N.d.].
8. AUDACITY TEAM. *Audacity Website* [https://www.audacityteam.org/]. [N.d.].
9. ROOM EQ WIZARD. *Room EQ Wizard Website* [https://www.roomeqwizard.com/]. [N.d.].
10. ARTA SOFTWARE. *ARTA Software Website* [https://artalabs.hr/]. [N.d.].
11. AFMG. *AFMG Easera Website* [https://www.afmg.eu/en/afmg-easera]. [N.d.].
12. BIDONDO, Alejandro. *Texture Analyzer v15*. [N.d.].
13. PYTHON. *Python 3.6* [www.python.org/]. [N.d.].
14. OPENAI. *ChatGPT* [chatgpt.com]. [N.d.].
15. COMMISSION, International Electrotechnical. *Electroacoustics - Octave-band and fractional-octave-band filters - Part 1: Specifications*. 2014. IEC 61260.
16. OPENAIR. *Open Acoustic Impulse Response* [https://www.openair.hosted.york.ac.uk/?page_id=770]. [N.d.].
17. ŢOPA, M. Experimental Acoustic Evaluation of an Auditorium. *Joseph CS Lai* [https://onlinelibrary.wiley.com/doi/10.1155/2012/868247]. 2012.
18. OPENAIR. *Open Acoustic Impulse Response (IR) Library*. 2023. Available also from: https://www.openair.hosted.york.ac.uk/?page_id=584. Accessed: 2025-07-03.
19. STREAMLIT. *Streamlit Website* [streamlit.io]. [N.d.].

7 Annex

a Python Code

i GUI.py

```
1  # -*- coding: utf-8 -*-
2
3  import tkinter as tk
4  from tkinter import filedialog, ttk
5  import numpy as np
6  from scipy.io import wavfile
7  import matplotlib.pyplot as plt
8  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
9  plt.ioff()
```

```
10 import random
11 import csv
12 from PIL import Image, ImageTk
13 import tkinter.messagebox as msgbox
14 import ctypes
15
16
17 #.py WITH ACOUSTIC CALCULATIONS FUNCTIONS
18 import procesamiento_rir as proc
19
20
21 #DEFINING CLASS
22 class IRAnalyzerApp:
23     def __init__(self, root):
24
25         # --- Initial values ---
26         self.root = root
27         self.root.title("IR Analyzer")
28         self.root.geometry("1100x680")
29         self.root.iconbitmap("logo_icono2.ico")
30
31         # Logo in the window icon
32         ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(u"IRAnalyzer")
33         self.root.iconbitmap("logo_icono2.ico")
34
35         self.fs = None
36         self.signal = None
37         self.label_nofile = None
38         self.stereo_signal = None
39         self.calculated_parameters = None
40         self.tt_global,self.edtt_global=None,None
41         self.tt,self.edtt = None,None
42
43         # Contextual menu to save image
44         self.popup_menu = tk.Menu(self.root, tearoff=0)
45         self.popup_menu.add_command(label="Save IR graph", command=self.save_IR_image)
46         self.popup_menu.add_command(label="Save Energy Decay or Parameter graph", comman
47
48
49         #self.calculated = 0 at the beginning and = 1 after parameters are calculated.
50         # It's used for not drawing Schroeder without parameters calculated.
51         self.calculated = 0
52
53
54         # Frame file 1 and 2
55         self.frame_12 = ttk.LabelFrame(root)
56         self.frame_12.pack(fill="both", expand=True, padx=5, pady=5)
57
58         # --- LOGO ---
59
60         # Loading image
61 #          self.image_original = Image.open("logoIR.png")
62 #          self.image_redim = self.image_original.resize((120, 120), Image.ANTIALIAS)
63 #          self.image_tk = ImageTk.PhotoImage(self.image_redim)
64
65         self.image_original = Image.open("logoIR.png")
66         self.image_redim = self.image_original.resize((120, 120), Image.Resampling.LANCZ
67         self.image_tk = ImageTk.PhotoImage(self.image_redim)
```

```
68
69
70         # Image label
71         self.label_image = tk.Label(self.frame_12, image=self.image_tk)
72         self.label_image.image = self.image_tk  # save it so it doesn't get deleted by g
73         self.label_image.pack(side='left', padx=5, pady=5)
74
75
76
77         # --- Load WAV file ---
78
79         #load wav frame
80         self.frame_file = ttk.LabelFrame(self.frame_12, text="1. Load WAV")
81         self.frame_file.pack(fill="x", padx=10, pady=5)
82
83
84         #Text
85         self.label_file = ttk.Label(self.frame_file, text="No file selected")
86         self.label_file.pack(side="left", padx=5)
87
88         #Button
89         self.btn_load = ttk.Button(self.frame_file, text="Select file", command=self.loa
90         self.btn_load.pack(side="right", padx=5)
91
92
93
94         # --- Processing options ---
95
96         # processing options frame
97         self.frame_options = ttk.LabelFrame(self.frame_12, text="2. Processing options")
98         self.frame_options.pack(fill="x", padx=10, pady=5)
99         self.frame_options.columnconfigure(5, weight=1)
100
101
102        # Window size
103        ttk.Label(self.frame_options, text="Window size [ms]:").grid(row=0, column=0, pa
104        self.entry_window = ttk.Entry(self.frame_options)
105        self.entry_window.insert(0, "50")
106        self.entry_window.grid(row=0, column=1, padx=5, pady=5)
107
108        # Select type of bands
109        self.filter_var = tk.StringVar(value="octave")
110        ttk.Label(self.frame_options, text="Filter:").grid(row=0, column=2, padx=5)
111        ttk.Radiobutton(self.frame_options, text="Octave", variable=self.filter_var, val
112        ttk.Radiobutton(self.frame_options, text="Octave-third", variable=self.filter_va
113
114       # Frequency bins determination
115        selected_filter = self.filter_var.get()
116        if selected_filter == "octave":
117            self.freqs = [31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000]
118        else:  # Octave-third
119            self.freqs = [25, 31.5, 40, 50, 63, 80, 100,
120            125, 160, 200, 250, 315, 400, 500, 630, 800, 1000,
121            1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300, 8000,
122            10000]
123
124        # Select time limits of the RIR
125        ttk.Label(self.frame_options, text="Start time [s]:").grid(row=1, column=0, padx
```

```
126         self.entry_tmin = tk.Entry(self.frame_options, fg="gray")
127         self.entry_tmin.grid(row=1, column=1)
128         self.entry_tmin.insert(0, "0.000")
129         self.entry_tmin.bind("<FocusIn>", lambda e: self.clear_placeholder(self.entry_tm
