UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

DANILLO JOSÉ CEZAR RIBEIRO

SENSOR DE PRESENÇA BASEADO EM RESPOSTA ACÚSTICA DO AMBIENTE

JOÃO PESSOA
2019

DANILLO JOSÉ CEZAR RIBEIRO

SENSOR DE PRESENÇA BASEADO EM RESPOSTA ACÚSTICA DO AMBIENTE

Monografia apresentada ao curso Engenharia de Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em Engenharia de Computação.
Orientador: Hugo Leonardo Davi de Souza Cavalcante

JOÃO PESSOA
2019

Catalogação na publicação
Seção de Catalogação e Classificação

R484s Ribeiro, Danillo José Cezar.
SENSOR DE PRESENÇA BASEADO EM RESPOSTA ACÚSTICA DO AMBIENTE / Danillo José Cezar Ribeiro. - João Pessoa, 2019.
55 f. : il.

Orientação: Hugo Leonardo Davi de Souza Cavalcante.
Monografia (Graduação) - UFPB/CI.

1. Sensor de presença. 2. propagação de som. 3. geração e captação de som. 4. funções de resposta. 5. processamento digital de sinais. 6. acústica do ambiente. 7. som. 8. pulso. 9. hardware. 10. sinal. 11. software. 12. python. 13. sounddevice. I. Cavalcante, Hugo Leonardo Davi de Souza. II. Título.

UFPB/CI

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

ATA DE DEFESA PÚBLICA DO TRABALHO DE CONCLUSÃO DE CURSO

Aos **27** dias do mês de **Setembro** de **2019**, às **14 horas** e **00 minutos**, em sessão pública no **Auditório do Centro de Informática** da Universidade Federal da Paraíba, na presença da banca examinadora presidida pelo professor orientador **Dr. Hugo Leonardo Davi de Souza Cavalcante** e pelos professores **Dr. Carlos Eduardo Coelho Freire Batista** e **Dr. Guido Lemos de Souza Filho**, o aluno **Danillo José Cezar Ribeiro** apresentou o trabalho de conclusão de curso intitulado **Sensor de Presença Baseado em Resposta Acústica do Ambiente** como requisito curricular indispensável para a integralização do **Curso de Bacharelado em Engenharia da Computação**.

Após a exposição oral, o candidato foi arguido pelos componentes da banca que reuniram-se reservadamente e decidiram aprovar a monografia, com nota 10,0. Divulgando o resultado formalmente ao aluno e demais presentes, eu, na condição de Presidente da Banca, lavrei a presente ata que será assinada por mim, pelos demais examinadores e pelo aluno.

Hugo L. D. S. Cavalcante
Prof. Dr. Hugo Leonardo Davi de Souza Cavalcante

Carlos Eduardo Coelho Freire Batista
Dr. Carlos Eduardo Coelho Freire Batista

Dr. Guido Lemos de Souza Filho

Danillo José Cezar Ribeiro
Danillo José Cezar Ribeiro

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

Primeiramente a Deus pelo Dom da vida, por ser consolo e alegria.

Aos meus pais por estarem ao meu lado em todos os momentos, pela educação e motivação em ser alguém melhor e por toda paciência.

À minha companheira por ser berço de conforto e aos meus amigos por serem motivos de alegria.

RESUMO

A propagação de ondas, que depende necessariamente da geometria do ambiente, é um problema relativamente bem definido. Contudo aplicações que dependam dessa característica são difíceis de implementar e suscetíveis a muitos tipos de interferência. Essa propagação em um ambiente espalhador geralmente produz sinais de resposta extremamente complexos que são característicos de cada geometria, das suas propriedades físicas e sensíveis a perturbações que possam ocorrer. Propomos usar as características desse ambiente espalhador para construir um sistema sensor de presença que emita um conjunto de sinais sonoros capazes de identificar alterações causadas, por exemplo, por uma pessoa não autorizada em um ambiente privado ou por acidentes que modifiquem essa geometria. A partir de etapas de calibração, controle e teste, este sistema é capaz de detectar e possivelmente alertar sistemas secundários de presenças indevidas. Este trabalho aplica técnicas matemáticas relativamente simples de análise de sinais sonoros para essa finalidade podendo ser implementado em hardware já existente, como computadores e telefones celulares.

Palavras-chave: Sensor de presença, propagação de som, geração e captação de som, funções de resposta, processamento digital de sinais, acústica do ambiente, som, pulso, hardware, sinal, software, python, sounddevice.

ABSTRACT

Wave propagation that necessarily depends on the geometry of the environment is a more or less well-defined problem. However applications that depend on this feature are difficult to implement and susceptible to many types of interference. This propagation in a spreader environment often produces extremely complex response signals that are characteristic of each geometry, its physical properties and sensitive to disturbances that may occur. We propose to use the characteristics of this spreader environment to construct a presence sensing system that emits a set of beeps capable of identifying changes caused, for example, by an unauthorized person in a private environment or by accidents that modify this geometry. From calibration, control and testing steps, this system is capable of detecting and possibly alerting secondary systems of improper presence. This work applies relatively simple mathematical sound analysis techniques for this purpose and can be implemented on existing hardware such as computers and mobile phones.

Key-words: presence sensor, sound propagation, sound generation and capture, response functions, digital signal processing, ambient acoustics, sound, pulse, hardware, signal, software, python, sounddevice.

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Sensores sonoros são amplamente utilizados em aplicações de eletrônica e na robótica, como no auxílio da localização de robôs e também no uso doméstico a exemplo de sensores de presença em sistemas de alarme. Esta última aplicação, entretanto, pode apresentar dificuldades em ser aplicada devido a sua complexidade em adaptar-se a múltiplos ambientes. O que pode relacionar-se com esse problema é o fato das ondas refletirem com mais facilidade e com uma grande periodicidade no meio espalhador como paredes, objetos ou até mesmo o ar. Essa propagação dificulta o uso de ondas sonoras em grandes volumes ou em situações que não possam ser controladas detalhadamente, transformando-se em um problema complexo de ser modelado.

Os sistemas que utilizam essa grandeza estão sendo cada vez mais utilizados em aplicações que não necessariamente envolvem a reprodução de um som audível. A faixa de frequência do espectro sonoro audível por um ser humano está entre 20 (frequência mais baixa) e 20000 (frequência mais alta) ciclos por segundo (Hz), mas não necessariamente frequências fora dessa faixa estão inutilizadas. Equipamentos de radar marítimo, como o CHIRP, utilizam dessas frequências como sua função principal, onde as mais baixas ajudam no reconhecimento de objetos a uma longa distância com baixa resolução, e com as mais altas pode-se identificar objetos com uma alta resolução porém a distâncias menores [1].

1.1 Tema

Apesar dessa grandeza fornecer uma resposta por vezes incerta e desfavorável, utilizar as suas características para ter respostas à favor de uma aplicação pode ser vantajoso pelo seu baixo gasto de energia.

O que será abordado neste trabalho trata-se de técnicas utilizadas de forma que essas medições sejam úteis na tomada de decisão ou na ativação de sistemas de detecção de intrusos e vigilância em ambientes, por exemplo.

1.2 Problema

A propagação de ondas no meio depende da geometria 3D de todo o ambiente e pode exibir soluções complicadas. As transformações sofridas por um pulso curto de onda, aqui chamado *sona*, enquanto este se propaga em um ambiente espalhador, sujeito a múltiplas reflexões e reverberações, produzem um sinal complexo, conhecido como *coda*. A forma específica da coda para uma dada sona depende dos detalhes da geometria do ambiente e das propriedades dos materiais presentes, os quais afetam a velocidade de propagação da onda. No entanto, estas transformações são determinísticas, e portanto, a replicação de uma mesma sona, no mesmo ambiente, necessariamente deve produzir a mesma coda.

1.2.1 Objetivo geral

Propomos usar a resposta do espalhamento da onda através do ambiente para desenvolver um sensor capaz de detectar a ocorrência de pequenas perturbações no meio, usando sonas de ondas acústicas produzidas por um alto-falante de computador ou de celular. Em vez de detectar diretamente as alterações da coda, que é uma função complicada e sensível, usaremos uma técnica de análise da resposta acústica para que o ambiente, através de sua ação normal, produza uma sona cuja fidelidade é afetada pela perturbação (TADDESE, 2010).

1.2.2 Objetivos específicos

A proposta deste trabalho é estudar a viabilidade e desenvolvimento deste sensor de presença/violação de perímetro, que pode indicar alterações inesperadas em ambientes vazios. Baseado na reprodução do experimento realizado por TADDESE, 2010, onde seu principal objetivo era a detecção de uma perturbação a partir da emissão de um sinal sonoro, o sinal emitido é “moldado” de acordo com as

características do ambiente, ouvido por um receptor e através de técnicas sensitivas e o cálculo da figura de mérito deste sinal a perturbação é detectada. Dessa forma, visamos a prototipagem e implementação em Python de um programa de computador que emite, recebe e analisa pulsos de ondas sonoras, determinando se houve alguma modificação no ambiente. O objetivo é utilizar este programa para ativar alarmes ou outras medidas de segurança, como o registro de imagens por câmeras, por exemplo. De acordo com a teoria explicada no artigo de referência, o eco produzido por este pulso é sensível à perturbações na sala, como por exemplo à presença de um intruso, mesmo se este estiver fora do campo de visão das câmeras de segurança. Com a aplicação de várias instâncias do programa, executadas em diferentes máquinas e, realizando uma vigilância integrada em vários ambientes de um prédio, temos uma forma de acompanhar o movimento de um dado indivíduo por corredores e salas, por exemplo.

2 CONCEITOS GERAIS E REVISÃO DA LITERATURA

A linguagem de programação Python 2.7 [2] foi utilizada na implementação do software e se destaca no uso para modelagens 3D, edição de imagens, segurança da informação, mas também amplamente aplicada em projetos com propósitos numéricos. O Python se destaca no quesito desempenho por possuir uma grande variedade de bibliotecas para esse fim, a exemplo da *matplotlib*, aqui utilizada para manipulação de gráficos 2D.

Além disso, para este trabalho utilizaremos o módulo *sounddevice* que, a partir do hardware presente no dispositivo, executa rotinas de reprodução e gravação de arrays de tipo *numpy* [3] contendo sinais de áudio. Feita a instalação, essas funções podem ser facilmente aplicadas, de maneira sequencial ou paralela, desde que se obedeçam os requisitos paramentais [4].

A IDE [5-6] *Spyder* também foi selecionada como ambiente de desenvolvimento. O nome vem de Scientific Python Development Environment [7] (Ambiente de Desenvolvimento Científico Python) e compreende-se como uma plataforma que é abundantemente empregada no uso de edição avançada, análise, debug e inspeção de dados, desenvolvido para cientistas, engenheiros e analistas.

