UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH, ESCOLA SUPERIOR D'ENGINYERIES INDUSTRIAL, AEROESPACIAL I AUDIOVISUAL DE TERRASSA

Study on a monitoring system of the noise level in the work rooms of the campus library in Terrassa

Author: Oriol Morros Llordella

Tutor: Albert Masip Alvarez

January 16, 2021

Abstract

This project aims at implementing the architecture proposed by a previous work done in the university, in which different paths to implementing an active noise cancelling system were attempted in simulation. The method chosen

consists in identifying the system between a workroom in the UPC library and the outside of it, though in this thesis the experiment was done with a box due to the covid-19 restrictions, and attempt to create a noise cancelling system with low cost hardware with both feedforward and feedback (PID) control through the tuning of said controllers using the parameters gotten in the identification of the system. The result will be evaluated and improvement proposals will be made.

CONTENTS

1 INTRODUCTION

1.1 AIM

This thesis is conceptualised as a continuation of a previous thesis, “Sound monitoring and active noise cancelling at the campus library” [1]. In that document, a theoretical analysis of the problem of active noise cancellation and control in the campus library and possible solutions was made, as well as a proposition for the architecture of those solutions. The aim of this thesis is to put that architecture into practice in either the campus library or a simplified version of it, by monitoring the noise level as well as implementing an active noise cancellation system in a single board computer and reviewing the results. Given the global health conditions, the library and travel has been severally limited, and thus this project will be done in a smaller simulated environment with a box meant to work as a stand in for the walls of the library room, and see the viability of it, easily implementable in a different environment once the process is prepared.

1.2 SCOPE

- Familiarisation with the software and building of programs to read the sound signals
- Processing the sound signals to identify the primary and secondary paths
- Controller tuning: feedback and feed-forward; application in simulation and comparison
- Controller tuning: feedback and feed-forward; real and comparative application
- Least mean squares adaptive filter, open loop application as feed forward structure
- ANC application with both filters
- Conclusions

1.3 REQUIREMENTS

- The cost of the components should be kept to the minimum necessary to guarantee a good noise cancellation.
- The processing unit should be fast enough to make the process causal.
- The data acquisition should be qualitative enough to provide relevant measurements that can be used for advanced audio parameters calculation.
- The processing unit has to be able to connect to the used peripherals (e.g microphone, loudspeaker).

1.4 JUSTIFICATION OF THE NEED

In many public spaces, including but not limited to schools, libraries and hospitals there is little sound isolation between rooms, and given the increasing need for separation in all those spaces and the much needed ambiance difference between rooms (for example between the work rooms, where conversation is allowed to happen and the main hall, where silence is required). Passive isolation, or what is commonly called sound isolation comes down to adding an isolating agent to the walls, and sometimes to the ceiling and floor too, like cork or sawdust. That process is very simple and it's effective, but it has its inconvenients. It usually does not allow for transparencies through the walls, it takes space away and can be expensive. The alternative solution proposed in this document is sound monitoring and active noise cancellation. It not only allows for automatic noise control, the threshold for which can be manipulated by the staff, but it also allows for an automated way to reduce the influence of the noise coming from the room without suffering from many of the flaws of the passive isolation alternative, becoming a very promising proposition. Previous works on the topic have given satisfying results [2] [3].

2 Background

In the UPC campus library in Terrassa, as well as many other libraries and similar places of study, there's the main hall where no chatter is allowed and also study rooms, where groups can go and work together. A higher level of sound is generated by groups talking inside the rooms, and it would be costly in both space and money to sufficiently stop the noise from being heard outside those rooms with a traditional passive isolation installation. A good application

of an active noise cancelling system, along with some noise monitoring and warning can prove to be a much cheaper and less space consuming approach to reducing the noise generated, making for a more modern and less bulky library and freeing up some work for the librarians.

2.1 NOISE MONITORING

The first approach to a solution would entail signaling the people inside the room when their volume of speech is too high. In order to prioritise the sounds that would bother human ears the most, an A weighting filter could be applied. The A weighting filter gives priority to sound pressure levels or the loudness of certain frequencies over others, allowing for lower frequency noises like the moving of chairs to not bother the algorithm and making it more precise in detecting human chatter[4].

In order for the A weighting algorithm to be implemented, the incoming sound needs to be transformed from the time domain to the frequency domain using the fast fourier transform, when in the frequency domain the A weighting filter would be applied, following up with the inverse fourier transform to get the volume response back in the time domain. After that point, the algorithm would only need to signal with a led or a small sound when the room exceeded the noise threshold.

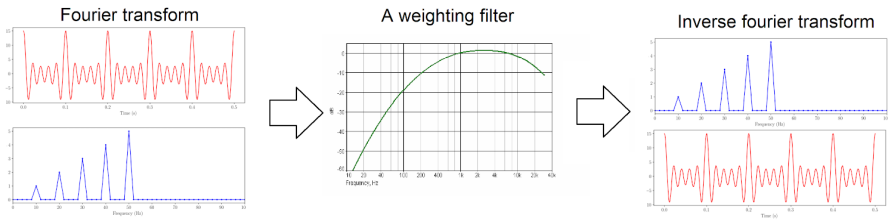

Figure 2.1: A weighting filter principle

2.2 Active noise cancellation

Active noise cancelling systems are already used in a variety of places in society, including mechanical cars, planes and ear protectors. It is most effective when used to reduce repetitive noise of low frequency, none of which will be the case for this thesis, but it can still bring a significant reduction in noise levels in the desired location.

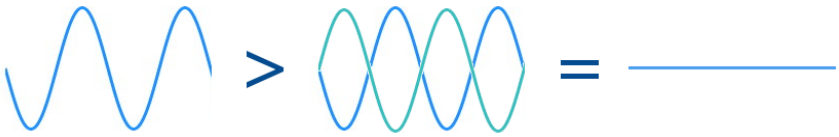

Figure 2.2: ANC principle

The principle is quite simple. A microphone records the reference audio, in this case it would be inside the room, then that audio is processed and inverted to come out of a speaker, generating “anti noise”, and a second microphone located at the target location will give feedback and try to correct the signal sent from the speaker.

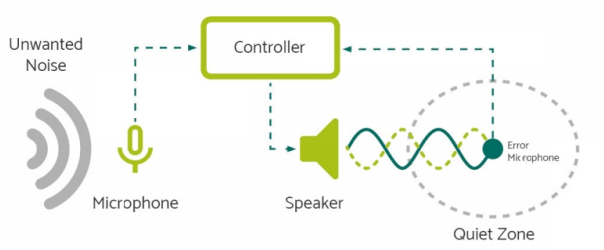

Figure 2.3: Overview of the process

PRIMARY AND SECONDARY PATHS: In order to design the controller and work with a simulation of the real process, two abstractions in the form of transfer functions will be created. From the reference microphone to the error microphone there's a number of obstacles and conditions which alter the form of the sound wave, acting like a filter. The difference between input and output in that direction will be referred to as the primary path. Similarly, the way from the speaker generating anti noise to the same error microphone will be called the secondary path. The identification of both primary and secondary paths will be necessary to design the feedback and feed forward controllers.

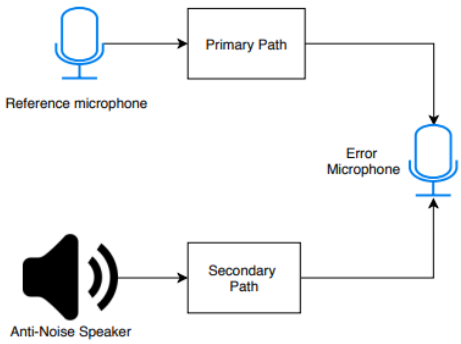

Figure 2.4: Primary and secondary paths

APPROACHES: There are two broad approaches to implementing a ANC application.

- An adaptive filter can be used, ignoring the primary path and using the input and output signals to correct the output signal making use of the least mean squares algorithm. This approach gave unsatisfactory results in the theoretical study of this problem in the theoretical analysis of it [1].

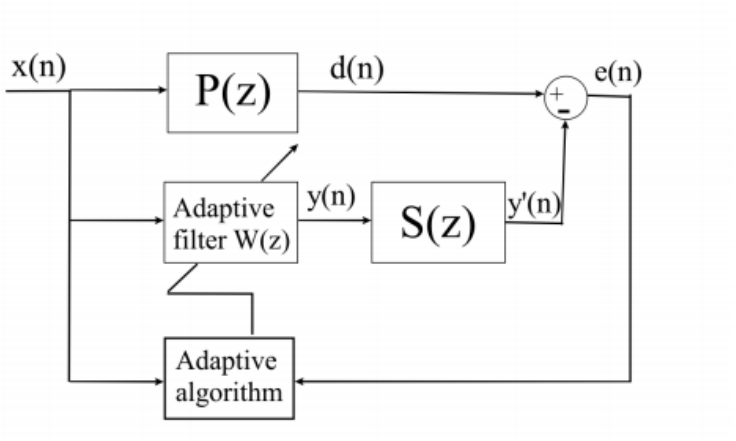

Figure 2.5: Adaptive filter approach

- The alternative solution, which gave better results is the Continuous Transfer Functions approach, which consists in the identification of the primary and secondary paths as high order transfer functions and the design of a feed forward controller fittingly, consisting of the division between those two transfer functions (-secondary path/primary path) as well as a feedback PID controller, designed based on the structure of the secondary path.

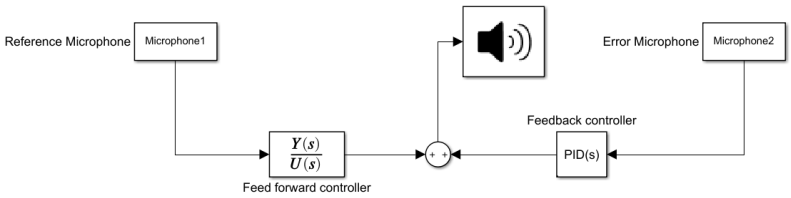

Figure 2.6: Transfer functions approach

2.3 BLACKMAN-HARRIS WINDOWING

A window function is a mathematical function zero-valued outside a chosen interval, symmetrical around the middle and usually near a maximum in the middle. When another data sequence is multiplied by a window function, the product is also zero valued outside the interval, in such a way that multiplying a function by a window function gives a window sized look into that function. In typical applications, the window functions used are non-negative, smooth, "bell-shaped" curves.