130         self.entry_tmin.bind("<FocusOut>", lambda e: self.restore_placeholder(self.entry
131 
132         ttk.Label(self.frame_options, text="End time [s]:").grid(row=1, column=2, padx=5
133         self.entry_tmax = tk.Entry(self.frame_options, fg="gray")
134         self.entry_tmax.grid(row=1, column=3)
135         self.entry_tmax.insert(0, "full length")
136         self.entry_tmax.bind("<FocusIn>", lambda e: self.clear_placeholder(self.entry_tm
137         self.entry_tmax.bind("<FocusOut>", lambda e: self.restore_placeholder(self.entry
138 
139         #Change limits button
140         self.btn_limits = ttk.Button(self.frame_options, text="Change limits", command=s
141         self.btn_limits.grid(row=1, column=4, padx=10)
142 
143 
144         # Reversed RIR
145         self.reversed_var = tk.BooleanVar()
146         self.check_reversed = ttk.Checkbutton(self.frame_options, text="Use reversed RIR
147         self.check_reversed.grid(row=0, column=5, padx=10)
148 
149         # Calculate parameters button
150         self.btn_calculate = ttk.Button(self.frame_options, text="Calculate acoustical p
151         self.btn_calculate.grid(row=1, column=6, padx=10, sticky="e")
152 
153 
154         # --- Results and graphics ---
155 
156         #Results frame
157         self.frame_results = ttk.LabelFrame(root, text="Results")
158         self.frame_results.pack(fill="both", expand=True, padx=10, pady=5)
159 
160         #Graphic and export buttons frame
161         self.frame_graphic = tk.Frame(self.frame_results, bg="white")
162         self.frame_graphic.pack(side="top", fill="both", expand=False)
163 
164 
165         # IR frame
166         self.graph_frame = tk.Frame(self.frame_graphic, bg='white')
167         self.graph_frame.pack(side='left', fill='both', expand=False)
168 
169         # Buttons frame
170         self.buttons_frame = tk.Frame(self.frame_graphic, bg='white')
171         self.buttons_frame.pack(side='right', fill='y', padx=10, pady=10)
172 
173         # Export buttons
174         self.btn_export_csv = tk.Button(self.buttons_frame, text="Export to .csv", comma
175         self.btn_export_csv.pack(side="bottom",pady=5, fill='x')
176 
177         self.btn_export_txt = tk.Button(self.buttons_frame, text="Export to .txt", comma
178         self.btn_export_txt.pack(side="bottom",pady=5, fill='x')
179 
180         # Channel L or R
181         self.btn_L = tk.Button(self.buttons_frame, text="Channel L", command=self.channe
182         self.btn_L.pack(side="top",pady=5, fill='x')
183 
```

```
184         self.btn_R = tk.Button(self.buttons_frame, text="Channel R", command=self.channe
185         self.btn_R.pack(side="top",pady=5, fill='x')
186 
187 
188         # Frame down
189         self.frame_down = tk.Frame(self.frame_results)
190         self.frame_down.pack(side="bottom", fill="both", expand= True)
191 
192         self.frame_down.configure(bg='white')
193 
194         self.frame_down.rowconfigure(0, weight=1)
195         self.frame_down.columnconfigure(1, weight=1)
196 
197 
198 
199         # RIR graphic
200         self.fig_ir, self.ax_ir = plt.subplots(figsize=(5, 1.8))
201         self.ax_ir.set_title("Impulse Response")
202         self.canvas_ir = FigureCanvasTkAgg(self.fig_ir, master=self.frame_graphic)
203         self.canvas_ir.get_tk_widget().pack(fill="x", ipadx=0, ipady=0)
204         self.canvas_ir.get_tk_widget().bind("<Button-3>", self.show_menu)
205 
206         self.ax_ir.set_xlabel("Time [s]", fontsize=6, fontweight="bold")
207         self.ax_ir.set_ylabel("Normalized Level", fontsize=6, fontweight="bold")
208 
209         self.ax_ir.set_title("Impulse Response", fontsize=7, fontweight="bold")
210 
211         # Reduce size of numbers in axes
212         self.ax_ir.tick_params(axis='x', labelsize=5.5)
213         self.ax_ir.tick_params(axis='y', labelsize=5.5)
214 
215         # Frame containing the table and the scrollbar
216         self.frame_table = tk.Frame(self.frame_down)
217         self.frame_table.grid(row=0,column=1,padx=3)
218 
219 
220         # Table
221         self.tree = ttk.Treeview(self.frame_table, columns=(), show='headings', height=8
222 
223 
224         columns = [str(f) for f in [31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000]]
225         self.tree["columns"] = columns
226         self.tree["show"] = "tree headings"
227 
228         self.tree.heading("#0", text="Freq [Hz]")
229         self.tree.column("#0", width=60, stretch=False)
230 
231         for freq in [31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000]:
232             self.tree.heading(freq, text=f"{freq}")
233             self.tree.column(freq, width=45, anchor="center", stretch=False)
234 
235         for i, name_param in enumerate(["EDT","T20", "T30", "C50", "C80", "Tt", "EDTt","
236             file = ["-","-","-","-","-","-","-","-","-"]
237             self.tree.insert("", "end", text=name_param, values=file)
238 
239 
240         # Scrollbar horizontal (ACTUAL)
241         self.scroll_x = ttk.Scrollbar(self.frame_table, orient="horizontal", command=sel
```

```
        self.tree.configure(xscrollcommand=self.scroll_x.set)

        # Ubication
        # self.tree.pack(side="top", fill="both", expand=True)
        # self.scroll_x.pack(side="bottom", fill="x")


        self.tree.grid(row=0, column=0, sticky="nsew")
        self.scroll_x.grid(row=1, column=0, sticky="ew")

        # Configure grid so the treeview expands
        self.frame_table.rowconfigure(0, weight=1)
        self.frame_table.columnconfigure(0, weight=1)

        # Vincule click for bars graphic and Schroeder
        self.tree.bind("<ButtonRelease-1>", self.on_treeview_click)
        self.root.bind_all("<Button-1>", self.on_root_click, add="+")


        # Schroeder inverse integral graphic
        self.fig_sch, self.ax_sch = plt.subplots(figsize=(5, 2.5))
        self.ax_sch.axis('off')
        self.ax_sch.set_xlabel("Time [s]",fontsize=5)
        self.ax_sch.set_ylabel("Level [dB]",fontsize=5)
        self.canvas_sch = FigureCanvasTkAgg(self.fig_sch, master=self.frame_down)