Algumas outras definições devem ser levadas em consideração no decorrer deste trabalho. Para que os experimentos sejam realizados de forma adequada, estabelecemos a priori um ambiente delimitado e um pulso de onda sonora a ser emitido.

Na análise das respostas devemos levar em conta as definições a seguir:

- Ruído: Antes do seu falecimento, John Cage [8-9] fala a seguinte frase em uma entrevista: “*There is no noise, only sound*” [10] (“Não existe barulho, apenas som.”) que define bem o conceito relativo de ruído. Neste trabalho definimos ruído não apenas como um barulho, visto que não há um barulho que não seja som, mas como todo e qualquer som indesejado produzido.
- Fenômenos associados a ondas sonoras: A análise de uma dada onda sonora deve ser feita levando em conta múltiplas variáveis do ambiente a ser

propagado e dos dispositivos emissores e receptores envolvidos. Essas variáveis alteram diretamente as propriedades e os efeitos físicos dessas ondas, que são:

- ↳ Emissão: relaciona-se com a fonte do sinal, o alto-falante. A emissão do sinal sonoro vai desde a geração da corrente pelo circuito, sua transmissão através de um meio condutor até a vibração da membrana na sua reprodução;
- ↳ Propagação: diz respeito a forma com que o sinal se difunde no meio físico. Alguns experimentos assemelham esse comportamento como o de uma gota caindo na superfície da água, sua propagação é circular;
- ↳ Reflexão: se esse meio possui obstáculos, por menores que sejam - até mesmo o ar pode ser considerado um obstáculo - esses, irão interferir de formas diferentes no espalhamento da onda. O sinal sonoro incide sobre o material e muda sua direção;
- ↳ Espalhamento: remete-se aos múltiplos desvios que o sinal pode assumir: o resultado de várias reflexões;
- ↳ Detecção: corresponde desde o momento em que a variação de pressão do ar é detectada pelo aparelho eletrônico, o microfone, até a amplificação desse sinal para o computador.

• Fenômeno da interferência de ondas: Há casos específicos onde duas ou mais ondas, ao se propagar no meio físico, podem acabar se interferindo. Por exemplo: dadas duas fontes, S1 e S2, produzindo ondas de mesma amplitude A, com a mesma frequência f e comprimento de onda λ em um meio físico, o fenômeno da interferência de ondas pode ocorrer de forma:

- ↳ Construtiva: onde as fases das ondas, no pico ou no vale de ambas, se cruzam e se somam resultando em uma onda de amplitude $2A$;
- ↳ Destrutiva: onde o pico de uma onda interfere no vale da outra, ou vice-versa, resultando em uma amplitude nula.

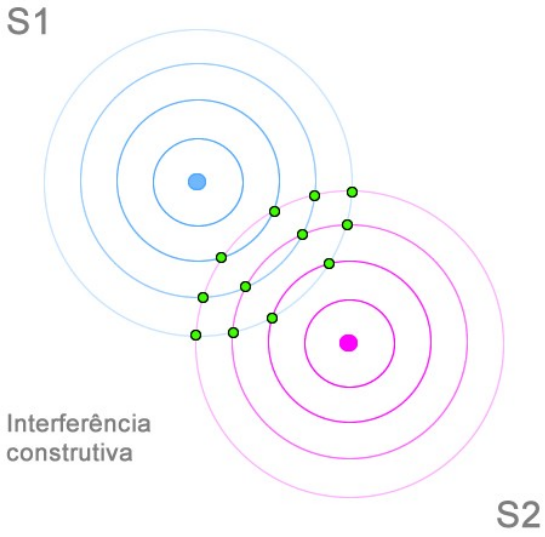

Figura 1: Ondas sonoras em interferência construtiva.

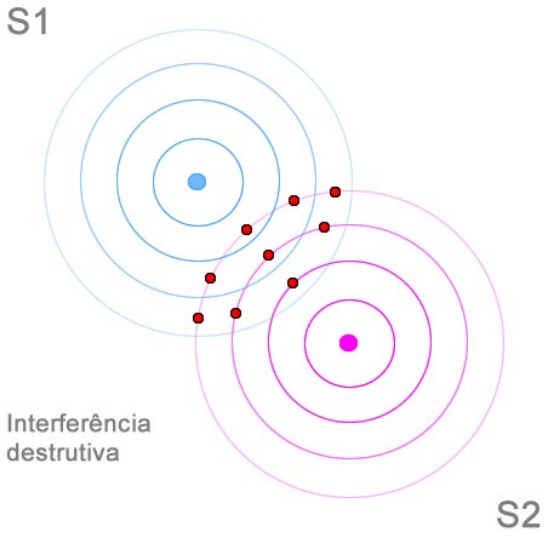

Figura 2: Ondas sonoras em interferência destrutiva.

- Pulso: neste trabalho quando mencionamos a palavra "pulso" nos referimos a emissão de um sinal de frequência, amplitude e período definidos, e não a um pulso delta. O pulso delta, em sinais e sistemas dinâmicos, se caracteriza por sua largura unitária e amplitude infinita.

2.1 Experimentos prévios

Para o experimento executado por Taddese, o objeto perturbador possuía dimensões de 60cm de altura x 35cm de largura enquanto o ambiente de testes possuía dimensões de 6m de profundidade x 2.5m de largura x 6.5m de altura.

	Volume
Ambiente de testes	97.5 m³
Objeto perturbador	0.06 m³
Razão de proporção	1:1625

Tabela 2.1: Razão de proporção no experimento prévio.

O autor usou um microfone condensador como receptor e um alto-falante como emissor, ambos localizados dentro do ambiente, porém conectados a um computador que se encontrava no exterior. Para os experimentos, Taddese posicionou o objeto em lugares distintos de forma a distanciá-lo gradativamente do emissor e nomeou esses locais de A a F.

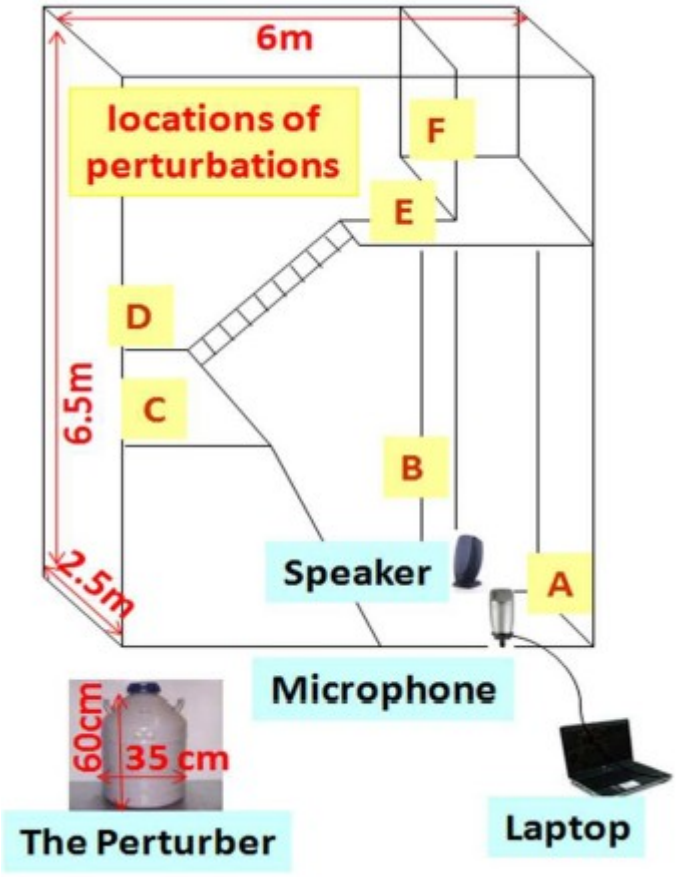

Figura 3: Estrutura do experimento feito por Taddese e locais do objeto perturbador.

Para cada localização ele empregou e comparou quatro técnicas sensitivas de análise dos sinais obtidos:

- CTRS1 (Chaotical Time Reversal Sensor 1): Com a reconstrução pela inversão temporal do pulso obtido no ambiente sem perturbação (BRP) e com perturbação (PRP), pode ser aplicada a técnica que mensura a diferença pico a pico das amplitudes desses sinais.

  $$I_{CTRS1} = \frac{Pk - Pk\ Amplitude_{PRP}}{Pk - Pk\ Amplitude_{BRP}}$$

  A técnica é mais eficiente computacionalmente e deve ter seu valor próximo a 1 caso as condições de perturbação do ambiente sejam iguais antes de depois da perturbação, ou seja, se PRP for igual a BRP. O índice I_{CTRS1} será menor que 1, caso contrário.

- CTRS2 (Chaotical Time Reversal Sensor 2): Já nesta técnica, o autor emprega a divisão das correlações entre a reconstrução da resposta do ambiente (com ou sem perturbação - PRP ou BRP) e o pulso original reconstruído (ROP), sobre a normalização desses sinais. O indicador desta técnica é:

  $$I_{CTRS2} = \frac{\frac{\langle PRP, ROP \rangle}{\|PRP\| \|ROP\|}}{\frac{\langle BRP, ROP \rangle}{\|BRP\| \|ROP\|}}$$, onde PRP , BRP e ROP são vetores no tempo. Quando sem perturbação, o índice I_{CTRS2} pode estar próximo, mas nunca igual a 1 (um), por fatores como: a não perfeição na reversão e emissão do pulso através da execução do alto-falante e a posterior gravação pelo microfone. Essa característica pode estar diretamente ligada a utilização de um ambiente não ideal, com a presença de imperfeições e ruídos indesejados.

- SCC (Sensing by Cross Correlation): Na detecção por correlação cruzada, a medida de similaridade entre dois sinais é calculada a partir do atraso aplicado a um deles [11-13]. Essa técnica é amplamente utilizada para encontrar um sinal padrão de curta duração em um de longa e é calculada a partir de:

  $$(X * Y)(t) = \int_{m=0}^{m=l} X(\tau) Y(t+\tau)\, d\tau$$, onde X e Y são os sinais

comparados, t é o valor do atraso aplicado com relação ao sinal Y e l é o máximo valor em que a função $X(\tau)$ e $Y(t+\tau)$ é bem definida.

- SMI (Sensing by Mutual Information): Esse método tem como base a detecção por informação mútua [14-15], seu conceito está diretamente ligado ao da entropia de variáveis aleatórias e é definida por:

  $I_{SMI}=\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2\left(\frac{p(x,y)}{p(x)p(y)}\right)$, onde $p(x)$ e $p(y)$ são as funções de massa de probabilidade marginal, $p(x,y)$ é a função de massa de probabilidade conjunta que diz respeito à probabilidade que o sinal X e Y assuma, respectivamente, os valores x e y .
  A informação mútua é 0 (zero) se os sinais comparados forem independentes, e acima de 0 (zero) caso sejam idênticos.