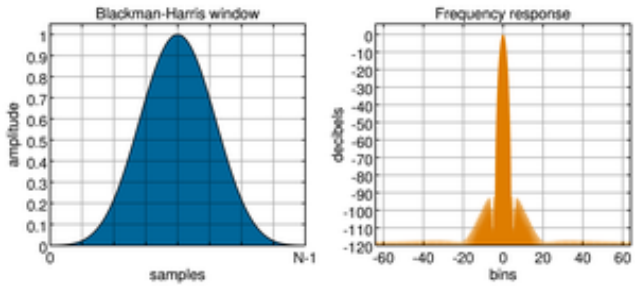

Figure 2.7: Blackman-Harris windowing

The Blackman-Harris [5] variant is a generalization of the Hamming family, produced by adding more shifted synchrony functions, meant to minimize side-lobe levels. It has a bell shaped curve, and is meant to make the more extreme sides of the function meet at zero, simplifying the sound signals thus reducing the number of harmonics in the signal and preventing truncation for the fourier transform, which requires the signal to be continuous and derivable.

2.4 COMPLEX CURVE FITTING E.C. LEVI

Curve fitting is the process of constructing a line, a curve with its mathematical function in order to best approximate a series of data points, similar to a linear regression, the latter being used more often in finding statistical uncertainty. In this thesis the use meant for curve fitting is approximating transfer functions into their mathematical equations.

The Levi Method [6], also referred to as the least squares method, makes the identification of the coefficients in a least squares sense, which allows for the minimisation of the residual data points. The numerical difference between the frequency response to be fitted and the model attempting to fit it is represented as the error to be reduced.

It is possible to give further weight to certain data points in order to make the function more closely approximate the frequency response, giving more weight to peaks and valleys and taking weight away from abnormalities.

[6]

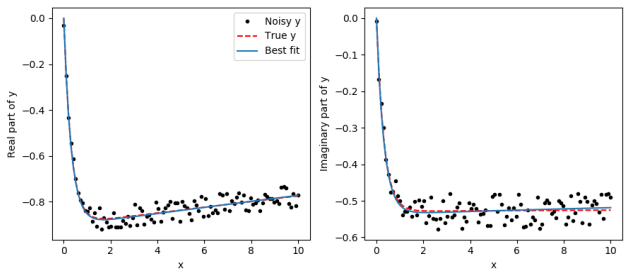

Figure 2.8: Example of curve fitting

2.5 HARDWARE

The necessary hardware for the implementation of the system consists of:

THE PROCESSING CORE: The processing core will be responsible for receiving the audio signal, processing it and sending the disturbance noise signal to the speakers. It must have a good computing power, a competitive price and high versatility. The dispositive chosen, as explained in [1], is the Raspberry Pi 4 Model B board[11] for it's competitive price (around 60 dollars) and the specifications:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 4GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage

- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient

This will allow for high speed processing, the for USB entries will allow for recording from two mics while using a keyboard and a mouse, it can access internet to download updates and look up script examples, it has a good sound card and it can run in a free operating system (Raspbian Linux distribution).

Initially, the processing core used was a simpler Raspberry 1 B, with access to internet, ports and the same software, but much slower. It was inherited from the student that proposed the architecture. An upgrade was required, as the previous core was unable to simultaneously input and output audio of any quality simultaneously without losing too much speed to make any ANC process causal, except for a simple noise inverting program.

THE TWO MICROPHONES: The microphones used are two "Logitech USBDesktop Noise" each costs around 30 euros. No software installation is required. It is very sensible, being able to pick up quiet speech from 30 cm away. It's tiltable, allowing for different positioning to receive sound. [10]

THE SPEAKERS: A standard set of desktop speaker should be sufficient for this task. The speaker used is not produced anymore, the model is NGS SOUNBAND, and used to cost around 10 euros.

2.6 SOFTWARE

The programming language used will be python. It's a high level programming language, very compact and with a lot of versatility, but not as fast as other languages like C. It allows for all the calculations necessary with a variety of libraries.

A fundamental part of this project is understanding how to read audio from more than one source simultaneously, as well as being able to output the processed audio at the same time. The python library pyaudio will be a useful tool in that pursuit [12].

In order to read audio, the library requires the creation of a stream, which will be able to interact with the audio input and output devices, reading and/or writing a chunk of data, usually 4096 units. It needs to have a set of parameters defined, the fundamental ones are the audio format, number of channels, sample rate, amount of frames per buffer, index device and whether it will input, output or both. By making use of those streams, sets of data can be read, be processed instantly and saved in a list. With this same format, audio can be sent to the output devices (speakers).

In order to read from two microphones simultaneously, two streams should be created with different device indexes referring to each microphone. If the objective of the project required perfectly simultaneous recording, that could be achieved creating and coordinating two threads, but that would consume a big amount of computing power, of which there is a limited supply. After some testing, the conclusion has been reached that the simple superposition of the streams is fast enough to be almost simultaneous, relieving the need for separate threads. It is, however, possible to simultaneously read and write from a single stream, speeding up the execution of the script. For the purposes of this project, the parameters for audio reading and writing will be the standard sample rate of 44100 Hz, which allows not to lose any useful information in sampling and satisfies the Nyquist frequency criterion.

If a stream was reading audio and received a new reading request, the program would stop automatically and print out an error. Fortunately, it's possible

```
import pyaudio
import numpy as np
#format
form_1 = pyaudio.paInt16
#channels
chans = 1
#sample rate
samp_rate = 44100
#chunk
chunk = 4096
#seconds recording
record_secs = 5
dev_index = 2


audio = pyaudio.PyAudio()

stream = audio.open(format = form_1, rate = samp_rate, channels = chans, \
                    input_device_index = dev_index, input = True, \
                    frames_per_buffer = chunk)
print("recording")
frames = []
signal = []
for ii in range(0,int((samp_rate/chunk)*record_secs)):
    data = stream.read(chunk)
    signalData = np.fromstring(data,dtype=np.int16)
    frames.append(data)


print("finished recording")

stream.stop_stream()
stream.close()
audio.terminate()
```

Figure 2.9: simple audio reading program

to get around that inconvenience through the command "exceptiononoverflow = False", which will allow those errors to be ignored and keep the program running, if with some cuts. Luckily, with a chunk size of 4096 only 10 % of a second worth of signal would be lost. This command is used in all noise cancelling scripts created in this project. The audio enters the stream in the form of a byte array, which can be translated into INT16 format, allowing for manipulation of the data. It is possible to accumulate ("append") those signals, and save them for conversion into an audio file at the end of the stream. Most of those conversions are done through numpy, which allows for a large amount of data conversions and operations.

2.7 LIMITATIONS

The simple principle of noise cancellation is complicated by real world factors.

- In an ideal environment, the sound would only flow in one direction, with walls all around. The fact that this is in an open space, with static sound of computers, or unrelated noises in the case of the library, complicates the system reducing the sound cancelling qualities of the system.
- The microphones and the speaker, even though of decent quality, are limited, the speakers set in particular.
- The system will react non linearly to different distances from the noise source.
- For active noise cancelling to be achievable, the program needs to output sound at least as fast as the time sound takes to travel the distance between the reference and error microphones. Failing that, the program would just overlap with the original sound, generating a louder overall signal, and would only be able to reduce low frequency narrow band noises. The original processing unit was dropped because of this.

3 IDENTIFICATION OF THE SYSTEM

Given the current global health conditions and safety measures, the experimental work has been done in a smaller simplified environment. The experiments will be done with a box to act as the walls of a room, with points for every dispositive set so that the dispositives always go in the same place. The set up will be the same as it would have been in the library, be it smaller, with the noise signal at a point inside the box, the reference microphone next to it, the speakers meant to send a disturbance noise out of the box, disposed in the direction of the error microphone, and the error microphone itself, which will be further away directed to the sound signals. If the dispositives remain the same, the system to study should be constant.

Figure 3.1: Experimental setup

The transfer functions approach requires the identification of the transfer function of the primary and secondary path. For that purpose, the audio will be acquired from the raspberry and processed with matlab. The functions will be windowed, translated in the frequency domain, then the transfer function will be identified and converted into a mathematical one that closely resembles

the original one. Those high order transfer functions will allow for the design of a controller.

3.1 PRIMARY PATH IDENTIFICATION

Figure 3.2: Audio reading system layout

In order to be able to design a fitting controller both the primary and the secondary paths need to be identified. The audio files necessary to identify the primary path (the transfer function between two microphones) are obtained by means of reading the input of both microphones at a time while playing a “talking noise” audio file from the speakers placed in the noise source. The transfer function, referred to as H(jw), will be the one to come out of the division between the two signals, the one to come out of the reference microphone, U(jw), and the one to come out of the error microphone, Y(jw).

$$H(jw) = \frac{Y(jw)}{U(jw)} \tag{3.1}$$

The python program used to record from two microphones hardly difeers from the example, recording 60 seconds of audio from both micropones simultaneously, thus needing two strings, saving all the recorded data in two lists and finally sending them in different wave files. The device indexes are two and

three because the board is configured in such a way that the first microphone added is considered the device number two.

```
import pyaudio
import wave
import matplotlib.pyplot as plt
import numpy as np

#declaring record parameters
form_1 = pyaudio.paInt16
chans = 1
samp_rate = 44100
chunk = 4096
record_secs = 60


audio = pyaudio.PyAudio()

#start stream
stream = audio.open(format = form_1, rate = samp_rate, channels = chans, \
                    input_device_index = 2, input = True, \
                    frames_per_buffer = chunk)

stream2=audio.open(format = form_1, rate = samp_rate, channels = chans, \
                    input_device_index = 3, input = True, \
                    frames_per_buffer = chunk)

print("recording")
#create lists
frames = []
signal = []
#start recording

for ii in range(0,int((samp_rate/chunk)*record_secs)):
    data = stream.read(chunk)
    signalData = np.fromstring(data,dtype=np.int16)
    frames.append(data)
    data2 = stream2.read(chunk)
    signalData = np.fromstring(data, dtype=np.int16)
    signal.append(data2)
```

Figure 3.3: audio recording program

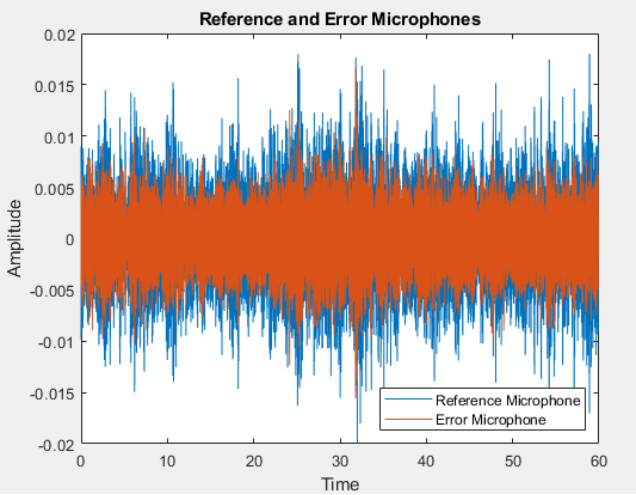

Figure 3.4: microphone amplitudes

In order to find the transfer function, a frequency domain identification procedure is required using the Blackman-Harris window and applying the fast fourier transform [5]. As explained before, the fourier transform function requires a continuous and derivable signal, and the Blackman Harris, in this case used as a matlab function, reduces the number of points in which the signal has any kind of discontinuity.