        self.canvas_sch.get_tk_widget().grid(row=0,column=0,padx=2)


        # Bars graphic
        self.fig_bars, self.ax_bars = plt.subplots(figsize=(5, 2.5))
        self.ax_bars.axis('off')

        self.canvas_barras = FigureCanvasTkAgg(self.fig_bars, master=self.frame_down)

        # Actual graphic is used to the software known if it's graph decay or bars.
        # Starts as None
        self.actual_graphic = None


    """
    --- Functions ---
    """
    def export_table_to_csv(self):
        filename = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CS
        if filename:
            with open(filename, mode='w', newline='', encoding='utf-8') as file:
                writer = csv.writer(file)
                # Head
                writer.writerow(["Parameter/Frequency"] + list(self.tree["columns"]))
                # Rows
                for item_id in self.tree.get_children():
                    row_name = self.tree.item(item_id)["text"]
                    row_values = self.tree.item(item_id)["values"]
                    writer.writerow([row_name] + row_values)

    def export_table_to_txt(self):
        filename = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Te
        if filename:
```

```
300                 with open(filename, mode='w', encoding='utf-8') as file:
301                     #Head
302                     file.write("Parameter/Frequency\t" + "\t".join(self.tree["columns"]) + "
303                     #Rows
304                     for item_id in self.tree.get_children():
305                         row_name = self.tree.item(item_id)["text"]
306                         row_values = self.tree.item(item_id)["values"]
307                         file.write(f"{row_name}\t" + "\t".join(str(v) for v in row_values) +
308 
309 
310         def on_root_click(self, event):
311             # Check if there is a click outside Treeview
312             widget = event.widget
313 
314             if widget not in (self.tree, self.entry_window, self.entry_tmin, self.entry_tmax
315                 # Desmark selection and focus
316                 self.tree.selection_remove(self.tree.selection())
317                 self.tree.focus("")
318                 self.graph_schroeder()
319 
320 
321         def clear_placeholder(self, entry, placeholder):
322             if entry.get() == placeholder:
323                 entry.delete(0, tk.END)
324                 entry.config(fg="black")
325 
326 
327         def calculate_parameters(self):
328 
329             # Obtain given limits
330             try:
331                 t_min_str = self.entry_tmin.get()
332                 t_max_str = self.entry_tmax.get()
333 
334                 t_min = float(t_min_str) if t_min_str != "0.000" else 0.0
335                 if t_max_str == "full length":
336                     t_max = len(self.signal) / self.fs
337                 else:
338                     t_max = float(t_max_str)
339 
340                 # Validate
341                 if t_min > t_max:
342                     t_min, t_max = t_max, t_min
343 
344             except ValueError:
345                 t_min = 0.0
346                 t_max = len(self.signal) / self.fs
347 
348             # Proccesing RIR with limits given
349             if self.stereo_signal is not None:
350                 self.rir_final_stereo =[]
351                 self.rir_final_stereo.append(self.stereo_signal[:, 0][int(t_min*self.fs):int
352                 self.rir_final_stereo.append(self.stereo_signal[:, 1][int(t_min*self.fs):int
353             else:
354                 self.rir_final = self.signal[int(t_min*self.fs):int(t_max*self.fs)]
355 
356 
357             #Schroeder integral array
```

```
358         if self.stereo_signal is not None:
359             etc_stereo = []
360             etc_stereo.append(proc.calcular_etc(self.rir_final_stereo[0]))
361             etc_stereo.append(proc.calcular_etc(self.rir_final_stereo[1]))
362 
363             self.sch_db_stereo = []
364             self.sch_db_stereo.append(proc.schroeder_reverse_integral_numba(etc_stereo[0
365             self.sch_db_stereo.append(proc.schroeder_reverse_integral_numba(etc_stereo[1
366 
367             if self.active_channel == "L":
368                 self.sch_db = self.sch_db_stereo[0]
369             else:
370                 self.sch_db = self.sch_db_stereo[1]
371 
372         else:
373             etc = proc.calcular_etc(self.rir_final)
374             self.sch_db = proc.schroeder_reverse_integral_numba(etc)
375 
376 
377         #Simulated values
378 #        if self.stereo_signal is not None:
379 #            self.calculated_parameters = ([],[])
380 #            for i in range(2):
381 #                # Generar datos simulados: matriz [parametro][frecuencia]
382 #                for _ in self.parameters:
383 #                    file = [round(random.uniform(0.3, 2.5), 2) for _ in self.freqs]
384 #                    self.calculated_parameters[i].append(file)
385 #            if self.active_channel == "L":
386 #                values = self.calculated_parameters[0]
387 #            else:
388 #                values = self.calculated_parameters[1]
389 #
390 #        else:
391 #            # Generar datos simulados: matriz [parametro][frecuencia]
392 #            values = []
393 #            for _ in self.parameters:
394 #                file = [round(random.uniform(0.3, 2.5), 2) for _ in self.freqs]
395 #                values.append(file)
396 
397 
398         #Calculate parameters function
399         if self.stereo_signal is not None:
400             self.IACC_array = proc.IACCearly_por_banda(self.rir_final_stereo[0],self.rir
401             self.calculated_parameters = [0,0]
402             self.tt_global,self.edtt_global =[0,0],[0,0]
403 
404             self.calculated_parameters[0],self.tt_global[0],self.edtt_global[0] = proc.p
405             self.calculated_parameters[1],self.tt_global[1],self.edtt_global[1] = proc.p
406 
407             if self.active_channel == "L":
408                 values = self.calculated_parameters[0]
409                 self.tt = self.tt_global[0]