Dentre as técnicas aplicadas, Taddese obteve os seguintes resultados:

Técnica sensitiva	**A**	**B**	**C**	**D**	**E**	**F**
CTRS1	97%	81%	73%	71%	67%	46%
CTRS2	90%	63%	67%	32%	48%	13%
SCC	100%	95%	87%	78%	87%	24%
SMI	76%	71%	89%	54%	76%	6%

Tabela 2.2: Máxima porcentagem de detecção para cada uma das quatro técnicas.

3 METODOLOGIA

Utilizamos como base o experimento feito por Taddese em 2010 para desenvolver este trabalho. Após analisar as técnicas e os resultados obtidos por ele, podemos então aplicá-los em ambientes de teste e objetos perturbadores distintos, por exemplo. As seções seguintes definem a metodologia aplicada na geração do pulso, nas simulações em Python e nas análises das respostas obtidas, assim como as características do experimento realizado como: o hardware utilizado, as dimensões do ambiente de testes e do objeto perturbador.

3.1 Especificações da máquina e dos equipamentos

Os experimentos foram realizados em um notebook com processador Intel® Core™ i5-3210M CPU @ 2.50GHz, memória RAM de 6,00 GB, com 1GB de memória dedicada a vídeo e sistema operacional Windows 8.1 de 64 bits com o auxílio de 2 (duas) e 6 (seis) caixas de som estéreo externas a depender do experimento, conectadas via USB e cabo P2.

3.2 Ambientes de teste

Os ambientes de teste escolhidos compreendem-se como 3 (três) locais passíveis de pouco ruído: uma sala de professor com dimensão aproximada de 366cm x 808cm, um quarto com 257cm x 280cm e uma sala do Centro de Informática - UFPB em dias sem aulas ou atividades próximas, com 366cm x 808cm, todos delimitados por 4 (quatro) paredes de concreto com espessura aproximada de 10 cm. Considerou-se também o repouso absoluto de todos os objetos contidos nesses ambientes para favorecer a precisão da execução e dos resultados.

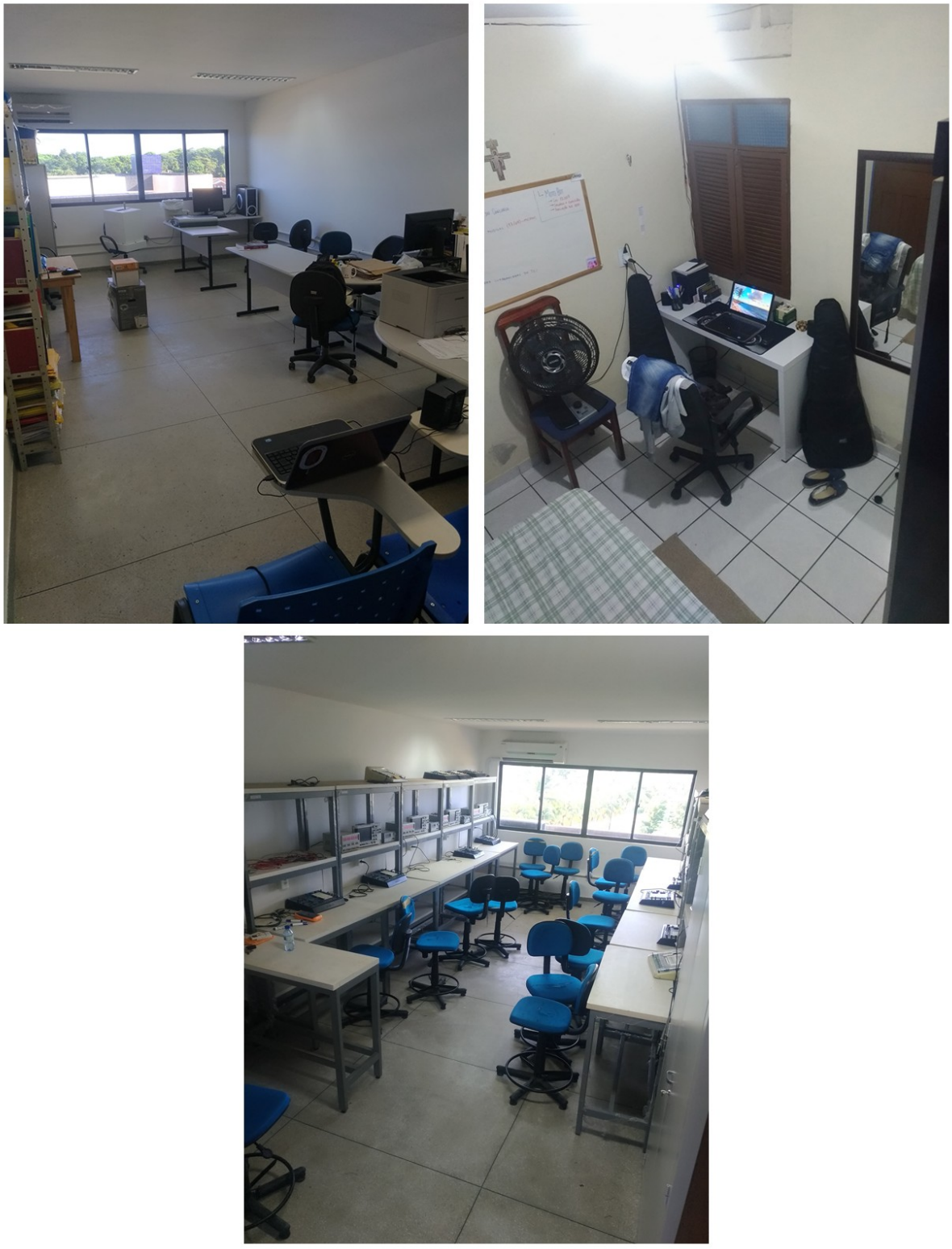

Figura 4: Ambientes de teste do dia 1 (superior esquerda), dia 2 (superior direita) e dia 3 (inferior).

3.3 Criação do pulso

Para que o experimento fosse realizado, precisaríamos de um pulso curto gerado a partir dos alto-falantes do dispositivo executado. A largura desse pulso foi definida de forma que sua duração fosse suficiente e não interferisse no processo de captura da resposta do ambiente. O pulso foi gerado a partir da multiplicação de um envelope de um super gaussiana $e^{-x^{10}}$ por uma portadora cossenoidal. A super gaussiana se difere da gaussiana simples e^{-x^2} apenas pelo seu expoente de decaimento 5 vezes maior, o que faz com que o pulso tenha uma "cauda" menor.

A portadora vai fornecer a frequência do pulso, já o envelope fornece a duração e o formato.

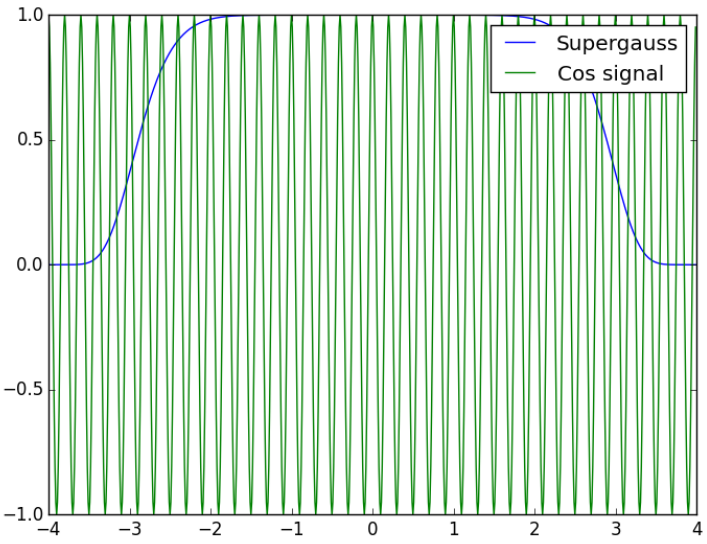

Figura 5: O sinal cossenoidal (em verde) e a supergaussiana (em azul).

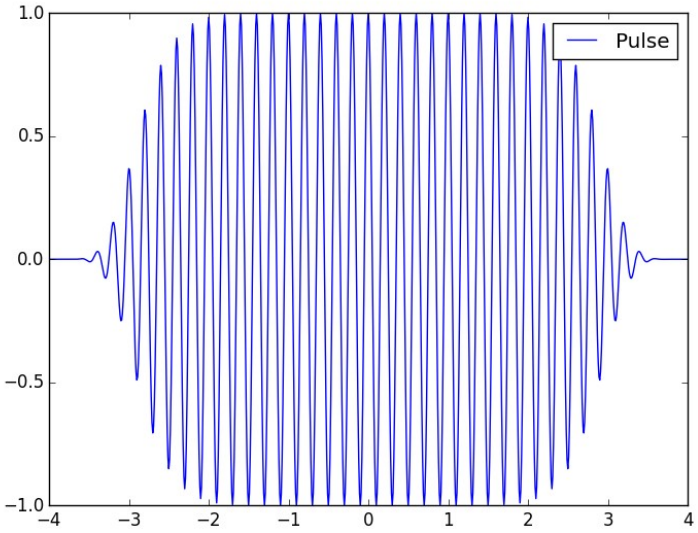

Figura 6: Resultado da multiplicação da portadora cossenoidal com o envelope supergaussiano.

A função em python feita para a geração desse pulso foi a *pulse_gen()*, com o envelope e portadora descritos abaixo:

- $s(t)=\cos(\omega t)$, onde $\omega=2\pi f$, e
- $e(t)=\mathrm{e}^{-(t/L)^{10}}$, onde L é a metade da duração do pulso.

Algumas características do pulso como sua duração, frequência e amplitude, foram implementadas como parâmetros de entrada dessa função e estão descritas abaixo:

- Frequência `f = 2200` Hz;
- Duração total da onda `waveform_duration = 800` ms;
- Duração do pulso `pulse_duration = 700` ms;
- Tempo inicial do pulso `t0 = 120` ms;

A seção 3.5.3 e o código-fonte *pulse_creation.py* (disponível no Apêndice A - Códigos-fonte, “Criação do pulso”) descrevem como esse processo foi feito.

3.4 Técnica de análise de ondas

Baseado no quadro comparativo (tabela 2.2), a técnica SCC (Sensing by Cross Correlation) apresentou maior porcentagem de acerto - em média 78% - dadas as condições do experimento feito. Seguindo essa direção, foi selecionado a detecção por correlação cruzada.