Getting rid of anomalies and focusing in on the relevant frequencies the difference between the two signals can be seen 3.6. As expected, the higher frequencies are more easily absorbed by the walls, making them more prevalent in the reference microphone than in the error microphone.

In order to find the transfer function, the signals are divided. To simplify the function and eliminate anomalies, only one in twelve signals are taken. The function is robust enough to not be undefined by this procedure.

Using the simplified version of the transfer function, the invfreqs function from matlab is used, based on the works of E.C.Levy on Complex Curve fitting [6], explained before. For this project, it is required that the functions are also stable, which is requirable through the use of it.

This function, given the source signal, the order in which it has to be approx-

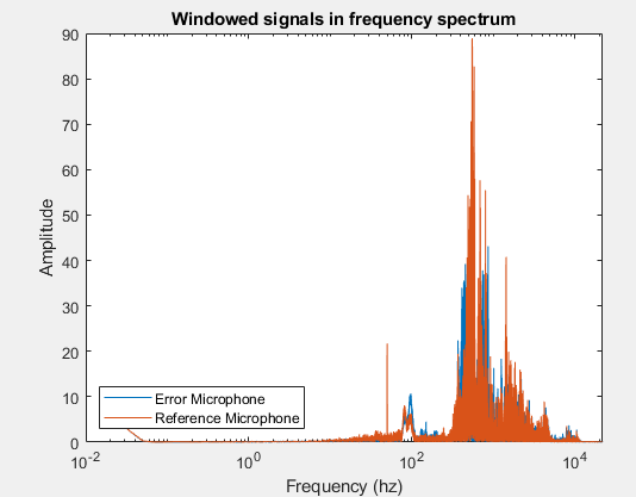

Figure 3.5: Microphone amplitudes in frequency spectrum

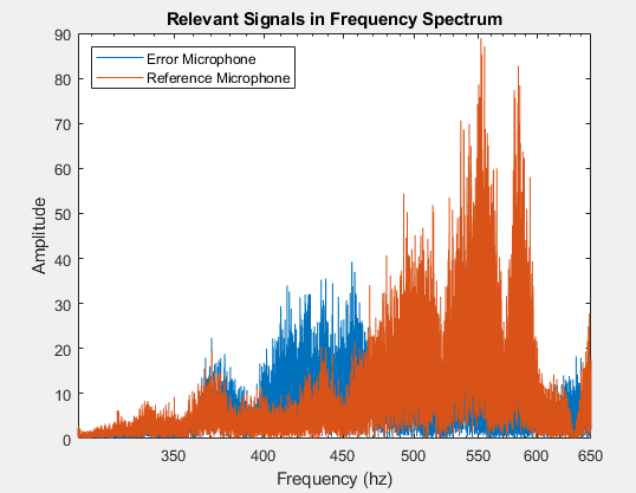

Figure 3.6: Microphone signals in relevant frequencies

imated and the weights of different points in the signal, does a mathematical approximation of it of varying accuracy depending on the input. The best approach to finding the best fitting function is giving the peaks and valleys a higher weight, to find the anomalous signals and weight them at zero and to try a combiantion of different weights and different numbers of poles and zeros.

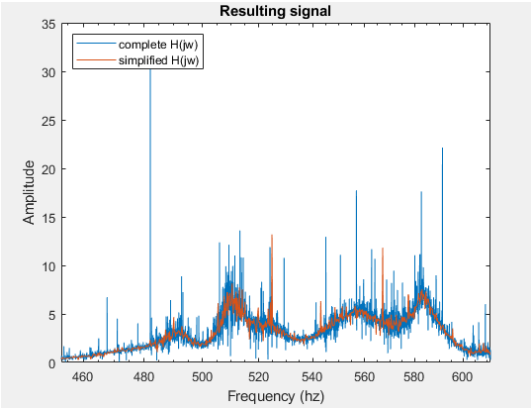

Figure 3.7: Primary path transfer function

The function has to be as small as possible. A video is linked with a number of attempts [9]. Some examples of non well fitting functions are added 3.8 3.9.

```
weights=ones(size(f3));
weights([8 135 194 232 287 352 418 518 594 652 709])=1e3;

weights([ 571 622 623])=0;
[B,A] = invfreqs(H3,2*pi*f3,30,30,weights, 0, 1);
```

Figure 3.8: Weights vector

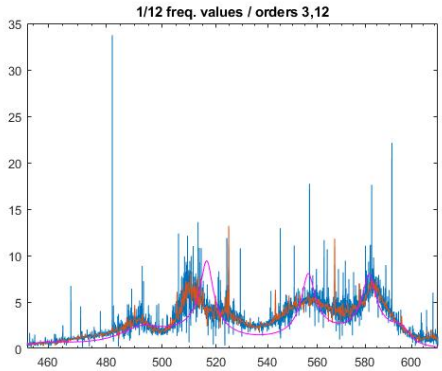

Figure 3.9: primary path curveffiting with order 3, 12

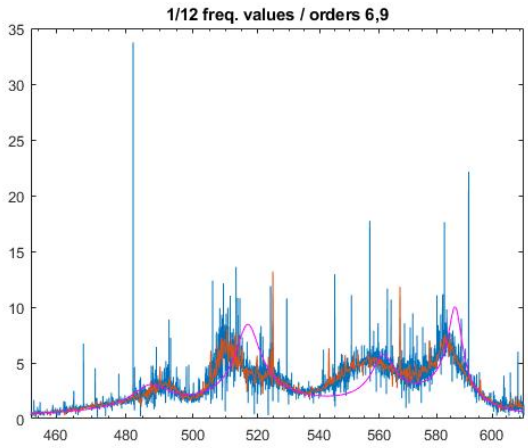

Figure 3.10: primary path curveffiting with order 6, 9

The chosen function to approximate this signal is one with sixteen poles and six zeroes. it approximates the original signal very closely on the most relevant points 3.11.

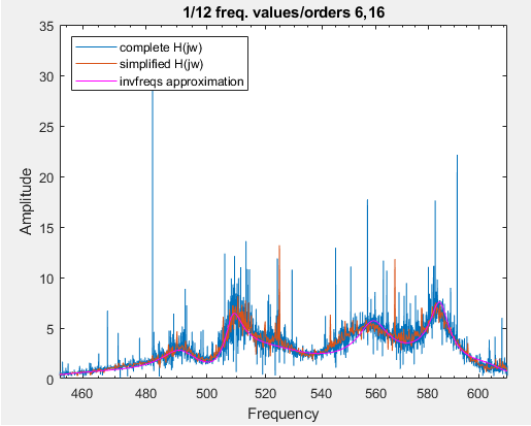

Figure 3.11: definitive primary path transfer function, orders 6, 16

$$Numerator = 6.741e31s^6 - 1.519e35s^5 + 3.362e39s^4 - 3.746e42s^3 + 4.853e46s^2 -$$

$2.213e49s + 2.165e53$

$Denominator = s^{16} + 6844S^{15} + 9.558e07s^{14} + 5.559e11s^{13} + 3.92e15S^{12} + 1.924e19s^{11} + 9.016e22S^{10} + 3.677e26s^9 + 1.273e30s^8 + 4.193e33s^7 + 1.129e37s^6 + 2.853e40s^5 + 6.142e43s^4 + 1.072e47s^3 + 1.871e50s^2 + 1.717e53s + 2.439e56$

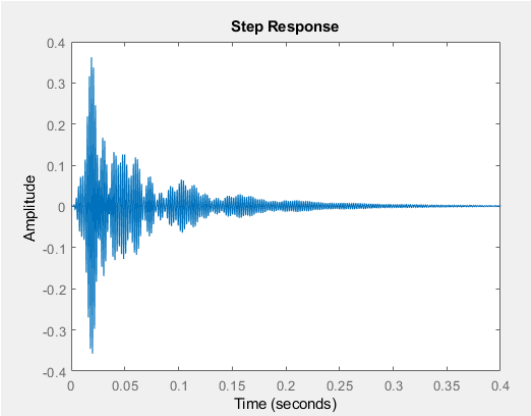

Figure 3.12: step response primary path transfer function

3.2 SECONDARY PATH IDENTIFICATION

In order to find the transfer function for this path, a multisine function is simultaneously sent through the anti noise source speakers and recorded by the error microphone, and the signals are compared.

Figure 3.13: Audio reading system layout

The program used for this process is a little more complicated. It requires two streams, but one of them is an output stream, which will allow to write the audio file "crest16.wav" with a multisine wave one chunk at a time. However, this program is run with the Raspberry Pi 1 board, and it is impossible to get any quality of sound without increasing the chunk to this kind of number, which impacts negatively the speed of both the reading and the writing.