410                 self.edtt = self.edtt_global[0]
411             else:
412                 values = self.calculated_parameters[1]
413                 self.tt = self.tt_global[0]
414                 self.edtt = self.edtt_global[0]
415 
```

```
416             else:
417                 values,self.tt,self.edtt = proc.procesar_rir_completo(self.rir_final,self.fs
418 
419 
420             self.update_table(self.freqs, self.parameters, values)
421             self.canvas_sch.get_tk_widget().grid(row=0,column=0,padx=2)
422             self.graph_schroeder()
423             self.calculated = 1
424             self.label_calculating.config(text="Done!")
425             print("Done!")
426             self.values_parametros = values
427             self.plot_signal()
428 
429 
430         def calculate_parameters_from_button(self):
431 
432             if self.signal is None:
433                 if self.label_nofile is not None:
434                     pass
435                 else:
436                     self.label_nofile = ttk.Label(self.frame_options, text="No file selected
437                     self.label_nofile.grid(row=0, column=6, padx=10)
438                 return
439 
440             if self.label_nofile is not None:
441                 self.label_nofile.grid_forget()
442 
443             try:
444                 window_ms = float(self.entry_window.get())
445                 if window_ms < 0.1:
446                     msgbox.showerror("Error", "Enter a valid window size.")
447                     return
448             except ValueError:
449                 msgbox.showerror("Error", "Window size must be positive.")
450                 return
451 
452             if self.entry_tmax.get() == "full length":
453                 pass
454             elif float(self.entry_tmax.get()) < 0.3:
455                 msgbox.showwarning("Warning", "IR length may be too low.")
456                 print("Warning: IR length may be too low.")
457 
458 
459             self.label_calculating = ttk.Label(self.frame_options, text="Calculating...")
460             self.label_calculating.grid(row=0, column=6, padx=10)
461 
462 
463             if self.stereo_signal is not None:
464                 print("Stereo signal")
465             else:
466                 print("Mono signal")
467             print("--------------")
468             print("Processing options:")
469             print(f"- Window size: {self.entry_window.get()} ms")
470             print(f"- Filter type: {self.filter_var.get()}")
471             print(f"- Reversed RIR: {self.reversed_var.get()}")
472 
473             # Frequency bins determination
```

```
474         filtro = self.filter_var.get()
475         if filtro == "octave":
476             self.freqs = [31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000]
477         else:  # Octave-third
478             self.freqs = [25, 31.5, 40, 50, 63, 80, 100,
479             125, 160, 200, 250, 315, 400, 500, 630, 800, 1000,
480             1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300, 8000,
481             10000]
482
483
484         self.parameters = ["EDT","T20", "T30", "C50", "C80", "Tt", "EDTt","IACC_e"]
485
486         # Calculating parameters in other function in order to label_calculating appears
487         self.root.after(100, self.calculate_parameters)
488
489
490
491     def update_table(self, freqs, parameters, values):
492         self.tree.delete(*self.tree.get_children())
493
494         columns = [str(f) for f in freqs]
495         self.tree["columns"] = columns
496         self.tree["show"] = "tree headings"
497
498         self.tree.heading("#0", text="Freq [Hz]")
499         self.tree.column("#0", width=80, stretch=False)
500
501         for freq in freqs:
502             self.tree.heading(freq, text=f"{freq}")
503             self.tree.column(freq, width=45, anchor="center", stretch=False)
504
505         for i, name_param in enumerate(parameters):
506             file = values[i]
507             self.tree.insert("", "end", text=name_param, values=file)
508
509
510     def graph_schroeder(self):
511         # Delete graphic is there is one existing
512         if self.actual_graphic is not None:
513             self.actual_graphic.get_tk_widget().destroy()
514             self.actual_graphic = None
515
516         fig, ax = plt.subplots(figsize=(5, 2))
517         ax.axis('on')
518         ax.plot(np.linspace(0,len(self.sch_db)/self.fs,len(self.sch_db)), self.sch_db)
519         ax.set_title("Energy Decay",fontsize=6, fontweight="bold", pad=3)
520         ax.set_xlabel("Time [s]",fontsize=5, fontweight="bold", labelpad=0.5)
521         ax.set_ylabel("Level [dB]",fontsize=5, fontweight="bold", labelpad=2)
522         ax.tick_params(axis='x', labelsize=4.5)
523         ax.tick_params(axis='y', labelsize=4.5)
524         ax.grid(True)
525         fig.tight_layout()
526
527
528         self.actual_graphic = FigureCanvasTkAgg(fig, master=self.frame_down)
529         self.actual_graphic.get_tk_widget().grid(row=0,column=0,padx=2)
530         self.actual_graphic.draw()
531         self.actual_graphic.get_tk_widget().bind("<Button-3>", self.show_menu)
```

```
532
533
534     def on_treeview_click(self, event):
535         # See what parameter on the table is clicked
536         item_id = self.tree.focus()
537         if not item_id:
538             return
539
540         # Delets previous graphic on the position if there is one
541         if self.actual_graphic is not None:
542             self.actual_graphic.get_tk_widget().destroy()
543             self.actual_graphic = None
544
545         name_param = self.tree.item(item_id)["text"]
546         values = [float(value) for value in (self.tree.item(item_id)["values"])]
547         freqs_float = [float(col) for col in self.tree["columns"]]
548         freqs = [int(f) if float(f) == int(f) else float(f) for f in freqs_float]
549
550
551         fig, ax = plt.subplots(figsize=(5, 2))
552         ax.axis('on')
553
554         # Width calculated to avoid superposition of bars
555         if len(freqs) < 11:
556             widths = np.diff(np.append(freqs, freqs[-1] * 2)) * 0.6
557         else:
558             widths = np.diff(np.append(freqs, freqs[-1] * 1.25)) * 0.6
559
560         ax.bar(freqs, values, width=widths, align='center', color='skyblue', edgecolor='
561
562         # Set graph title
563         if name_param != "C50" and name_param != "C80" and name_param!= "IACC_e":
564             ax.set_title(f"{name_param} [s]", fontsize=5, fontweight="bold")
565         elif name_param == "IACC_e":
566             ax.set_title(f"{name_param}", fontsize=6, fontweight="bold")
567         else:
568             ax.set_title(f"{name_param} [dB]", fontsize=5, fontweight="bold")
569
570         ax.set_xlabel("Freq [Hz]",fontsize=5)
571         ax.set_xscale("log")
572
573         ax.set_xticks(freqs)
574
575
576         # Set x ticks for good representation
577         if len(freqs) > 11:
578             visible_freqs = []
579             i = 0
580             for f in freqs:
581                 if i == 0:
582                     visible_freqs.append(f)
583                     i = 3
584                 else:
585                     visible_freqs. append("")
586                     i -=1
587         else:
588             visible_freqs = freqs
589         ax.set_xticklabels(visible_freqs, ha='center')
```

```
590
591         ax.tick_params(axis='x', labelsize=5.5)
592         ax.tick_params(axis='y', labelsize=5.5)
593
594
595         ax.grid(True, which='major', axis="y", linestyle='-', linewidth=0.7)
596         fig.tight_layout()
597
598
599         self.actual_graphic = FigureCanvasTkAgg(fig, master=self.frame_down)
600         self.actual_graphic.get_tk_widget().grid(row=0,column=0,padx=2)
601         self.actual_graphic.draw()
602         self.actual_graphic.get_tk_widget().bind("<Button-3>", self.show_menu)
603
604
605
606     def restore_placeholder(self, entry, placeholder):
607         #Sets default values for boxes
608         if not entry.get():
609             entry.insert(0, placeholder)
610             entry.config(fg="gray")
611
612
613     def load_file(self):
614         file_loaded = filedialog.askopenfilename(filetypes=[("WAV files", "*.wav")])
615         if file_loaded:
616             self.label_file.config(text=file_loaded.split("/")[-1])
617             self.fs, data = wavfile.read(file_loaded)
618
619             # Convert to float if it's int16
620             if data.dtype == np.int16:
621                 data = data / 32768.0
622             elif data.dtype == np.int32:
623                 data = data / 2147483648.0
624
625             #Stereo or mono
626             if len(data.shape) == 2:
627                 self.stereo_signal = data
628                 self.signal = data[:, 0]    # For default: Channel L
629                 self.active_channel = "L"
630             else:
631                 self.stereo_signal = None
632                 self.signal = data
633
634             self.calculated = None
635
636             # Times by default
637             length = len(self.signal) / self.fs
638             self.entry_tmin.delete(0, tk.END)
639             self.entry_tmin.insert(0, "0.000")
640             self.entry_tmin.config(fg="gray")
641
642             self.entry_tmax.delete(0, tk.END)
643             self.entry_tmax.insert(0, "full length")
644             self.entry_tmax.config(fg="gray")
645
646             self.tt_global,self.edtt_global=None,None
647             self.tt,self.edtt = None,None
```

```
648
649                 self.plot_signal()
650
651         def plot_signal(self):
652             if self.signal is None:
653                 return
654
655             self.ax_ir.clear()
656             t = np.arange(len(self.signal)) / self.fs
657             self.ax_ir.plot(t, self.signal)
658
659             # Obtain given limits
660             try:
661                 t_min_str = self.entry_tmin.get()
662                 t_max_str = self.entry_tmax.get()
663
664                 t_min = float(t_min_str) if t_min_str != "0.000" else 0.0
665                 if t_max_str == "full length":
666                     t_max = len(self.signal) / self.fs
667                 else:
668                     t_max = float(t_max_str)
669
670                 # Validate
671                 if t_min > t_max:
672                     t_min, t_max = t_max, t_min
673
674             except ValueError:
675                 t_min = 0.0
676                 t_max = len(self.signal) / self.fs
677
678             # Drawing limits line
679             self.ax_ir.axvline(x=t_min, color='green', linestyle='--')
680             self.ax_ir.axvline(x=t_max, color='red', linestyle='--')
681
682             #Drawing Tt line and Tt and EDTt text
683             if self.tt is not None and self.edtt is not None:
684                 self.ax_ir.axvline(x=self.tt, color='blue', linestyle='-')
685                 self.ax_ir.text(self.tt + 0.02, self.ax_ir.get_ylim()[0] + 0.9 * (self.ax_ir
686             "Tt", color='blue', ha='left', va='top', fontsize=8,bbox=dict(facecolor='white',
687
688                 self.ax_ir.text(0.4, 0.8, f"Tt = {self.tt:.2f} s     EDTt = {self.edtt:.2f} s
689
690             self.ax_ir.set_xlabel("Time [s]", fontsize=6, fontweight="bold", labelpad=0.5)
691             self.ax_ir.set_ylabel("Normalized Level", fontsize=6, fontweight="bold", labelpa
692
693
694             # Reduce size of axes numbers
695             self.ax_ir.tick_params(axis='x', labelsize=5.5)
696             self.ax_ir.tick_params(axis='y', labelsize=5.5)
697
698             if self.stereo_signal is not None:
699                 self.ax_ir.set_title("Impulse Response " + "Channel " + self.active_channel,
700             else:
701                 self.ax_ir.set_title("Impulse Response", fontsize=7, fontweight="bold")
702
703             self.ax_ir.grid()
704             self.fig_ir.tight_layout()
705             self.canvas_ir.draw()
```

```
706
707
708     def update_vertical_lines(self):
709         try:
710             if float(self.entry_tmin.get()) < 0:
711                 msgbox.showerror("Error", "Start time less than zero.")