Como vimos anteriormente, esta técnica é utilizada na comparação de sinais para identificar semelhanças e é definida por:

$(X*Y)(t)=\int_{m=0}^{m=l} X(\tau)Y(t+\tau)\,d\tau$, todavia para que esse cálculo seja executado no computador é necessário a discretização dos valores. Desse modo, a integral torna-se um somatório de valores discretos:

$(X*Y)[n]=\sum_{m=0}^{m=l} X[m]Y[m+n]$.

Em acréscimo, para a normalização dos valores obtidos, um conceito importante e bastante útil é a aplicação da correlação de Pearson onde o acréscimo da normalização faz com que os valores obtidos na equação $(X*Y)(t)$ variem apenas entre -1 e +1. A representação discreta a seguir é bastante semelhante à da equação anterior $(X*Y)[n]$:

$(X*Y)[n]=\frac{\sum_{m=0}^{m=l} X[m]Y[m+n]}{\|X\|\|Y\|}$. A magnitude dos sinais de comparação X e Y é calculada no denominador, com a utilização da norma. A semelhança é dada a partir do máximo valor encontrado na correlação. Em um cenário com nenhuma perturbação esse valor deve estar próximo a 1 (um).

3.5 Simulação em ambiente ideal

Antes de executar testes práticos houve a necessidade da simulação dos conceitos e técnicas estudadas em um ambiente ideal para testar sua funcionalidade. Este ambiente deveria ser livre de qualquer perturbação exterior mas

poderia simular e se aproximar ao máximo de um ambiente real com atenuação do sinal de acordo com o tempo, a distância emitida e também suas respectivas reflexões. As simulações foram desenvolvidas em software onde o objetivo final era obter um histograma das figuras de mérito dos grupos de controle e de teste. Se os grupos se sobrepunham, então o conceito se tornaria inválido, não havendo a necessidade da aplicação em um ambiente real.

3.5.1 Breves conceitos

Antes de definir cada função contida no código, é importante que alguns conceitos sejam lembrados. São eles: a média, o desvio padrão e a figura de mérito.

1. Média (μ): A média de n pulsos representados por um array no Python, nada mais é que a soma dos respectivos índices dividido pelo número de pulsos.

   $$\mu = \frac{1}{n}\sum_{i=1}^{n} pulse_i$$

   Em python, a função numpy *mean()* traz esse resultado de forma automática.

2. Desvio padrão (σ): A partir da média e da variância, podemos calcular a medida de dispersão de n pulsos utilizamos o desvio padrão.

   $$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (pulse_i - \mu)^2}$$

   Essa medida também pode ser calculada em python através da função numpy *std()*, que usa a raiz quadrada da média dos desvios quadrados médios.

3. Figura de mérito (FOM): A figura de mérito (Figure of Merit) é uma medida que compara os méritos relativos das técnicas empregadas no trabalho. Essa medida ajuda na escolha do conjunto ótimo de parâmetros que irão detectar uma perturbação e é calculada a partir da razão da subtração da média (μ

) com o valor da medida de correlação (I) pelo desvio padrão (σ): $FOM = \frac{\mu - I}{\sigma}$.

3.5.2 Grupos de sinais

Por questões de organização, foram definidos três grupos de 25 pulsos cada nas fases de simulação em ambiente ideal e aplicação em ambiente real. 25 pulsos foi uma quantidade suficiente para que o resultado não fosse influenciado caso alguma perturbação não desejada acontecesse. A resposta do ambiente para cada pulso de cada grupo foi armazenada em disco. O software fará o cálculo das FOMs de cada grupo, e a partir disso, realizará a devida análise dos resultados. Os grupos são:

1. **Grupo de calibração:** os sinais contidos neste grupo pertencem a fase de calibração do ambiente de teste. Caso haja alguma perturbação indesejada, de curta duração, o cálculo da média e do desvio padrão dos 25 sinais emitidos auxiliarão na tomada de decisão. Foi definido em software com a variável X.
2. **Grupo de controle:** aqui os sinais servirão de controle do ambiente que ainda assim não poderá ter uma perturbação. Tendo isso em vista, esse grupo deverá ter seus resultados semelhantes ao grupo anterior. Sua função principal é testar a funcionalidade do programa e detectar falsos-positivos. Foi definido no software com a variável Y.
3. **Grupo de teste:** esse grupo é responsável pelo registro do movimento ou da perturbação causada no ambiente. Se o ambiente permanece inalterado, esse grupo se mantém semelhante aos anteriores. Caso não, a perturbação será detectada nas etapas posteriores do programa. Foi definido no software com a variável Z.

3.5.3 Análise do código

Agora podemos partir para a definição das funções. Nessa etapa de simulação o código utilizado é o *ping_test.py* (disponível no Apêndice A - Códigos-fonte, “Simulação em ambiente ideal”).

De início, temos a função que é primordial para o programa:

```
def pulse_gen(t, A):
    return A*np.cos(2*np.pi*f*t)*np.exp(-((t-t0)/pulse_duration)**10)
```

A função *pulse_gen()* é responsável por gerar os pulsos do tipo super gaussiana com decaimento exponencial que serão emitidos, refletidos no ambiente e analisados como resposta, como explicitado na seções anteriores. Essa mesma função vai ser aplicada nos grupos de calibração, controle e teste - aqui chamados de X, Y e Z, respectivamente. Para simular um ambiente real, com pequenas perturbações, cada série do grupo adicionará um “ruído artificial” aos seus valores a partir de uma distribuição normal. Este ruído está diretamente relacionado com o valor das variáveis *noise_amp* e *noise_amp_dist*. A variável noise_amp é atribuída às séries de calibração e controle e a variável noise_amp_dist ao grupo de testes, como mostrado abaixo.

```
for i in range(N_samples):
  X[:,i] = pulse_gen(t, A) + normal(0, noise_amp*A, N)
  Y[:,i] = pulse_gen(t, A) + normal(0, noise_amp*A, N)
  Z[:,i] = pulse_gen(t, A) + normal(0, noise_amp_dist*A, N)
```

Após a criação dos grupos, é feita a correlação de cada n sinal do grupo de controle (Y[n]) e teste (Z[n]) com um de referência do grupo de calibração (X[0], à escolha). No trabalho, a correlação cruzada foi feita a partir da implementação da função *xcorr()*, que calcula a média local e o desvio padrão de cada uma das séries passadas como parâmetro.

```
def xcorr(a1, a2):
  Num_points = np.amin([len(a1), len(a2)])
  mu_a1 = np.mean(a1)
  mu_a2 = np.mean(a2)
```

```
  a1_local = a1-mu_a1
  a2_local = a2-mu_a2
  A1 = np.std(a1_local)
  A2 = np.std(a2_local)
  X = np.correlate(a1_local, a2_local, mode =
'same')/(Num_points*A1*A2)
  return X
```

São usadas a função *amin()* que calcula o menor tamanho entre os arrays a1 e a2 passados como parâmetro e a *correlate()*, pertencente ao módulo numpy, no modo 'same' que calcula a correlação cruzada entre duas séries de tamanhos iguais.

Com o array de correlações pronto e a média e o desvio padrão calculadas anteriormente, podemos enfim obter o valor das figuras de mérito de cada série dos grupos Y e Z:

```
### TEST STEP
for i in range(N_samples):
  corrsY[i] = np.amax(xcorr(X[:,0], Y[:,i]))    #correlation of each
Y with reference signal X_0
  FOM_Y[i] = (u - corrsY[i])/sigma               #Figure Of Merit of
each Y
  corrsZ[i] = np.amax(xcorr(X[:,0], Z[:,i]))    #correlation of each
Z with reference signal X_0
  FOM_Z[i] = (u - corrsZ[i])/sigma               #Figure Of Merit of
each Z
```

A cada iteração é computado o valor máximo, *amax()*, do array de correlações do grupo Y e Z com um sinal de referência X, a partir da função xcorr(). Para esse trabalho o primeiro sinal do grupo de calibração foi escolhido como referência e o resultado dessa operação foi armazenado nos arrays *FOM_Y* e *FOM_Z* para servir de base de dados na construção do histograma.

Os histogramas exibem a variação dos valores das FOMs, no eixo x, e com que frequência elas ocorrem, no eixo y. Ou seja, é mostrada a distinção entre os sinais dos grupos Y e Z com relação a X. Portanto, se há uma perturbação no ambiente - nesse caso simulada a partir do ruído artificial -, teremos valores distintos

nas séries dos grupos Y e Z. Por consequência, as FOMs desses grupos também serão distintas e o valor de seus histogramas irão se distanciar.

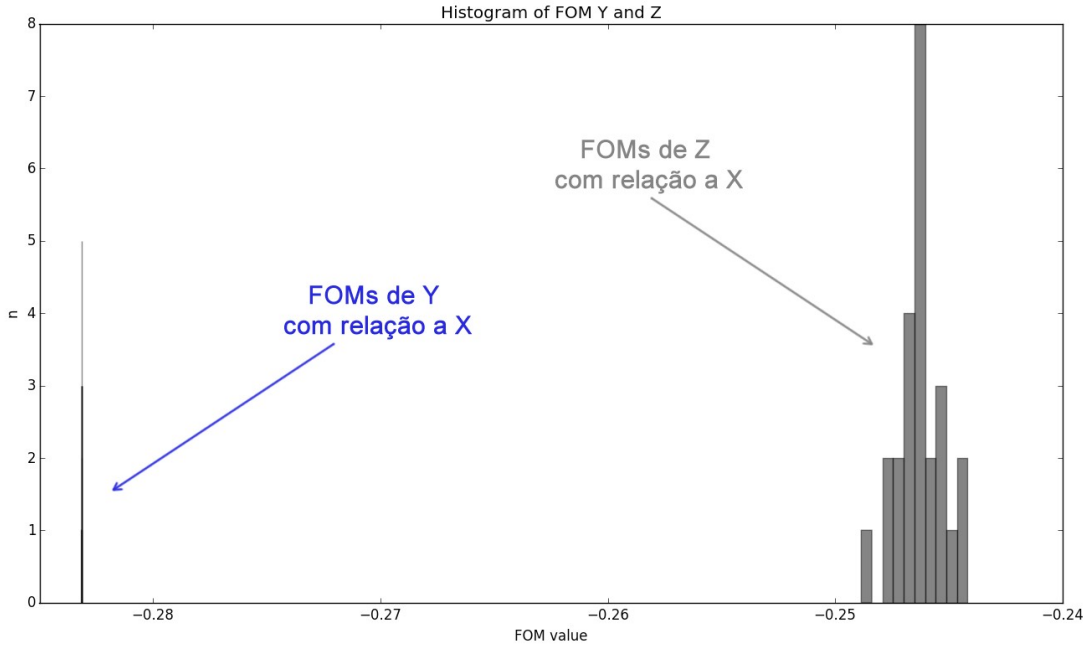

Figura 7: FOMs de Y e Z simulada a partir de um ruído artificial.