Both streams are put in a loop and made to read and write the same amount of frames simultaneously. The data of both is saved in wav files in order to compare them later.

```
import pyaudio
import wave
import numpy as np

f = wave.open("crest_16.wav","rb")
form_1 = pyaudio.paInt16
chans = 1
samp_rate = 44100
chunk = 44100
dev_index = 2
wav_output_filename= 'output_multisinus.wav'

p = pyaudio.PyAudio()

stream = p.open(format = form_1, rate = samp_rate, channels = chans, \
                    input_device_index = dev_index, input = True, \
                    frames_per_buffer = chunk)

stream2 = p.open(format = p.get_format_from_width(f.getsampwidth()),
                channels = f.getnchannels(),
                rate = f.getframerate(),
                output = True)


data2 = f.readframes(chunk)
frames=[]
signal=[]
print("going")
while data2:
    stream2.write(data2)
    data2 = f.readframes(chunk)
    data = stream.read(chunk)
    signalData = np.fromstring(data, dtype=np.int16)
    frames.append(data)
    signal.append(data2)

print("finished going")


stream.stop_stream()
stream2.stop_stream()
stream.close()
stream2.close()
p.terminate()
```

Figure 3.14: multisinus reading and writing python script

The signals are of very different amplitudes, as one comes from a microphone and the other from a .wav audio file. The original signal as shown in 3.15 in blue is the original one.The signal in orange, taken by the microphone, is much more irregular. 3.16 It has been recorded with the raspberry 1, and serious latency issues are evident, as the board is trying to output and input audio simultaneously. It also records in irregular chunks of data, getting overwhelmed at times. When translated in the frequency domain, those issues will

be mitigated but not erased.

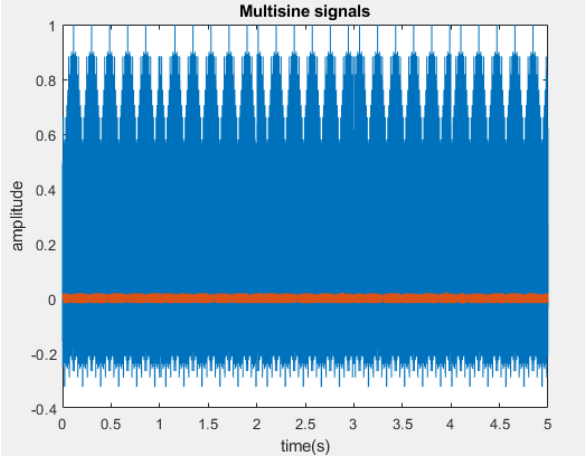

Figure 3.15: Multisine signals in time domain

Figure 3.16: microphone signal zoom

The comparison is done in a frequency spectrum, using the discrete fourier transform, and blackman-harris windowing [7] is applied again. To avoid low level static noise, only the peaks are taken into account for the comparison, as shown in 3.17.

The resulting signals are similar in the way the amplitudes appear in the selected frequencies,suggesting an improvement from the signals in time domain, though the higher frequencies are clearly mitigated in the microphone signal. In order to find the transfer function of the secondary path, the orignial signal will be divided by the microphone signal, resulting in the signal shown in figure 3.20.

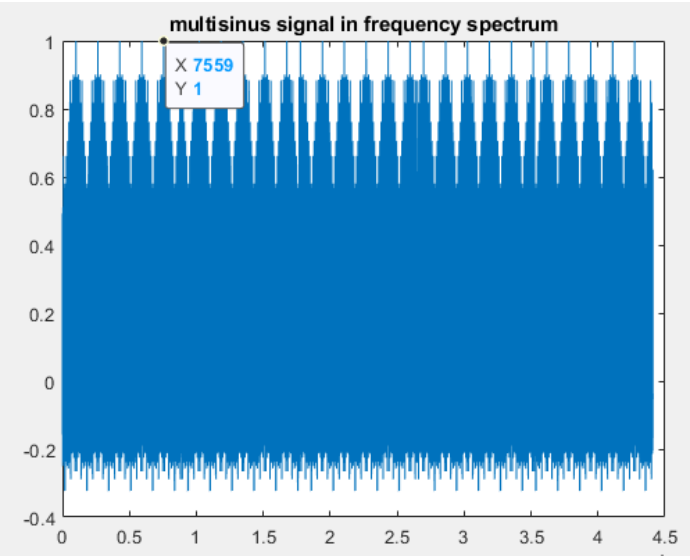

Figure 3.17: Multisine signal in frequency domain

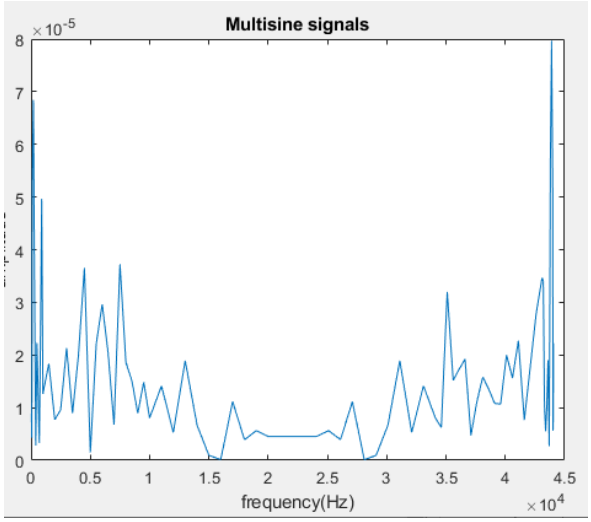

Figure 3.18: original multisine signal in frequency domain in selected frequencies

The same curve fitting matlab function is applied in order to find the mathematical approximation of the transfer function. The final result is a transfer function with 11 zeros and 12 poles.

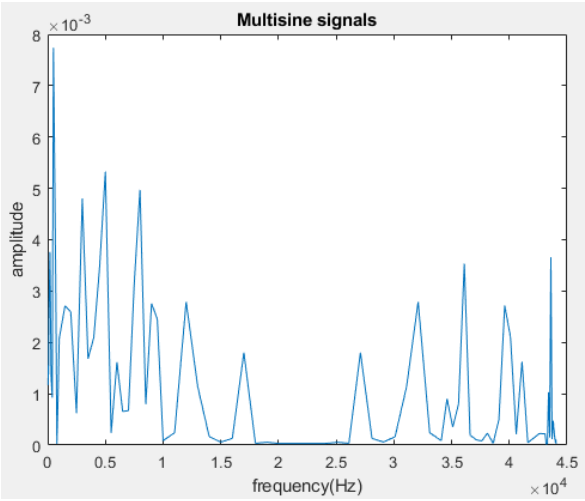

Figure 3.19: microphone multisine signal in frequency domain in selected frequencies

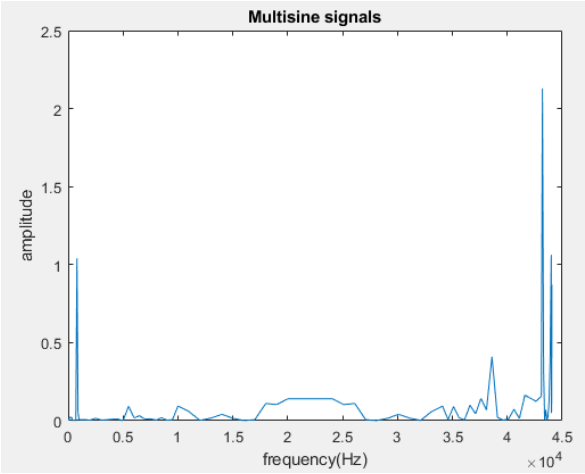

Figure 3.20: transfer signal in frequency domain in selected frequencies

$$Numerator = 3351s^{11} - 5.144e07s^{10} + 7.322e14s^9 + 6.692e18s^8 + 5.615e25s^7 + 1.855e30s^6 + 1.752e36s^5 + 1.022e41s^4 + 1.842e46s^3 + 1.713e51s^2 + 8.304e54s + 2.122e60$$

$$Denominator = s^{12} + 2.625e04s^{11} + 2.472e11s^{10} + 5.84e15s^9 + 2.289e22s^8 + 4.688e26s^7 + 9.698e32s^6 + 1.601e37s^5 + 1.825e43s^4 + 2.014e47s^3 + 1.21e53s^2 + 3.423e56s + 4.116e60$$

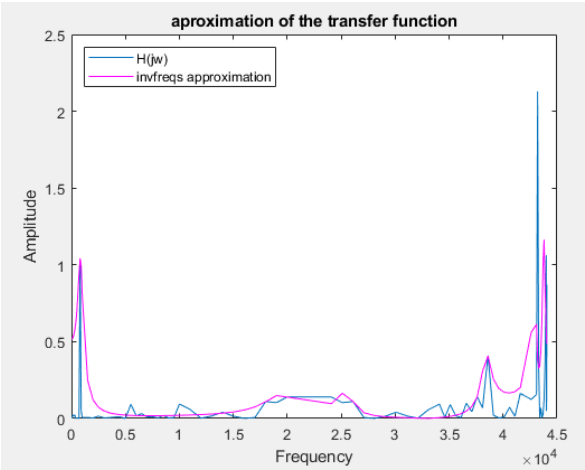

Figure 3.21: secondary path transfer function approximation

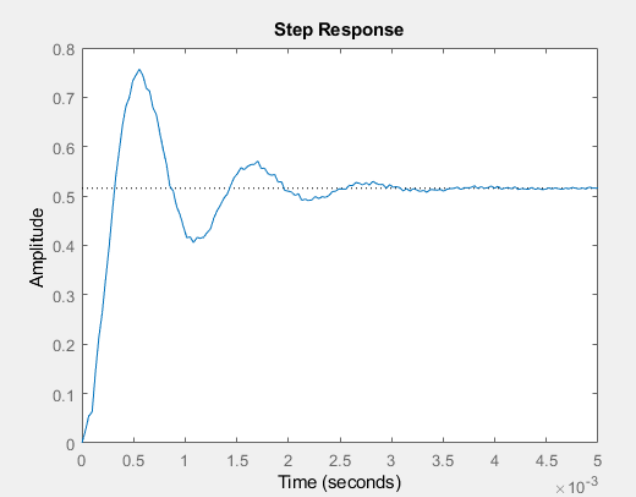

Figure 3.22: step response

The transfer function will be used in both controller identifications, and its correct performance will be vital.