712                 return
713         except ValueError:
714             msgbox.showerror("Error", "Start time less than zero.")
715             return
716 
717         try:
718             if self.entry_tmax.get() == "full length":
719                 pass
720             elif float(self.entry_tmax.get()) > len(self.signal)/self.fs:
721                 msgbox.showerror("Error", "End time greater than signal length.")
722                 return
723         except ValueError:
724             msgbox.showerror("Error", "End time greater than signal length.")
725             return
726 
727 
728         if self.signal is not None:
729             self.plot_signal()
730 
731     def channel_L(self):
732         if self.stereo_signal is not None and self.calculated == 1:
733             self.active_channel = "L"
734             self.sch_db = self.sch_db_stereo[0]
735             self.signal = self.stereo_signal[:, 0]
736             self.tt,self.edtt = self.tt_global[0],self.edtt_global[0]
737             self.plot_signal()
738 
739             self.update_table(self.freqs, self.parameters, self.calculated_parameters[0]
740             self.canvas_sch.get_tk_widget().grid(row=0,column=0,padx=2)
741             self.graph_schroeder()
742 
743     def channel_R(self):
744         if self.stereo_signal is not None and self.calculated == 1:
745             self.active_channel = "R"
746             self.sch_db = self.sch_db_stereo[1]
747             self.signal = self.stereo_signal[:, 1]
748             self.tt,self.edtt = self.tt_global[1],self.edtt_global[1]
749             self.plot_signal()
750 
751             self.update_table(self.freqs, self.parameters, self.calculated_parameters[1]
752             self.canvas_sch.get_tk_widget().grid(row=0,column=0,padx=2)
753             self.graph_schroeder()
754 
755     def show_menu(self, event):
756         try:
757             self.popup_menu.tk_popup(event.x_root, event.y_root)
758         finally:
759             self.popup_menu.grab_release()
760 
761     def save_image(self):
762         if self.actual_graphic is None:
763             msgbox.showwarning("Warning", "No energy decay or parameter graph.")
```

```
            print("No energy decay or parameter graph.")
            return
        filename = filedialog.asksaveasfilename(defaultextension=".png",
                                                 filetypes=[("PNG", "*.png"), ("All file
        if filename:
            self.actual_graphic.figure.savefig(filename)

    def save_IR_image(self):
        filename = filedialog.asksaveasfilename(defaultextension=".png",
                                                 filetypes=[("PNG", "*.png"), ("All file
        if filename:
            self.canvas_ir.figure.savefig(filename)


#------Running the app--------
if __name__ == "__main__":
    root = tk.Tk()
    app = IRAnalyzerApp(root)
    root.mainloop()



```

ii procesamiento_rir.py

```
import numpy as np
import pandas as pd
from scipy.signal import hilbert, firwin, fftconvolve, medfilt
from scipy.stats import linregress
from concurrent.futures import ThreadPoolExecutor
from numba import njit
from scipy.ndimage import uniform_filter1d
from scipy.stats import linregress

# ----------------------------- TRUNCAMIENTO INICIAL ------------------------------
def truncado_inicial(rir, fs=None, margen_ms=0):
    envolvente = np.abs(hilbert(rir))
    idx_max = np.argmax(envolvente)
    if fs is not None and margen_ms > 0:
        muestras_margen = int(fs * (margen_ms / 1000))
        idx_inicio = max(0, idx_max - muestras_margen)
    else:
        idx_inicio = idx_max
    return rir[idx_inicio:]

# ----------------------------- FILTRADO ULTRA RÁPIDO ------------------------------
def filtrar_banda_fir(fc, low, high, rir, fs, reversed=False, numtaps=1024):
    fir = firwin(numtaps, [low, high], pass_zero=False, fs=fs)
    if reversed:
        rir = rir[::-1]
        rir_filtrada = fftconvolve(rir, fir, mode='same')[::-1]
    else:
        rir_filtrada = fftconvolve(rir, fir, mode='same')
    return fc, rir_filtrada

def filtrar_por_bandas(rir, fs, modo='octave', reversed=False):
    sqrt2 = np.sqrt(2)
    bandas = {
```

```
34          'octave': [31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000],
35          'third': [25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315,
36                    400, 500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150,
37                    4000, 5000, 6300, 8000, 10000]
38      }
39
40      if modo not in bandas:
41          raise ValueError("Modo inválido. Usar 'octave' o 'third'.")