3.6 Aplicação em ambiente real

Já que a fase de simulação alcançou resultados satisfatórios com a detecção e separação das FOMs, podemos prosseguir com os testes agora em um ambiente real. Na fase da aplicação em ambiente real foi testada de fato a viabilidade do experimento analisando as respostas reais do ambiente.

3.6.1 Objetos perturbadores

A princípio, os objetos perturbadores tratavam-se de um conjunto de caixas empilhadas e uma cadeira com alguns objetos e roupas nos experimentos da sala do professor e no quarto, dias 1 e 2, respectivamente. O objeto perturbador deveria dispor de uma área e volume considerável. Tal característica serviu para facilitar o processo de detecção nos primeiros testes. No terceiro dia, no laboratório, os

objetos perturbadores variavam de uma a duas pessoas. As dimensões aproximadas desses estão descritas no quadro abaixo:

Dia	Objeto	Dimensões (LxAxP)
1	Caixas empilhadas	30cm x 44cm x 54cm + 23cm x 24cm x 31cm
2	Cadeira	58cm x 100cm x 50cm
3	Pessoas	1.65m (pessoa 1) e 1.75m (pessoa 2)

Tabela 3.1: Dimensões aproximadas dos objetos perturbadores.

Figura 8: Objetos perturbadores do dia 1 (esquerda) e dia 2 (direita).

3.6.2 Análise do código

Antes de analisar o código-fonte, devemos levar em consideração que:

1. Toda a emissão e recepção dos sinais construídos em python foi feita e analisada a partir de funções da biblioteca *sounddevice*;

2. As funções de geração do pulso (*pulse_gen*) e correlação dos sinais (*xcorr*) se mantiveram as mesmas pois o tratamento dos sinais é o mesmo;
3. Entre a execução dos grupos X, Y e Z houve a necessidade de aplicar um intervalo de tempo para que as mudanças necessárias fossem feitas no ambiente. Por exemplo, para inserir o objeto perturbador ou sair da sala; e que,
4. Por limitações no hardware, os experimentos foram realizados com o notebook dentro da sala.

Após essas considerações podemos partir para a definição das funções do código *real_ping_test.py* (disponível no Apêndice A - Códigos-fonte, “Aplicação em ambiente real”).

Começamos com a execução dos pulsos onde cada grupo emite seus pulsos a partir de um laço de repetição:

```
for i in range(N_samples):          #Grupo de calibração da sala
  x[i] = sd.playrec(pulse)          #Executa e grava o pulso
  print("Sinal " + str(i+1))
  sd.sleep(int(waveform_duration*1100))
X = np.array(x)
```

A reprodução e gravação é feita com a função *playrec()* do módulo que é executada de forma paralela. Para garantir que um sinal gravado não interfira no outro utilizamos a função *sleep()* informando um tempo em milissegundos como parâmetro. As 25 séries são armazenadas em uma variável, nesse caso X, para posteriormente calcular a média e o desvio padrão. O processo é feito de forma paralela mas alguns procedimentos prévios com a gravação tornam os valores iniciais das séries com valor vazio. Isso não pode acontecer pois interferiria diretamente no valor da média e do desvio padrão.

```
i_cut = int(0.337/Ts)      #Dead time cut index of X, Y and Z
X = X[:, i_cut:]
Y = Y[:, i_cut:]
Z = Z[:, i_cut:]

#Standard deviation and mean of calibration group
sigma = np.std(X)
```

```
u = np.mean(X)
```

A variável *i_cut* que está diretamente ligada à taxa de amostragem do programa serve como índice para o corte desses primeiros termos das séries. As funções *std()* e *mean()* do módulo *numpy* ficam responsáveis pelo cálculo do desvio padrão e da média do grupo de calibração.

Após o corte das séries de cada um dos grupos e o cálculo da média e do desvio padrão, podemos enfim correlacionar as séries e calcular suas figuras de mérito:

```
### TEST STEP
for i in range(N_samples):
  corrsY[i] = np.amax(xcorr(X[0].flatten(), Y[i].flatten()))
#correlation of each Y with reference signal X_0
  FOM_Y[i] = (u - corrsY[i])/sigma              #Figure Of Merit of
each Y
  corrsZ[i] = np.amax(xcorr(X[0].flatten(), Z[i].flatten()))
#correlation of each Z with reference signal X_0
  FOM_Z[i] = (u - corrsZ[i])/sigma              #Figure Of Merit of
each Z
```

A função *flatten()* retorna uma cópia do array em uma única dimensão e também pertence ao módulo numpy.

Finalmente as séries estão prontas para serem analisadas. O software armazena uma descrição visual do pulso emitido e um sinal de cada um dos grupos de calibração (X[0]), controle (Y[0]) e teste (Z[0]), assim como o histograma das FOMs de Y e Z em arquivos *png*.

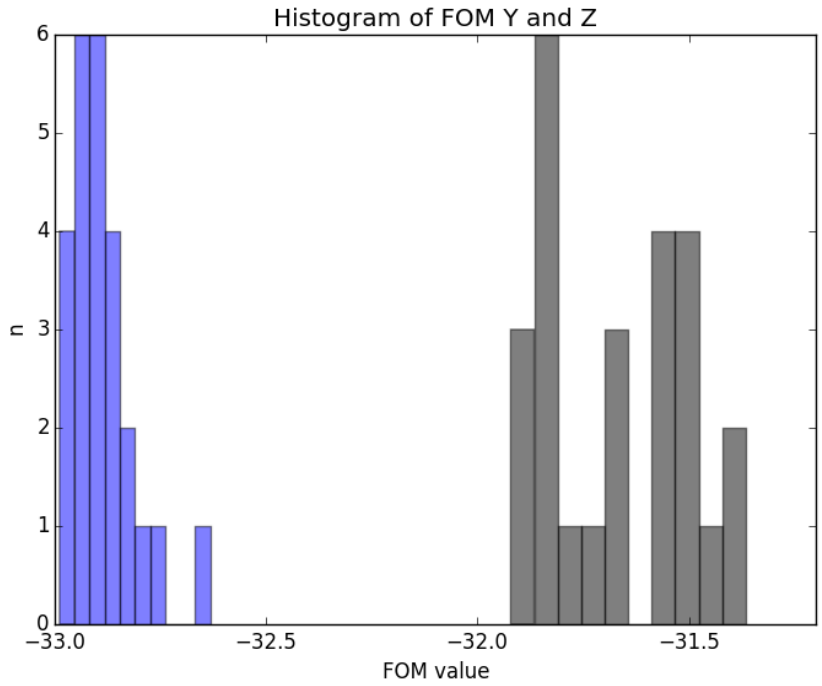

Figura 9: FOM de Y em relação a X (em azul) e FOM de Z em relação a X (em cinza).

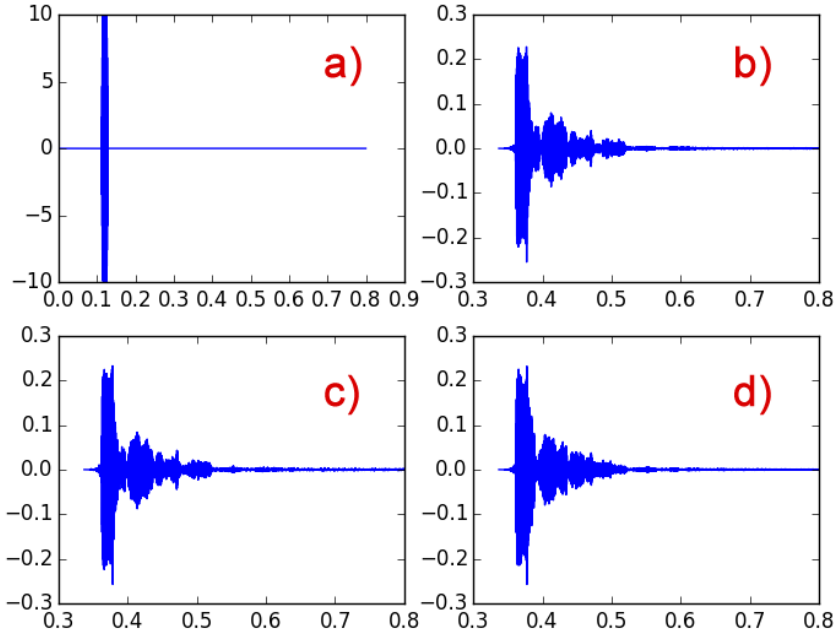

Figura 10: O pulso emitido (figura a), a série X[0] (figura b), Y[0] (figura c) e Z[0] (figura d).

Para fins de consulta ou debug, o software também armazena o valor das 25 FOMs de Y e Z em relação a X, ambas em arquivos *txt*.

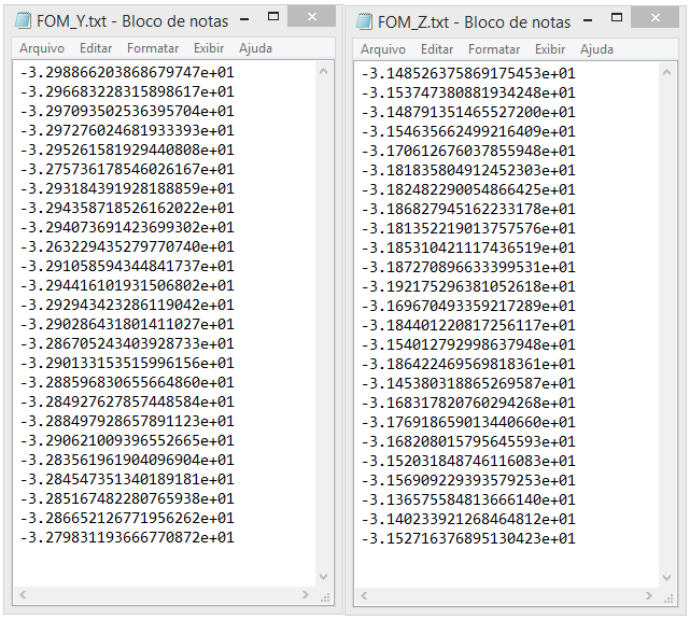

Figura 11: Arquivos das FOMs de cada série armazenados em disco.

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Prosseguimos então para a análise dos resultados obtidos nas fases de simulação e aplicação. Foram realizados testes em 3 (três) ambientes distintos para calibrar os parâmetros das variáveis de geração do pulso. A cada dia era observado qual resposta era obtida e assim seus parâmetros eram adaptados para o teste seguinte (vide Apêndice B - Estrutura dos diretórios de teste).