4 CONTROLLER TUNING

4.1 FEED FORWARD CONTROLLER

In order to tune the feed forward controller, the inverse of the primary path transfer function is divided with the secondary path transfer function. This results in a stable transfer function with 6 zeros and 16 poles.

$Numerator = -6.741e31s^6 + 1.519e35s^5 - 3.362e39s^4 + 3.746e42s^3 - 4.853e46s^2 + 2.213e49s - 2.165e53$

$Denominator = s^16 + 6844s^{15} + 9.558e07s^{14} + 5.559e11s^{13} + 3.92e15s^{12} + 1.924e19s^{11} + 9.016e22s^{10} + 3.677e26s^9 + 1.273e30s^8 + 4.193e33s^7 + 1.129e37s^6 + 2.853e40s^5 + 6.142e43s^4 + 1.072e47s^3 + 1.871e50s^2 + 1.717e53s + 2.439e56$

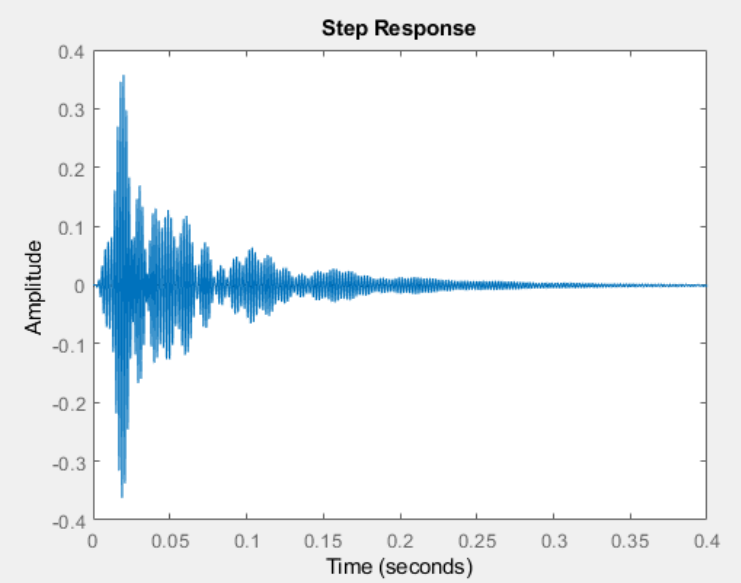

Figure 4.1: feed forward controller step response

4.2 FEED BACK CONTROLLER

There are several methods to tune the PID response of the secondary controller, but the one used in this project is the PID autotuning open loop simulink tool. It requires to set up a system such that the PID feeds a constant into the autotuning block, the block in turn gives output to the transfer function meant to be the plant of the system (the secondary path in this case) and if activated, at the end of the tuning it will provide the parameters for the PID controller depending on different configurable parameters inside the block itself, which are type, form, domain, sample time, and method.

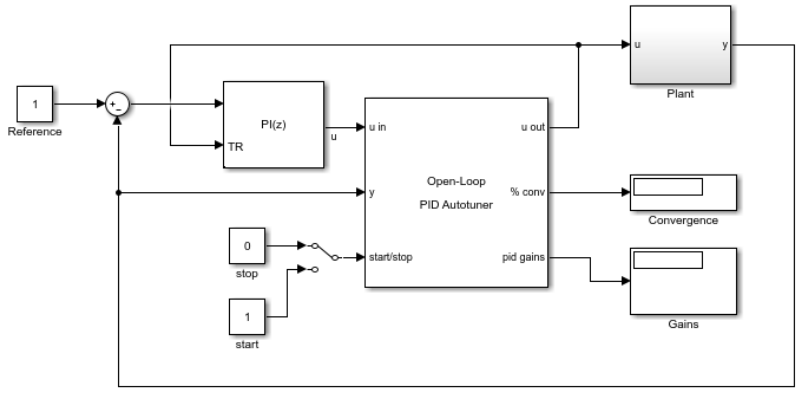

Figure 4.2: PID autotune layout

The type used in this project will be a PI controller, as the derivative action tends to get descontrolated if the input is too variable, which is the case. The form used is parallel, the sample time 0.02, the approximate number of 1024 size chunks to be read in a second, it is done in time domain and with the forward euler integrator method.

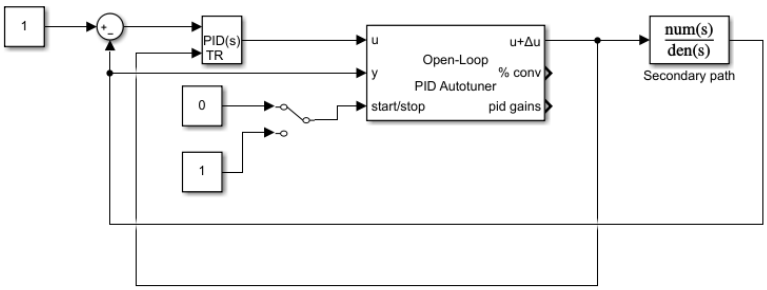

Figure 4.3: my PID autotune layout

The resulting PI controller has a KP parameter of 0.4548 and a KI parameter of 1 s^{-1}.

5 SIMULATION

Before implementing the obtained results into the building of a program for the raspberry pi, it will be tested in a simulated environment using the Matlab tool Simulink. The sound signal of people talking, used in the identification of the primary path is introduced and filtered through the primary and secondary paths in the same way it would be in the real world, and then the controllers are implemented.

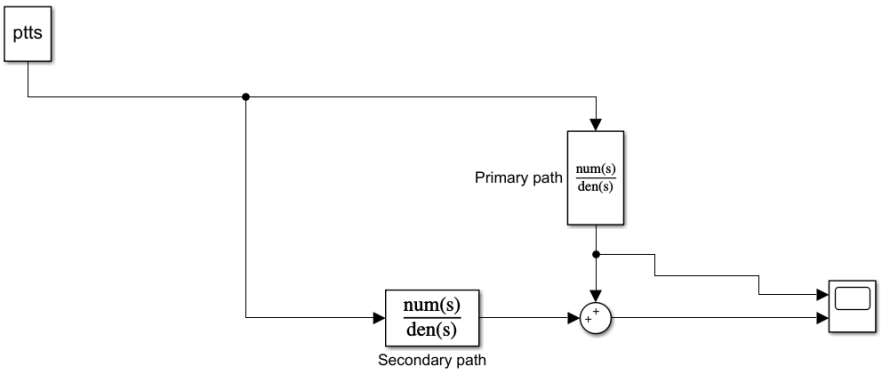

Figure 5.1: simulation of non controlled noise

The controllers are implemented, the feed forward controller coming from the sound signal and the feedback controller coming from the result of the previous iteration. The sum of the resulting signals, processed through the secondary path as it would be in the real world is added to the original sound signal, reducing the overall sound level.

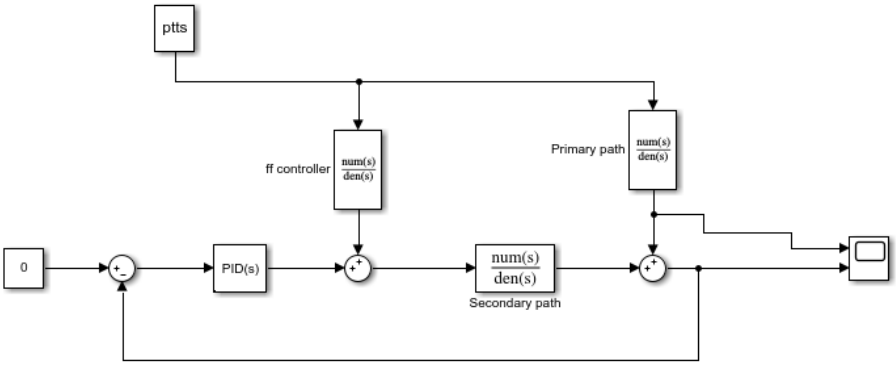

Figure 5.2: simulation of controlled noise

As can be seen in 5.3 , the results are satisfactory, with the blue signal representing the sound filtered through the primary path and the orange one being the reduced sound, reduced by the action of the controllers. The average resulting amplitude in the error microphone sound is reduced to a 35% of the original.

Kp	Ki(s^{-1})	Reduction(%)
0.4548	1	65
0.4548*2	1	73
0.4548	10	64
0.4548*0.5	1	63
0.4548*0.5	10	62

Different parameters of KP and KI for the PI controller have been tried in order to ensure the responsivity of the controller within the system.

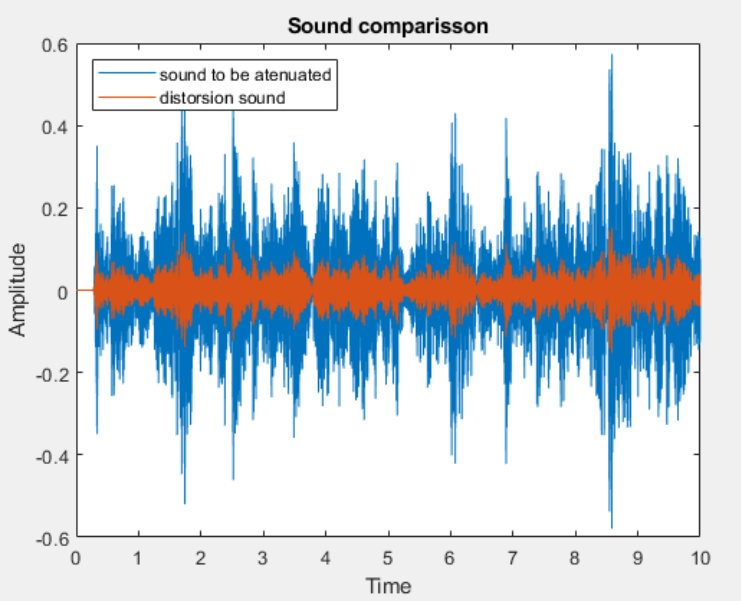

Figure 5.3: results of the simulation

The different stable KP and KI parameters can still be tried in the real system.

6 IMPLEMENTATION OF THE SYSTEM

As explained before, the system is implemented in a raspberry pi 4 B through a python program, with the help of several libraries.