42
43      fc_list = np.array(bandas[modo])
44      f1 = fc_list / sqrt2
45      f2 = fc_list * sqrt2
46
47      with ThreadPoolExecutor() as executor:
48          results = list(executor.map(
49              lambda args: filtrar_banda_fir(*args, rir=rir, fs=fs, reversed=reversed),
50              zip(fc_list, f1, f2)
51          ))
52      return results
53
54  # ----------------------------- ENERGY TIME CURVE (ETC) -----------------------------
55  def calcular_etc(rir):
56      h = hilbert(rir)
57      energia = np.abs(h) ** 2
58      return energia / np.max(energia + 1e-12)
59
60  def calcular_tamano_ventana_mm(fs, ventana_ms):
61      muestras = int((ventana_ms / 1000) * fs)
62      return muestras + 1 if muestras % 2 == 0 else muestras
63
64  def aplicar_mmf(etc, tamano_ventana):
65      return medfilt(etc, kernel_size=tamano_ventana)
66
67  @njit
68  def schroeder_reverse_integral_numba(etc):
69      energia = np.cumsum(etc[::-1])[::-1]
70      energia /= energia[0] + 1e-12
71      sch_db = 10 * np.log10(energia + 1e-12)
72      return sch_db
73
74  def procesar_etc_con_mmf(rir, fs, ventana_ms):
75      etc = calcular_etc(rir)
76      ventana = calcular_tamano_ventana_mm(fs, ventana_ms)
77      etc_suavizada = aplicar_mmf(etc, ventana)
78      sch_db = schroeder_reverse_integral_numba(etc_suavizada)
79      return etc_suavizada, sch_db
80
81  # ---------------------- EXPORTAR A CSV ----------------------
82  def exportar_parametros_csv(param_dict, ruta, nombre_archivo="parametros.csv"):
83      df = pd.DataFrame(param_dict).T
84      df = df.T
85      df.index.name = "Parameter/Frequency"
86      df = df.round(3)
87      orden_parametros = ["EDT", "T20", "T30", "C50", "C80", "Tt", "EDTt"]
88      df = df.reindex(orden_parametros)
89      df = df.applymap(lambda x: str(x).replace('.', ','))
90      df.to_csv(f"{ruta}/{nombre_archivo}", sep=';', index=True, encoding='utf-8')
91
```

```
# ------------------------------ RT y EDT ------------------------------
def calcular_rt(sch_db, fs, tipo=None):
    def calcular_tiempo(sch_db, fs, db_ini, db_fin):
        if sch_db[0] < db_ini or sch_db[0] < db_fin:
            return np.nan
        mask_ini = sch_db <= db_ini
        mask_fin = sch_db <= db_fin
        if not np.any(mask_ini) or not np.any(mask_fin):
            return np.nan
        idx_ini = np.argmax(mask_ini)
        idx_fin = np.argmax(mask_fin)
        if idx_fin <= idx_ini:
            return np.nan
        tiempo = np.arange(len(sch_db)) / fs
        x = tiempo[idx_ini:idx_fin]
        y = sch_db[idx_ini:idx_fin]
        slope, _, _, _, _ = linregress(x, y)
        return -60 / slope if slope < 0 else np.nan

    resultados = {
        'EDT': calcular_tiempo(sch_db, fs, 0, -10),
        'T20': calcular_tiempo(sch_db, fs, -5, -25),
        'T30': calcular_tiempo(sch_db, fs, -5, -35)
    }
    return resultados if tipo is None else resultados.get(tipo.upper(), np.nan)

# ------------------------------ C50 y C80 ------------------------------
@njit
def calcular_clarity_numba(rir, fs, limite_ms):
    energia_total = np.sum(rir**2)
    limite_muestras = int((limite_ms / 1000) * fs)
    energia_temprana = np.sum(rir[:limite_muestras]**2)
    energia_tardia = energia_total - energia_temprana
    return 10 * np.log10(energia_temprana / energia_tardia + 1e-12)

# ------------------------------ Tt y EDTt ------------------------------

def tt_edtt(rir, fs):
    """
    La RIR debe estar truncada y ser mono, siendo el t=0 s el punto máximo de la misma.
    """

    # Normalizar
    rir = rir / np.max(np.abs(rir))

    # Energía instantánea
    energia = rir**2

    # Energía acumulada
    energia_acumulada = np.cumsum(energia)

    # Energía total
    energia_total = np.sum(energia)

    # Índice donde se alcanza el 99%
    indice_tt = np.where(energia_acumulada >= 0.99 * energia_total)[0][0]

    # Tiempo de transición
```

```
150     tt = indice_tt / fs
151 
152     # RIR en dB
153     energia_db = 10 * np.log10(energia + 1e-20)  # evitar log(0)
154 
155     # Suavizar curva en dB
156     # energia_db_suavizada = uniform_filter1d(energia_db, size=100)
157 
158     # Datos entre el pico y el Tt para regresión
159     x = np.arange(0, indice_tt) / fs
160     y = energia_db[0:indice_tt]
161 
162     # Regresión lineal
163     slope, intercept, _, _, _ = linregress(x, y)
164 
165     # Punto de cruce de la regresión con nivel en Tt
166     # y_tt = energia_db[indice_tt]
167     # t_cruce = (y_tt - intercept) / slope
168 
169     edtt = -60 / slope
170 
171     return tt, edtt
172 
173 #----------------------------------------IACCEarly----------------------------------------
174 
175 def IACCearly(señal_izq, señal_der, fs, tau=0.08):
176     """