4.1 Teste de volume

Um dos primeiros testes foi calibrar o volume ideal para o experimento. A potência sonora pode ter relação direta nas reflexões da onda na parede ou em objetos, alterando a análise das respostas. Sabe-se que, quanto maior a potência sonora, maior são as chances de se detectar um ruído externo, por exemplo, na sala ao lado, pois o pulso poderia facilmente penetrar a parede. Por outro lado, quanto menor a potência sonora, menor a precisão do experimento, pois o sinal se dissiparia mais facilmente no ambiente. É necessário encontrar um bom balanceamento desses quesitos. Os testes as seguir foram feitos com a variação de 50% e 100% da potência dos alto-falantes.

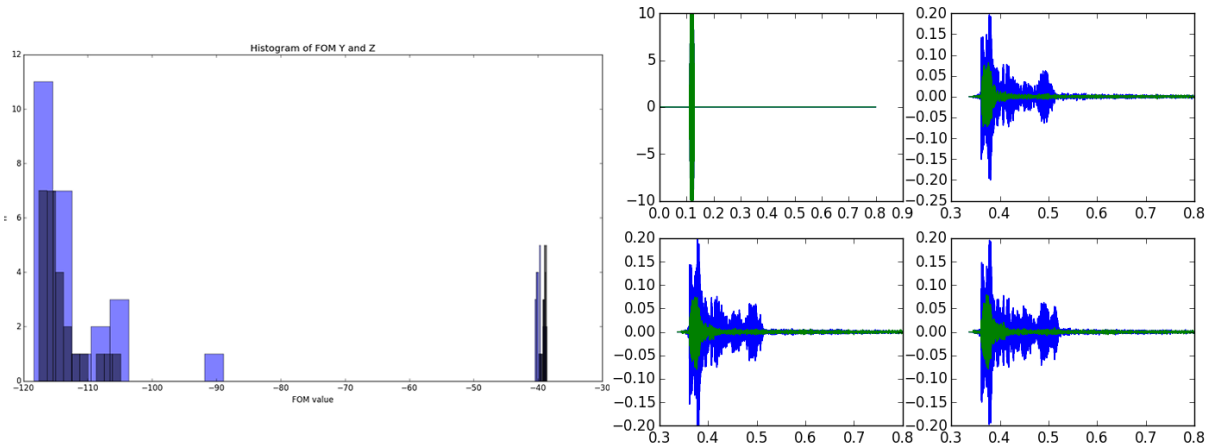

Figura 12: Experimento realizado com 25 pulsos e potência dos alto-falantes em 50%.

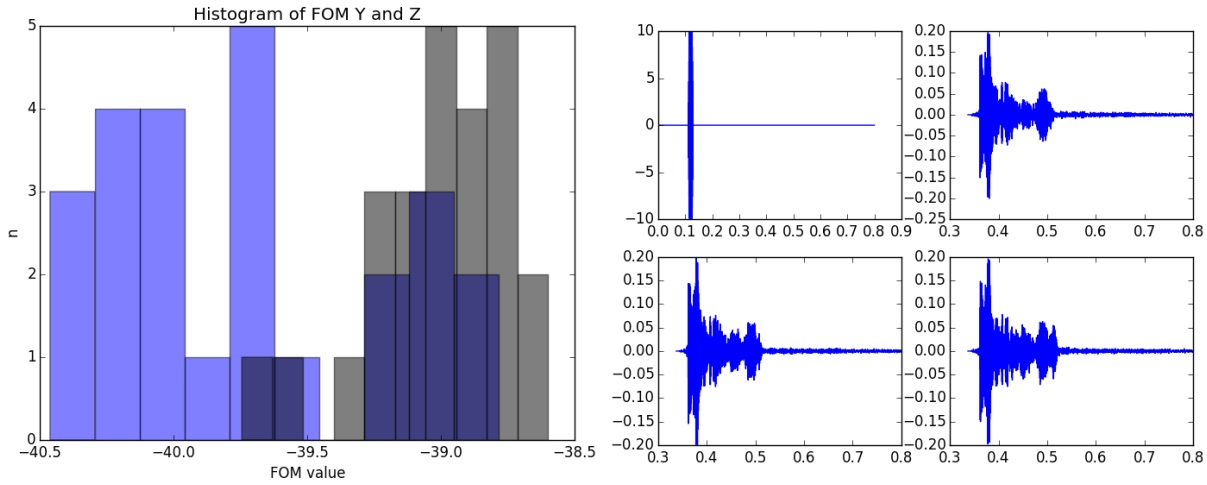

Figura 13: Experimento realizado com 25 pulsos e potência dos alto-falantes em 100%.

O volume foi adaptado ao tamanho da sala, e, como podemos observar acima, após as execuções para a potência dos alto-falantes, verificamos que as FOMs de Y e Z se encontram muito próximas com 50% da potência sonora emitida (figura 12) enquanto para o teste com potência máxima (figura 13), as FOMs alcançam valores mais distantes.

Dessa forma o parâmetro escolhido foi 100% de potência nos alto-falantes da máquina de teste e 10 como parâmetro de amplitude na função de geração do pulso. Esses valores foram suficientes para obter êxito nas outras execuções.

```
A = 10.0  #pulse amplitude
```

4.2 Teste da quantidade de pulsos

O parâmetro seguinte para ajuste é a quantidade de pulsos emitidos. Deve haver um balanceamento entre o número de pulsos para que nenhuma perturbação relativamente grande ou o tempo de execução interferisse no processo de calibração da sala. Os testes para os dias 1 e 2 foram feitos com uma variação de 5, 25 e 50 pulsos emitidos por grupo.

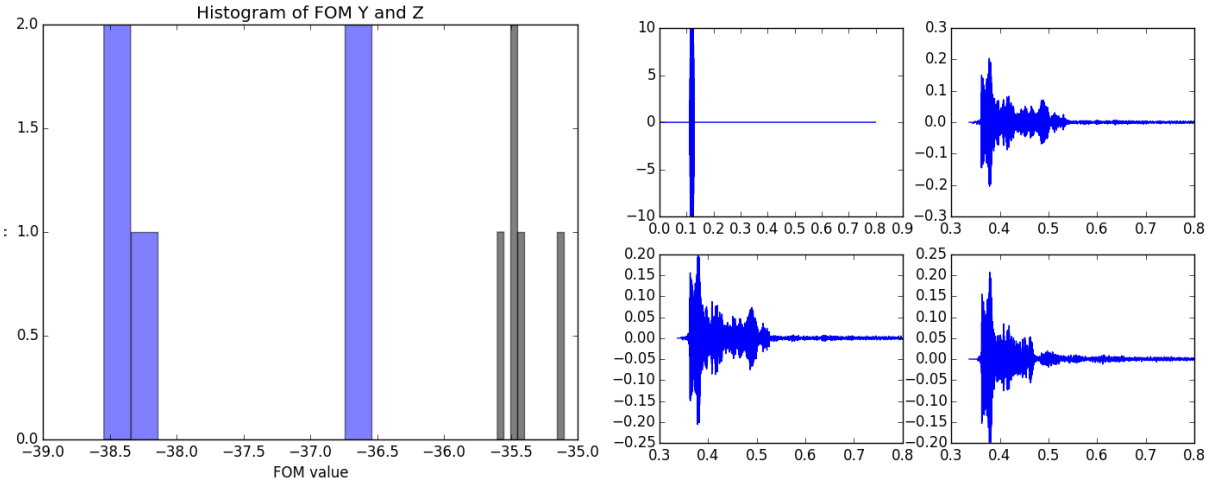

Figura 14: Experimento realizado com 5 pulsos no primeiro dia de testes.

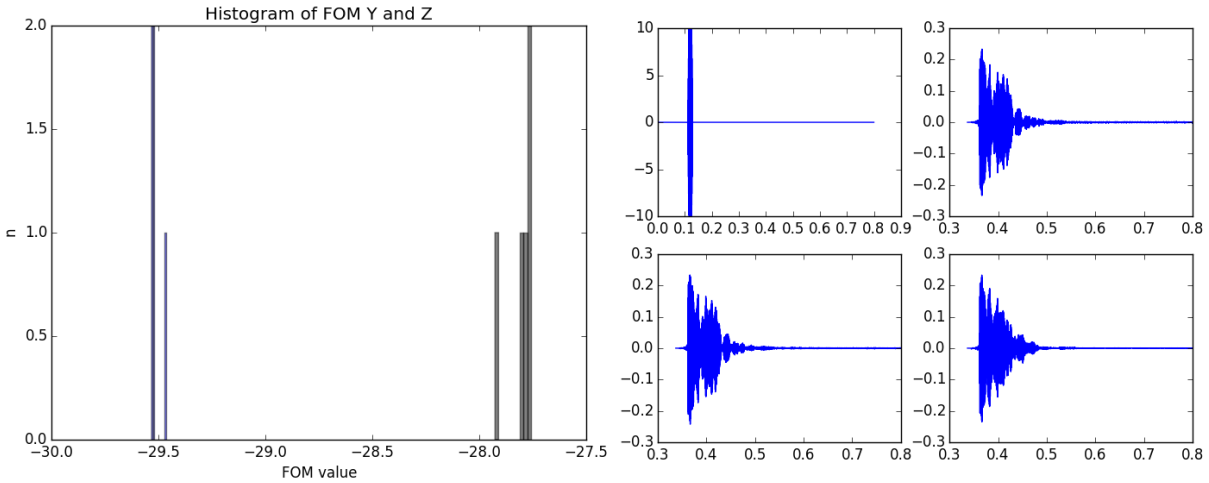

Figura 15: Primeiro experimento realizado com 5 pulsos no segundo dia de testes.

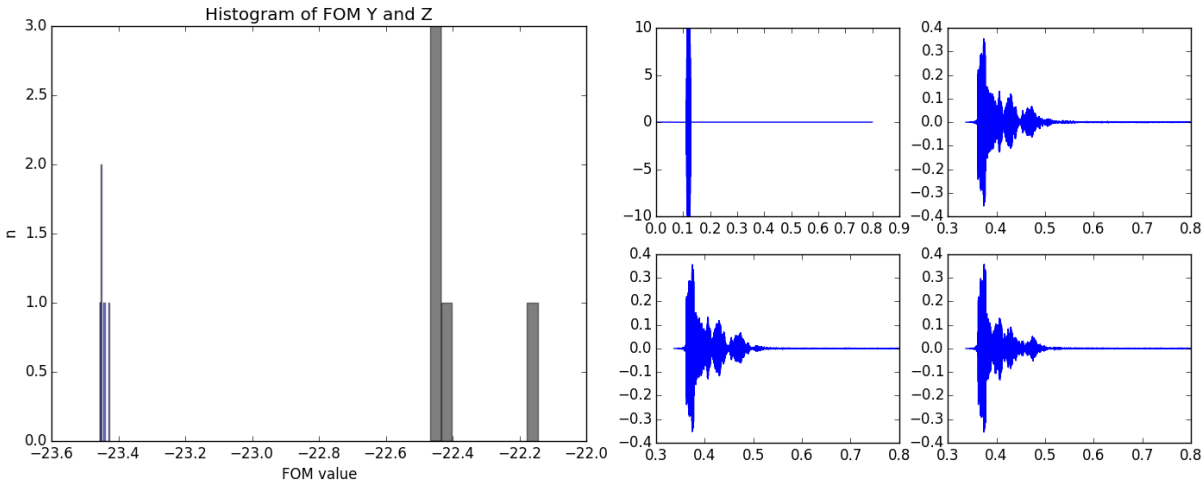

Figura 16: Segundo experimento realizado com 5 pulsos no segundo dia de testes.