The system is tried as it was set up, with the reference microphone inside the box, working as the room, and the error microphone and speakers in the same position as in the identification, outside the box.

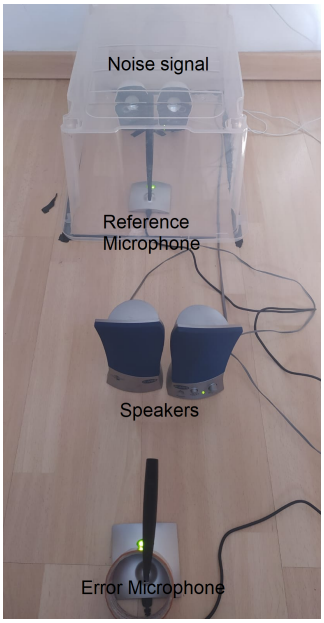

Figure 6.1: implementation layout

As explained previously, the processing core used initially is a Raspberry Pi 1. This board worked well for the first path identification, dubiously for the secondary path identification but proved unusable in the implementation of the desired system.

Various speeding up effects have been tried, including:

- Limiting the number of streams: Each stream addition adds load to the

program. Uniting input and output in a single stream takes computing weight off the program. This is kept in the final program.

- Adding different strings for different streams: It is possible to record using strings, that is dividing the program into separate lines of code, each one doing different functions simultaneously. In this case, strings were created for each microphone. This approach, while good for speeding up microphone recording, had a set of problems. The streams rarely communicated between each other, and in the event that they did it was once in about six loops, making the output repeat itself until it got an input change. It is possible to make strings wait for each other, but that slowed down the process worse than having them together in one loop.

- Reducing the program to solely feedforward or feedback: that gave a significant improvement but at cost of half the system.

- Increasing the chunk size: this increased the quality but reduced the speed, making the feedback unusable.

- Reducing the number of calculations: the feed forward controller, being a order 16 transfer function slowed down the process, and it was significantly simplified. This is kept in the definitive program

The conclusion achieved is that the first board is only able to run a simplified feedforward inverter script with any quality and possibility of audio cancellation. 6.2

At this point, the decision was made that a better processing core was needed. The raspberry Pi 4, with much better processing speed proved to be able to run the system much better.

```
import pyaudio
import numpy as np
import sys

w =signal.TransferFunction(num, den)
# Size of each read-in chunk
CHUNK = 1024
# Amount of channels of the live recording
CHANNELS = 1
# Sample width of the live recording
WIDTH = 2
# Sample rate in Hz of the live recording
SAMPLE_RATE = 44100

pa = pyaudio.PyAudio()


def invert(data):
    # Convert the bytestring into an integer
    intwave = np.fromstring(data, np.int16)
    # Invert the integer
    bd = np.invert(intwave)
    # Convert the integer back into a bytestring
    inverted = np.frombuffer(intwave, np.byte)
    # Return the inverted audio data
    return inverted

stream = pa.open(
    format=pa.get_format_from_width(WIDTH),
    channels=CHANNELS,
    rate=SAMPLE_RATE,
    frames_per_buffer=CHUNK,
    input=True,
    output=True
)

print("donne")

for i in range(0, int(SAMPLE_RATE / CHUNK * sys.maxunicode)):
    original = stream.read(CHUNK, exception_on_overflow=False)
    inverted = invert(original)
    stream.write(inverted, CHUNK)
stream.stop_stream()
stream.close()
pa.terminate()
print("donne")
```

Figure 6.2: simple inverting program

6.1 PYTHON PROGRAM

This section contains a brief explaination of the way the active noise cancelling program operates.

The program is built with two streams, one with both input and output, that will take care of the reference microphone and the speakers, and the other one only for input. The sample rate is 44100Hz, as explained before, and the chunk is reduced to 1024

The program is built with two streams, one with both input and output, that will take care of the reference microphone and the speakers, and the other one only for input. The sample rate is 44100Hz, as explained before, and the chunk is reduced to 1024, allowing for a fast reaction from the feedback controller. The KP and KI are declared as found previously. Two lists are also declared.

It has been attempted to use python libraries to implement the controllers in a more straightforward manner. However, they require an input of a single value per time and measuring the array of data given by a chunk is not possible for them. It is also not viable to get the chunk size to one, as that would require the program to start reading, processing and writing sound commands 44100 times per second.

The loop, meant to play the number of times required by the chunk size and the sample rate each second, about 43, hosts all the audio readings, writings and calculations. Firstly, the two microphones are read simultaneously, and then each is processed through their respective controller, feedforward for the reference microphone and feedback for the error microphone.

Given the fact that specific PID and Transfer function libraries were discarded for the building of the controllers, they are implemented in a discrete fashion, taking in chunks of data instead of individual data points. That is why the lists are built, not only can they be used to save the recorded audio, they are necessary for the PID controller to take into account previous measures. The feed forward controller has been simplified to the independent factor, and all that is required is for the data to be converted into the int 16 format, before and after the calculations, as the multiplication turns the number into a float.

```
# Size of each read-in chunk
CHUNK = 1042
# Amount of channels of the live recording
CHANNELS = 1
# Sample width of the live recording
WIDTH = 2
# Sample rate in Hz of the live recording
SAMPLE_RATE = 44100
#PID parameters
KP=0.45484
KI=(1/44100)
k=0
M=[]
E=[]

pa = pyaudio.PyAudio()



stream = pa.open(
    format=pa.get_format_from_width(WIDTH),
    channels=CHANNELS,
    rate=SAMPLE_RATE,
    frames_per_buffer=CHUNK,
    input=True,
    output=True
)

stream2 = pa.open(
    format=pa.get_format_from_width(WIDTH),
    channels=CHANNELS,
    input_device_index=2,
    rate=SAMPLE_RATE,
    frames_per_buffer=CHUNK,
    input=True,

)
```

Figure 6.3: First half of the definitive ANC program

The feedback controller is built as seen in figure 6.5 , except the derivative part, which is 0 in our system. It requires the same conversions as the previous one, but it also requires to save the previous iteration of the PID action.

```
M.append(0)
E.append(0)
for i in range(0, int(SAMPLE_RATE / CHUNK * sys.maxunicode)):
    k = k + 1

    #read from reference microphone
    reference = stream.read(CHUNK, exception_on_overflow=False)
    error = stream2.read(CHUNK, exception_on_overflow=False)
    intwave = np.frombuffer(reference, np.int16)
    intwave = np.invert(intwave)*0.6
    intwave = intwave.astype(np.int16)
    ffed = np.frombuffer(intwave, np.byte)

    intwav = np.fromstring(error, np.int16)
    M.append(intwav)
    intwav = KI*intwav+M[k-1]+KP*(intwav-E[k-1])
    E.append(intwav)
    intwav=intwav.astype(np.int16)
    fbed = np.frombuffer(intwav, np.byte)
    stream.write(ffed+fbed, CHUNK)

stream.stop_stream()
stream.close()

stream2.stop_stream()
stream2.close()

pa.terminate()
```

Figure 6.4: Second half of the definitive ANC program

$$mv(k)=mv(k-1)+K_p'\cdot(e(k)-e(k-1))+K_i'\cdot e(k)+K_d'\cdot(e(k)-2\cdot e(k-1)+e(k-2))$$

Figure 6.5: PID discretisation principle

The program can be downlaoded and used free of charge from GitHub [13]

6.2 Results

In order to evaluate the performance of the system, the cancelling noise program is run during ten seconds with a constant sine wave of different frequencies coming from the "noise" generating speakers inside the box. The frequencies chosen are 400, 500, 700 and 1000Hz. Two programs have been run alongside the standard active noise cancelling one. Those two new programs run only one of the two functions, feedback or feedforward. This gives the advantage of processing faster, and also separating their actions, giving a better understanding of the system. A table is shown giving the reduction in percentage and decibels of different configurations for signals in different frequency ranges, with a positive % reduction and a negative decibel number meaning a reduction in the noise . The signals that produce reduction are highlited in green.

	400Hz		500Hz		700Hz		1000Hz	
Reduction	db	%	db	%	db	%	db	%
feedforward	-1.791	32.921	-1.647	31.56	-	-43.02	-0.1850	4.13
feedback	-	-1084	-	-349.9	-	-1061.4	-	-979.67
feedforward + feedback	-1.686	32.18	-0.1791	4.04	-0.1493	7.38	-	-54.65

Figure 6.6: Table of results

The results are very variable, with the feedback+feedforward script giving the most consistent value, the only feedback one being consistently wrong and the feedforward giving decent results but lacking consistency. As can be seen in the figures, the reduction is also inconsistent with time, with peaks and valleys, and specially the 400 hz one giving very good value and a sign of better reduction over time.