177     Calcula IACCearly según ISO 3382.
178 
179     Parámetros:
180         señal_izq: ndarray - señal del oído izquierdo
181         señal_der: ndarray - señal del oído derecho
182         fs: int - frecuencia de muestreo [Hz]
183         tau: float - ventana de integración en segundos (ej. 0.08 para 80ms)
184 
185     Retorno:
186         iacc_early: float - valor de IACCearly
187     """
188     # Número de muestras a considerar
189     muestras_ventana = int(fs * tau)
190 
191     # Recortar ambas señales a la duración de integración (desde t = 0)
192     señal_izq = señal_izq[:muestras_ventana]
193     señal_der = señal_der[:muestras_ventana]
194 
195     # Rango de desfase interaural ±1 ms
196     max_desfase = int(fs * 0.001)
197     lags = np.arange(-max_desfase, max_desfase + 1)
198 
199     correlaciones = []
200     for lag in lags:
201         if lag >= 0:
202             izq = señal_izq[:muestras_ventana - lag]
203             der = señal_der[lag:muestras_ventana]
204         else:
205             izq = señal_izq[-lag:muestras_ventana]
206             der = señal_der[:muestras_ventana + lag]
207 
```

```
208         # Normalizar y calcular correlación cruzada
209         izq = np.array(izq)
210         der = np.array(der)
211         num = np.sum(izq * der)
212         den = np.sqrt(np.sum(izq ** 2) * np.sum(der ** 2))
213         correlaciones.append(np.abs(num / den) if den > 0 else 0)
214 
215 
216     return np.max(correlaciones)
217 
218 
219 def IACCearly_por_banda(rir_L, rir_R, fs, tau=0.08, modo="octave"):
220     """
221     Calcula IACCearly por banda para una respuesta estéreo al impulso.
222 
223     Parámetros:
224     - rir_L: ndarray, respuesta al impulso canal izquierdo
225     - rir_R: ndarray, respuesta al impulso canal derecho
226     - fs: int, frecuencia de muestreo
227     - tau: float, ventana de integración en segundos (por defecto 0.08 s)
228     - modo: str, tipo de filtrado por bandas ("octave", "1/3oct", etc.)
229 
230     Retorna:
231     - Lista de valores IACCearly, uno por banda.
232     """
233 
234     # Truncar al inicio (1 ms)
235     rir_truncada_L = truncado_inicial(rir_L, fs, margen_ms=1)
236     rir_truncada_R = truncado_inicial(rir_R, fs, margen_ms=1)
237 
238     # Filtrar por bandas (usando `reversed`, que se asume definido externamente)
239     lista_izq = filtrar_por_bandas(rir_truncada_L, fs, modo, reversed=reversed)
240     lista_der = filtrar_por_bandas(rir_truncada_R, fs, modo, reversed=reversed)
241 
242     # Aplanar y asegurar arrays 1D
243     lista_izq = [np.ravel(np.array(rir)) for _, rir in lista_izq]
244     lista_der = [np.ravel(np.array(rir)) for _, rir in lista_der]
245 
246     assert len(lista_izq) == len(lista_der), "Las listas deben tener la misma longitud."
247 
248     # Cálculo paralelo
249     with ThreadPoolExecutor() as executor:
250         results = list(executor.map(
251             lambda args: IACCearly(*args, fs=fs, tau=tau),
252             zip(lista_izq, lista_der)
253         ))
254 
255     return results
256 
257 # ----------------------------- FUNCION PRINCIPAL COMPLETA -----------------------------
258 def procesar_rir_completo(
259     rir_original, fs,IACC_e=None,
260     modo='octave', ventana_ms=30,
261     ruta_export="./", nombre_csv="parametros.csv",
262     reversed=False
263 ):
264     rir_truncada = truncado_inicial(rir_original, fs, margen_ms=1)
265     bandas_filtradas = filtrar_por_bandas(rir_truncada, fs, modo, reversed=reversed)
```

```
266
267     tt_global,edtt_global= tt_edtt(rir_truncada, fs)
268
269     resultados = {}
270     i=0
271     for fc, rir_filtrada in bandas_filtradas:
272         etc, sch_db = procesar_etc_con_mmf(rir_filtrada, fs, ventana_ms=ventana_ms)
273         rt_vals = calcular_rt(sch_db, fs)
274         c50 = calcular_clarity_numba(rir_filtrada, fs, 50)
275         c80 = calcular_clarity_numba(rir_filtrada, fs, 80)
276         tt, edtt = tt_edtt(rir_filtrada, fs)
277         if IACC_e is not None:
278             resultados[fc] = {
279                 'T20': rt_vals['T20'],
280                 'T30': rt_vals['T30'],
281                 'EDT': rt_vals['EDT'],
282                 'C50': c50,
283                 'C80': c80,
284                 'Tt': tt,
285                 'EDTt': edtt,
286                 'IACC_e': IACC_e[i]
287             }
288         else:
289             resultados[fc] = {
290                 'T20': rt_vals['T20'],
291                 'T30': rt_vals['T30'],
292                 'EDT': rt_vals['EDT'],
293                 'C50': c50,
294                 'C80': c80,
295                 'Tt': tt,
296                 'EDTt': edtt,
297                 'IACC_e': 1
298             }
299         i+=1
300
301     lista_final = []
302
303     parametros = ["EDT","T20", "T30", "C50", "C80", "Tt", "EDTt", "IACC_e"]
304
305     if modo=="octave":
306         freqs = [31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000]
307     else:
308         freqs = [25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315,
309                  400, 500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150,
310                  4000, 5000, 6300, 8000, 10000]
311     for parametro in parametros:
312         fila = []
313         for freq in freqs:
314             valor = resultados.get(freq, {}).get(parametro, "0")
315             if isinstance(valor, (int, float)):
316                 valor = round(valor, 2)
317             fila.append(valor)
318         lista_final.append(fila)
319
320     return lista_final,tt_global,edtt_global
321
322
```