Nas figuras acima observamos que apesar das FOMs se encontrarem suficientemente separadas, o número de sinais (cinco) seria pouco e facilmente alterado por perturbação ou ruído externo. Tentaremos agora com 50 pulsos.

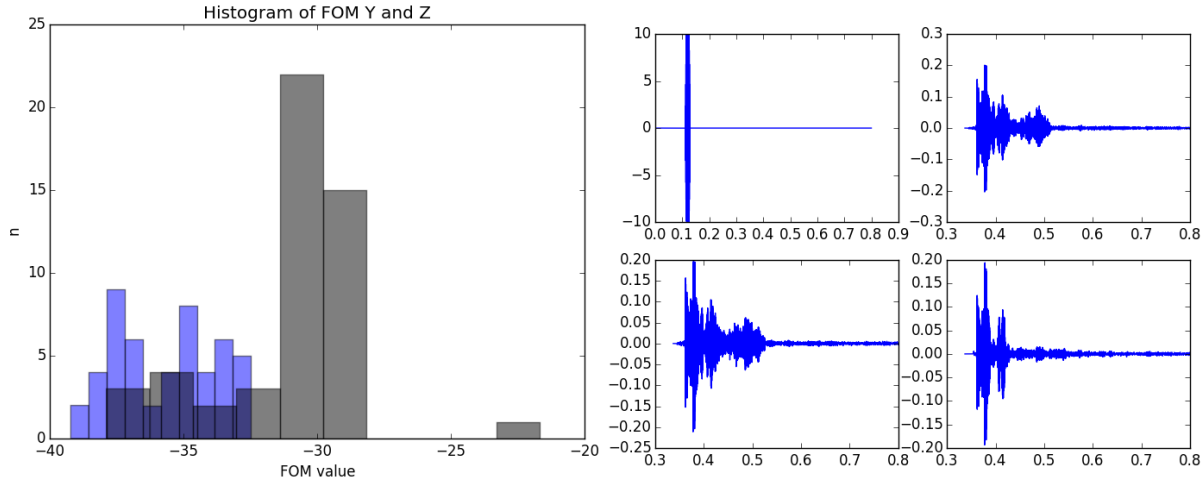

Figura 17: Experimento realizado com 50 pulsos no primeiro dia de testes.

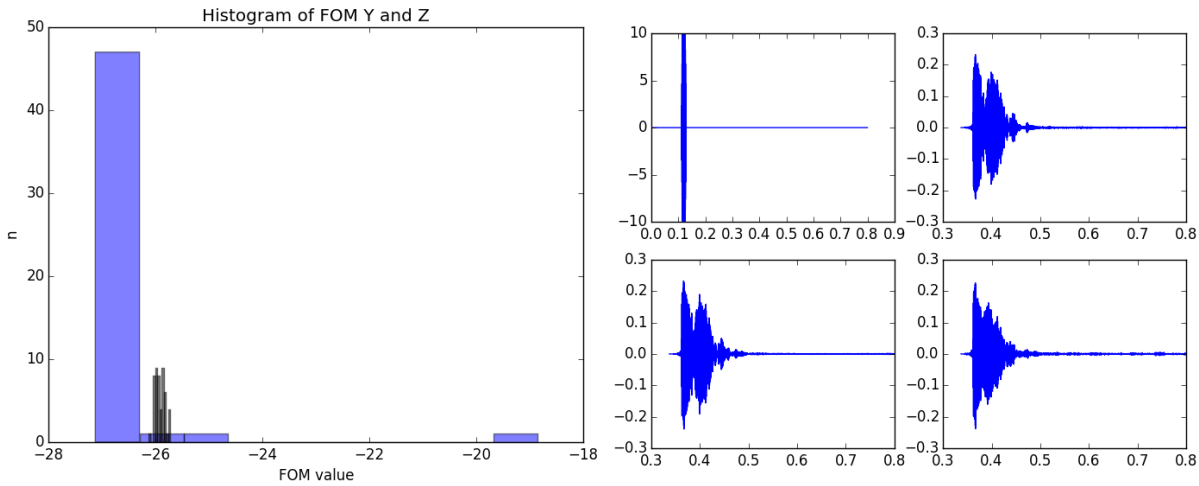

Figura 18: Primeiro experimento realizado com 50 pulsos no segundo dia de testes.

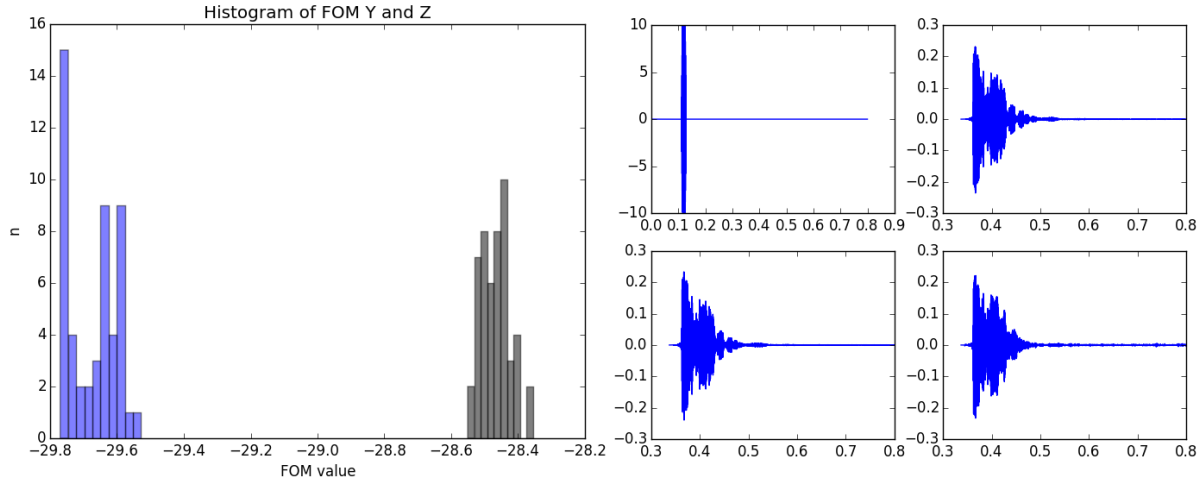

Figura 19: Segundo experimento realizado com 50 pulsos no segundo dia de testes.

Da mesma forma, apesar dos bons resultados, verificou-se que uma quantidade exagerada de pulsos emitidos seria desperdício de memória e principalmente desperdício de tempo para calibração, visto que cada pulso é emitido e analisado em cerca de 2 segundos, havendo assim maior probabilidade de uma perturbação externa ocorrer nesse tempo. Os testes com 25 pulsos emitidos foram suficientes e igualmente satisfatórios.

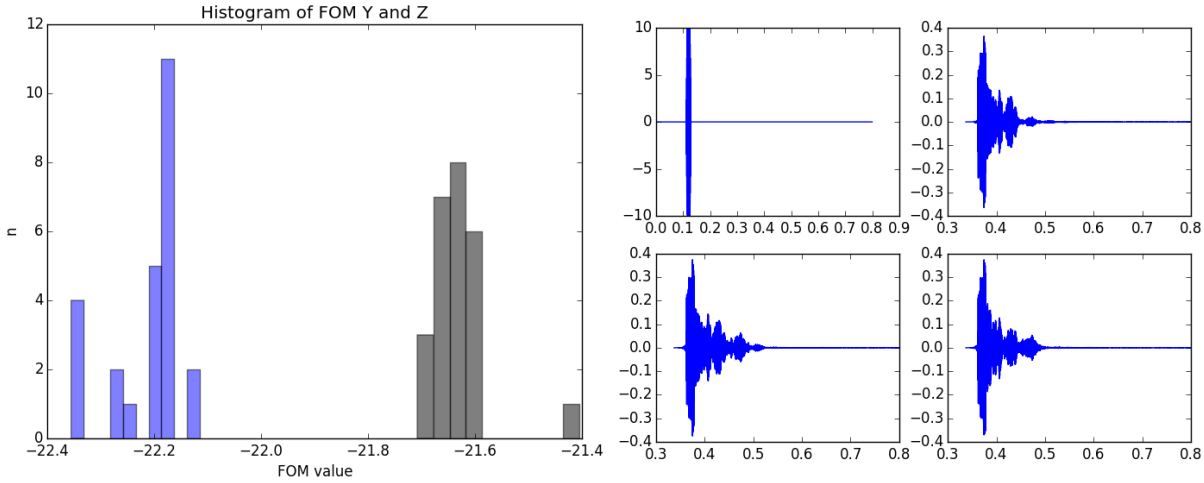

Figura 20: Primeiro experimento realizado com 25 pulsos no segundo dia de testes.

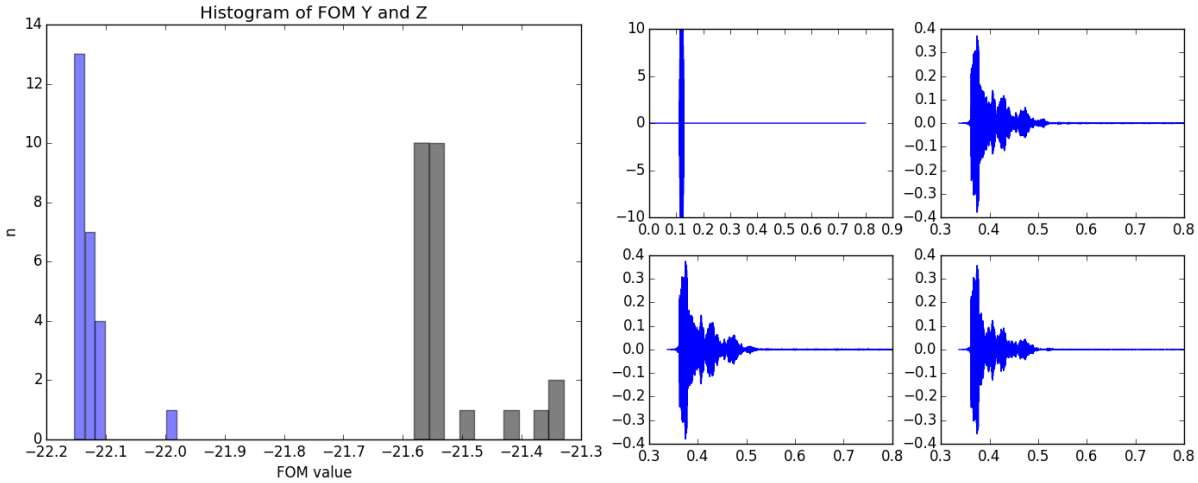

Figura 21: Segundo experimento realizado com 25 pulsos no segundo dia de testes.