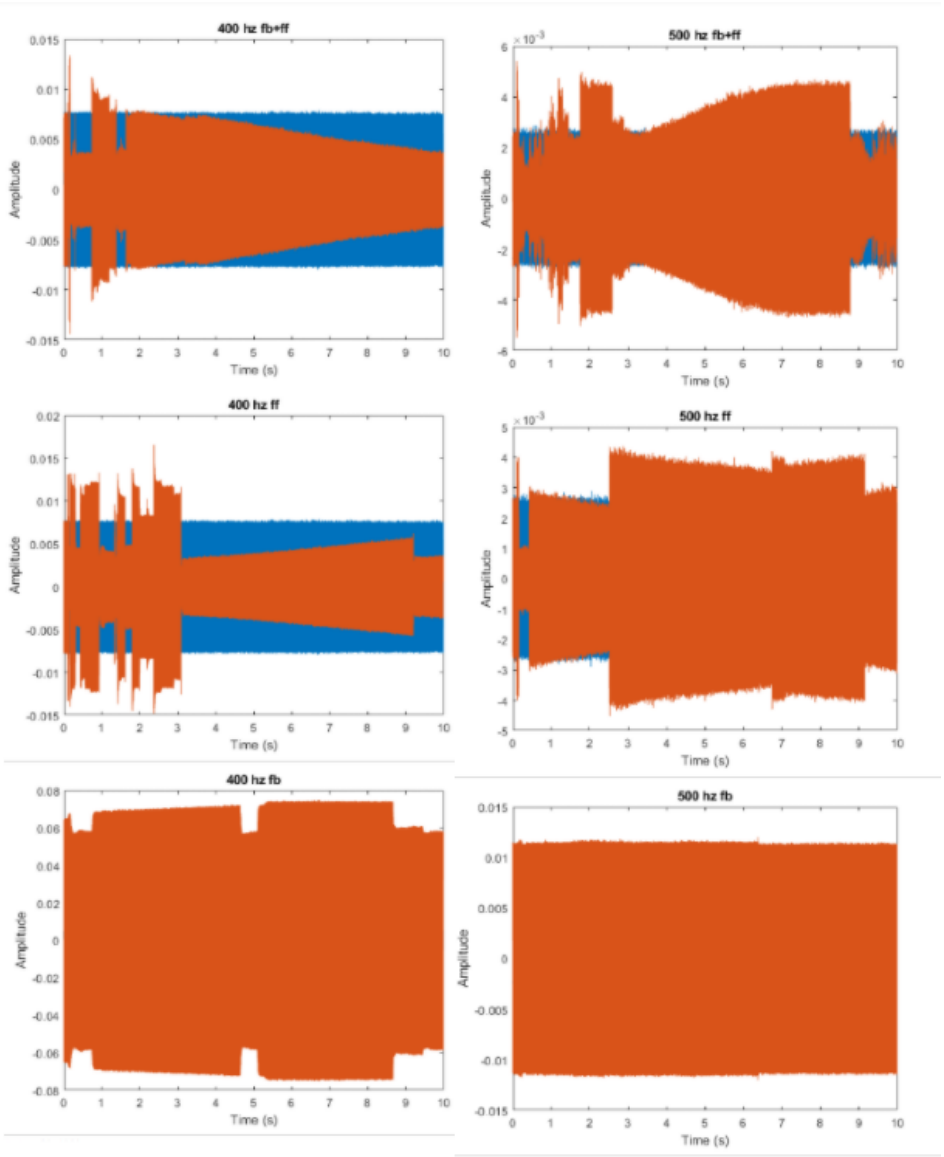

Figure 6.7: Signals 400 and 500 Hz

The figures included show the microphone reading without any noise controlling system in place in blue, and the altered sound in orange.

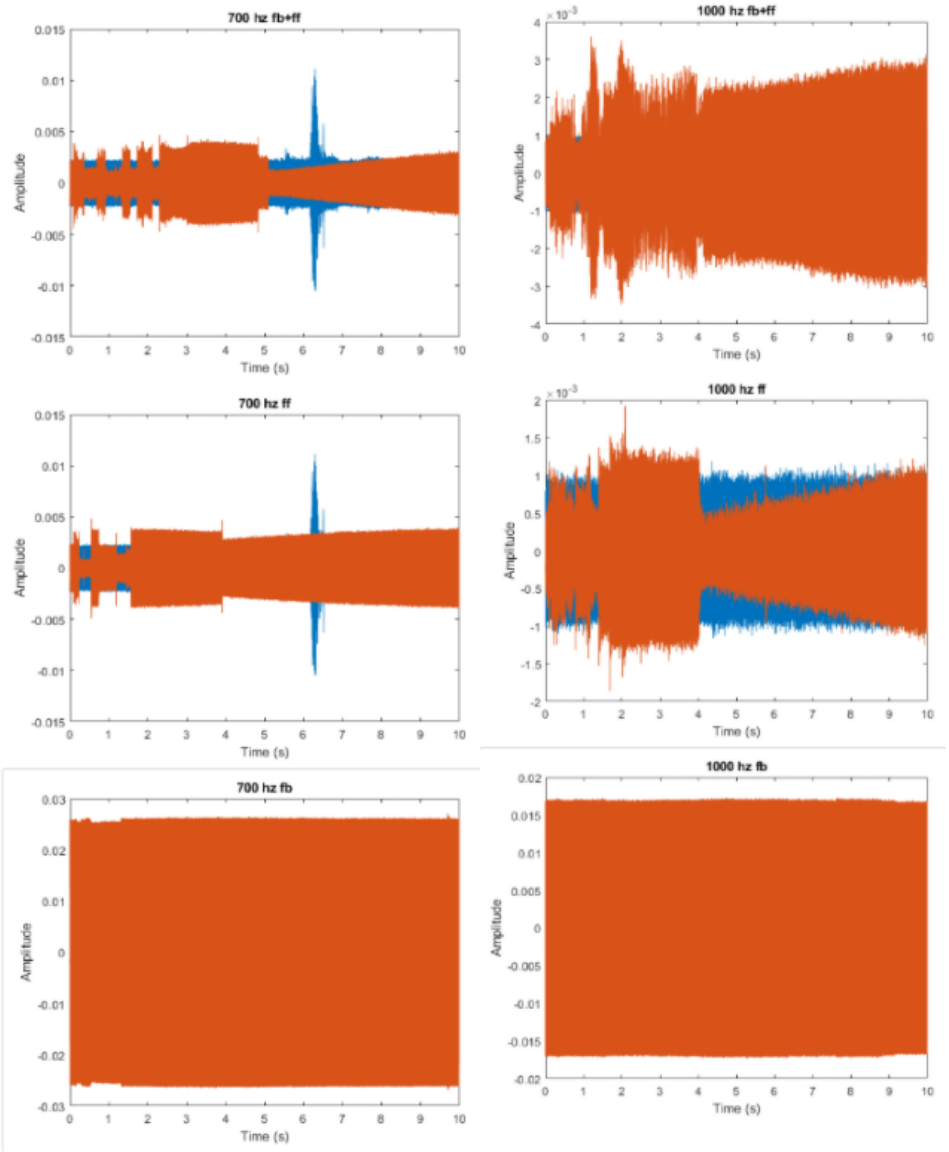

Figure 6.8: Signals 400 and 500 Hz

In order to test the system, the PI implementation with the best results ($KP = 0.4548 * 2, KI = 10s^{-1}$) is implemented in the same manner as the previous experimental procedure.

	400Hz		500Hz		700Hz		1000Hz	
Reduction	db	%	db	%	db	%	db	%
feedforward	-	-39.279	-	-2.55	-	-43.02	-	-8.7059
feedback	-	-28.4	-1.9675	36.430	-	-1061.4	-	-33.779
feedforward + feedback	-	-478.69	-	-356.68	-11.3	7.38	-	-1196

Figure 6.9: Table of results

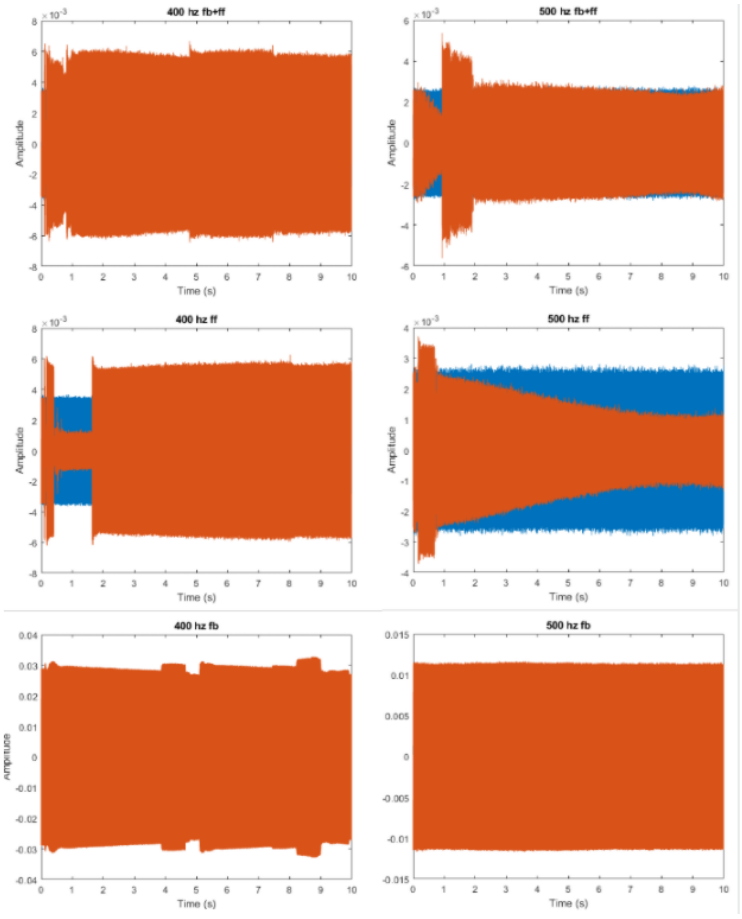

Figure 6.10: Signals 400 and 500 Hz

This procedure seems to create a positive feedback loop more often, making it worse than the original one.

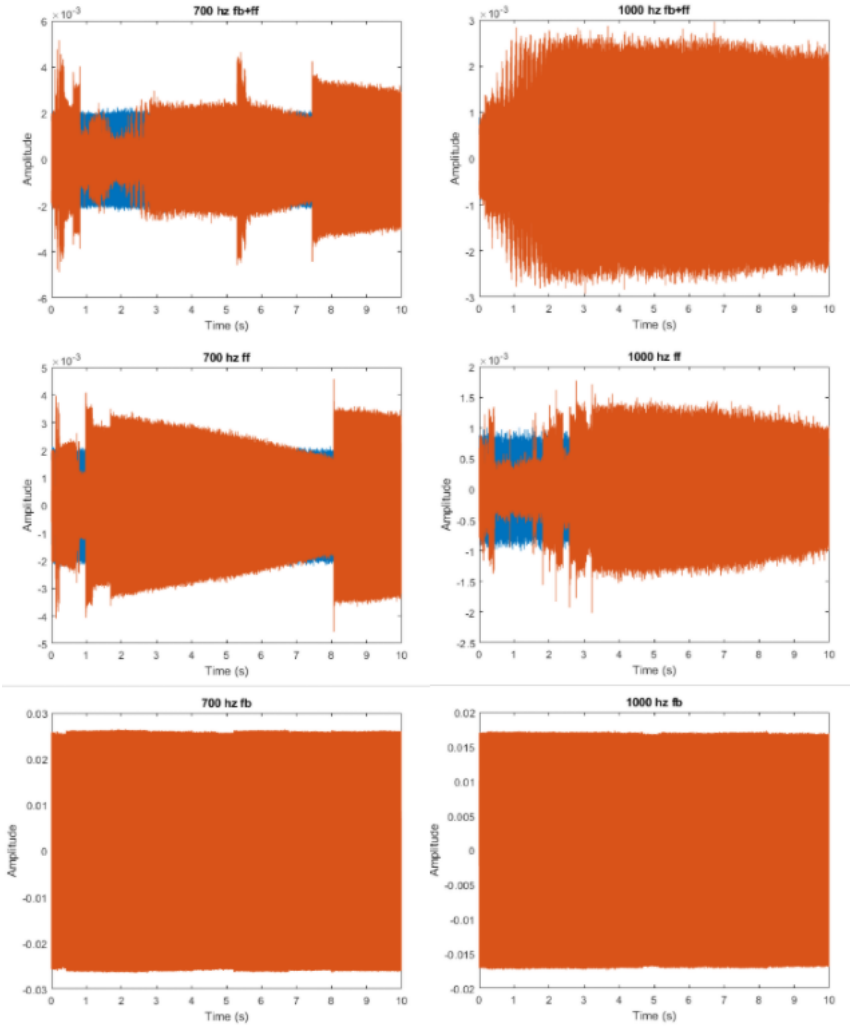

Figure 6.11: Signals 700 and 1000 Hz

As seen in the previous experiments, a reduction in the PID parameters is in order. The proposed parameters for the next iteration consists of simply halving the original parameters.

	400Hz		500Hz		700Hz		1000Hz	
Reduction	db	%	db	%	db	%	db	%
feedforward	-	-6.4	-	-4.2	-	-12	-	-10.18
feedback	-	-28.188	-1.9675	7.1	-	-24	-	9.64
feedforward + feedback	-	-522	-	-244	-	-724	-	-403.2

Figure 6.12: Table of results

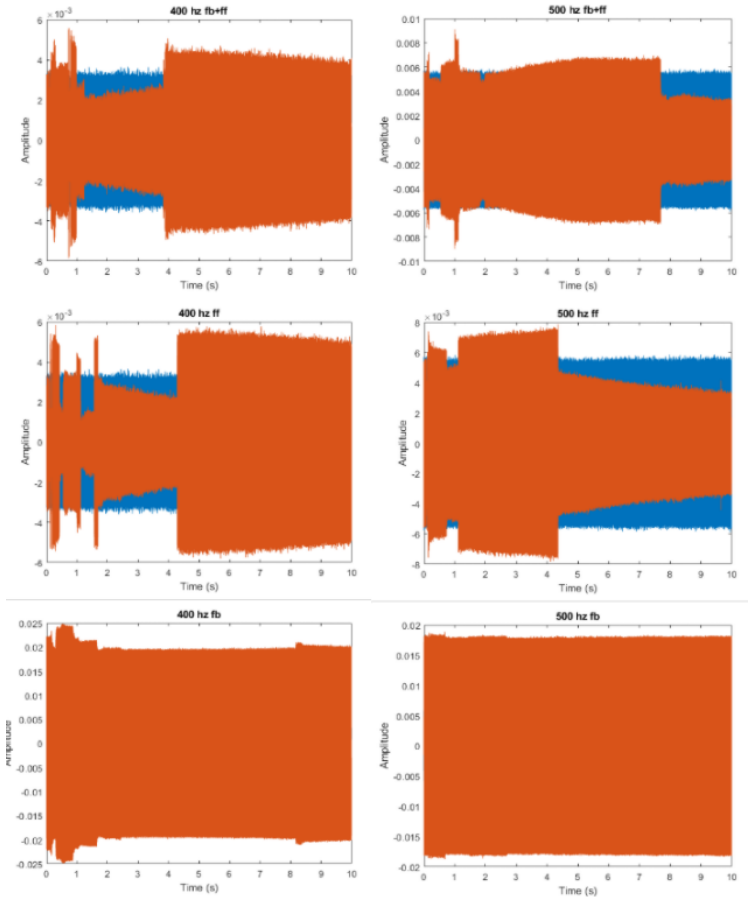

Figure 6.13: Signals 400 and 500 Hz

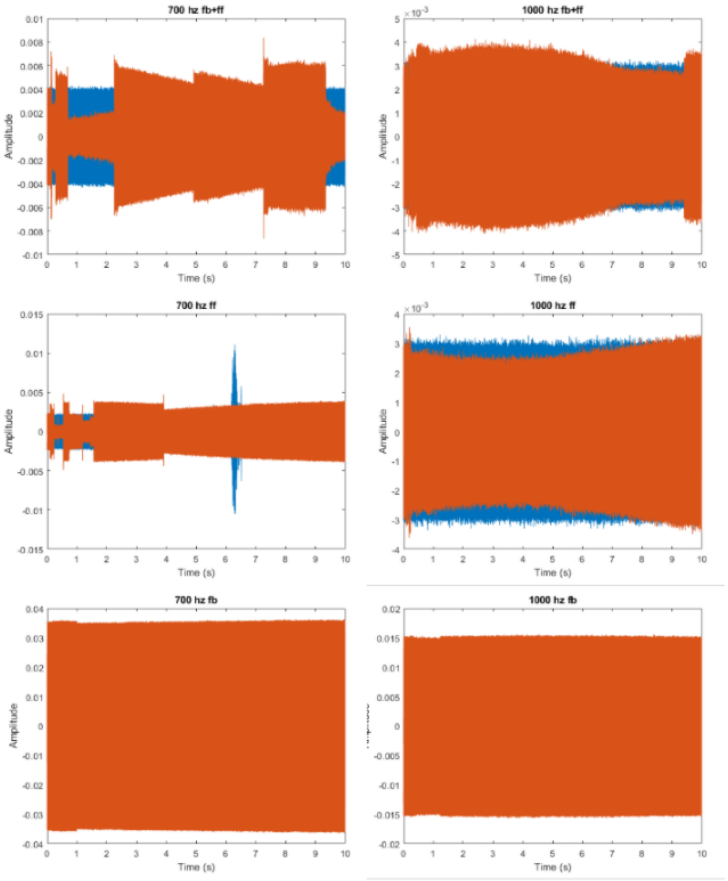

Figure 6.14: Signals 700 and 1000 Hz

The best results come from the initial parameters with both feed forward and feedback, that gave positive results in all frequencies except 1000Hz, and the feed forward, that gave positive results except for the 700 Hz frequency signal,though there is still a lot of space for experimentation.

Even in the better cases, the reduction does not reach the level of the throretical program. That is to be expected, as the theoretical program ran with a better implementation of the feed forward controller, with no overlap and no background noise.

In conclusion, much experimentation can still be made and the results in certain frequencies look very promising.

7 CONCLUSIONS

During the length of this project, the architecture for the active noise cancelling system from a different room has been put into practice. That required the identification of the system through the primary and the secondary path, the identification of the transfer functions of those paths, and the tuning of the controllers by the use of those transfer functions.

The limits of the Raspberry Pi 1 as far as Active Noise Reduction goes have been tested and reached, making it a great opportunity for learning new ways to speed up the process that have been implemented in the definitive program. The application of the controllers has given very promising results in the simulations, though in practice the system has given much more mixed results, with very respectable noise reductions in certain frequency ranges and being completely counterproductive in others.

These results open up a lot of work to improve the reduction. A lot of improvement could come too from toying around with the parameters and the results of the final system, and a new identification of the system with the faster board could bring substantial improvements. The aim of this thesis has been reached, with the system being implemented and giving (be it variably) positive results with the proposed relatively low-cost equipment.

8 BIBLIOGRAPHY

REFERENCES

[1] Eduard-Mihai Oprea, (2020). Final degree report Bachelor's Degree in Industrial Electronics and Engineering Sound monitoring and active noise cancelling at the campus library Masip-alvarez Tutor. June.

[2] ACTIVE NOISE CANCELLATION A Degree Thesis Submitted to the Faculty of the Escola Tècnica d ' Enginyeria de Telecomunicació de Barcelona Universitat Politècnica de Catalunya by Guillem de la Torre Rovira In partial fulfilment of the requirements for the de. (2017). June.

[3] ACTIVE NOISE CANCELLATION OVER RASPBERRY PI Submitted to the faculty of Escola Tècnica d ' Enginyeria de Telecomunicació de Barcelona Universitat Politècnica de Catalunya by Jordi Agudo. (2014). Director: Francesc Vallverdú Bayés

[4] A weighing in detail. (n.d.). Retrieved December 8, 2020

[5] Harris, F. J. (1978). On the Use of Windows with the Discrete Fourier Transform. Proceedings of the IEEE, 66(1), 51–83.

[6] E. C. Levy, "Complex-curve fitting," in IRE Transactions on Automatic Control, vol. AC-4, no. 1, pp. 37-43, May 1959, doi: 10.1109/TAC.1959.6429401.

[7] Krishnamurthy, V., Seshadri, V. (1978). Model Reduction Using the Routh Stability Criterion. IEEE Transactions on Automatic Control, 23(4), 729–731.

[8] EFFICIENT RASPBERRY PI IMPLEMENTATION OF A SISO ACTIVE CONTROL SYSTEM Charlie House , Jordan Cheer Institute of Sound and Vibration Research , University of Southampton. (n.d.). 1–5.

[9] Curve fitting Process - YouTube. (n.d.).

[10] Microphones used- Amazon. (n.d.)

[11] Raspberry Pi 4 Model B specifications – Raspberry Pi. (n.d.)

[12] Hubert Pham. (2006). PyAudio Documentation — PyAudio 0.2.11 documentation.

[13] ANC program from GitHub

ANNEX VI – DECLARACIÓ D'HONOR

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

I declare that,

the work in this Master Thesis / Degree Thesis (*choose one*) is completely my own work,

no part of this Master Thesis / Degree Thesis (*choose one*) is taken from other people's work without giving them credit,

all references have been clearly cited,

I'm authorised to make use of the company's / research group (*choose one*) related information I'm providing in this document (*select when it applies*).

I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by *The Universitat Politècnica de Catalunya - BarcelonaTECH*.

Student Name: Oriol Morros

Date: 13/07/2021

Title of the Thesis : Study of a monitoring system of the noise level in the work rooms of the campus library in Terrassa