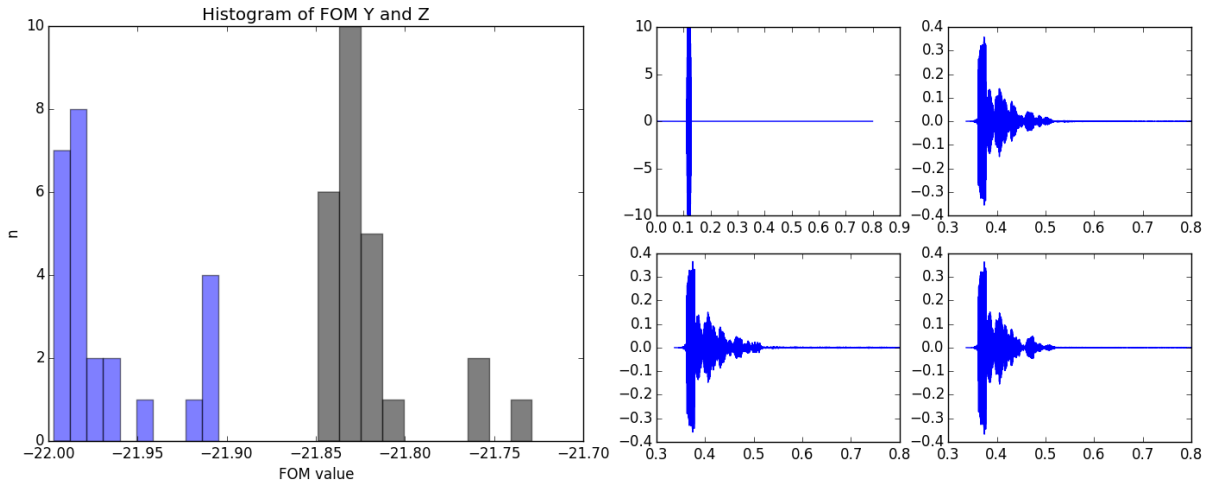

Figura 22: Terceiro experimento realizado com 25 pulsos no segundo dia de testes.

O software foi capaz de detectar até uma perturbação pequena (figura 22). A diferença dos sinais dos grupos de calibração, controle e teste, visualmente são imperceptíveis, todavia os seus histogramas ainda continuam separados a uma distância que varia de acordo com o tamanho do objeto perturbador. Após a análise dos resultados acima, ajustamos o parâmetro para o terceiro dia de testes com 25 pulsos.

4.3 Teste de precisão

O próximo teste verificou dentre outros aspectos a precisão do sistema, ou seja, quais perturbações e em que condições o sistema foi capaz de detectar um “intruso”. O terceiro dia de testes foi realizado em um laboratório do Centro de Informática.

Com o número de pulsos escolhidos dessa vez, podemos agora variar as perturbações em tamanho e posição. O software foi executado variando de 0 a 2 pessoas no ambiente, simulando um objeto em repouso inserido, e se posicionando na frente e por trás do emissor.

O primeiro teste executado no terceiro dia foi a última verificação de funcionalidade do software. O ambiente ficou sem perturbação durante a execução dos grupos de calibração, controle e teste. Já que nenhuma perturbação deveria ser detectada era de se esperar que os histogramas finais se sobrepusessem. E foi o

que obtivemos.

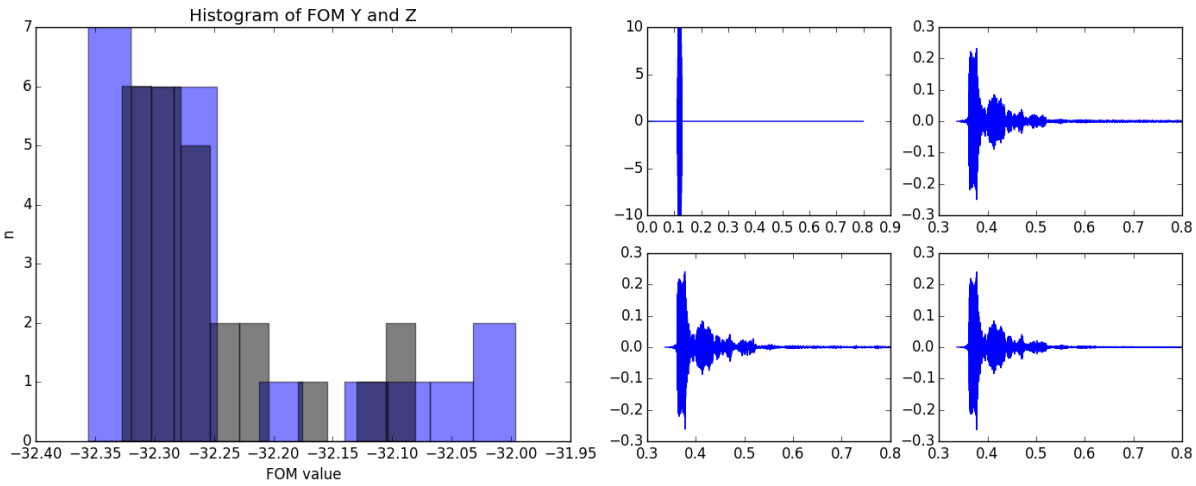

Figura 23: Experimento realizado com 25 pulsos e 0 pessoas no terceiro dia de testes.

Podemos então prosseguir com os testes. Ainda não sabemos qual a sensibilidade do software a perturbações grandes e pequenas. A sensibilidade foi testada gradativamente. Os experimentos abaixo foram executados com perturbações de duas pessoas posicionadas em frente ao emissor.

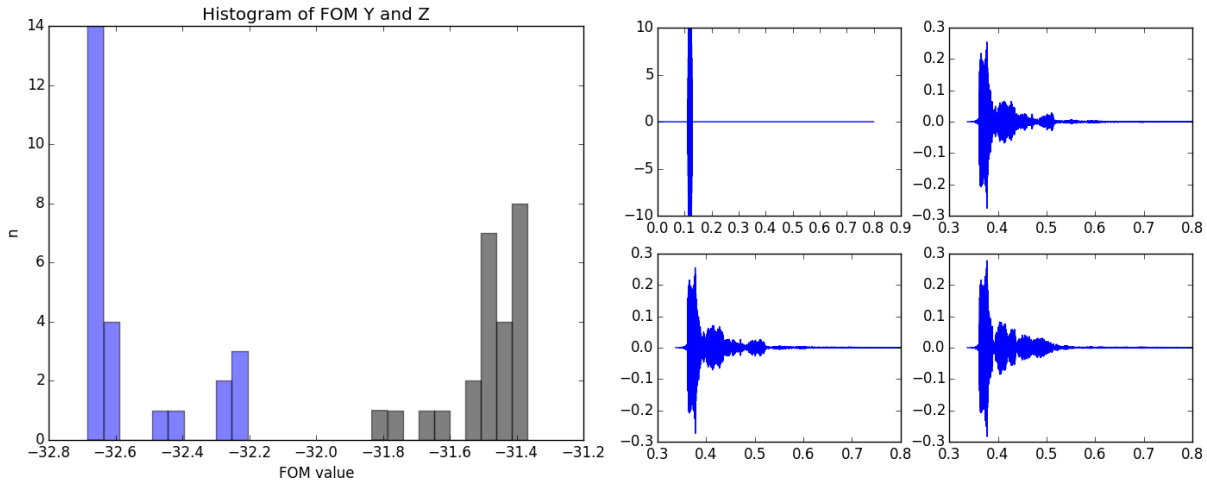

Figura 24: Primeiro experimento realizado com 25 pulsos e 2 pessoas no terceiro dia de testes.

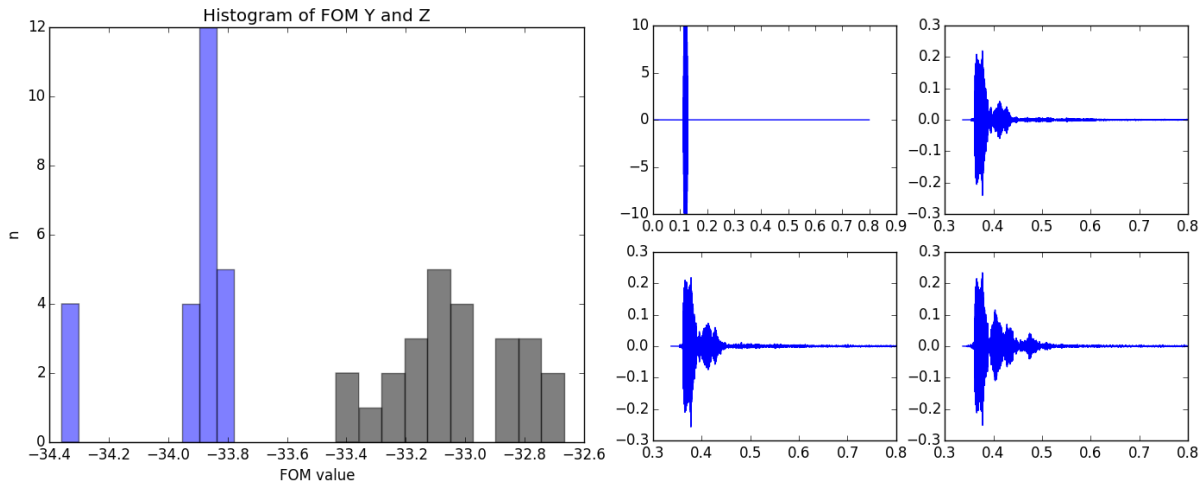

Figura 25: Segundo experimento realizado com 25 pulsos e 2 pessoas no terceiro dia de testes.

Como constatado acima, foram realizados dois testes com 2 pessoas como objeto perturbador. Para esses dois, o software também conseguiu detectar diferença entre as FOMs dos grupos Y e Z. É importante lembrar que apesar dos sinais de referência se encontrarem ligeiramente parecidos a olho nu, o valor de suas funções de média e desvio padrão mudam, consequentemente a correlação e o cálculo da FOM também muda.

Outra questão importante que pode ser levantada é onde posicionar o emissor para conseguir melhores resultados. O próximo teste diminui o objeto perturbador para uma pessoa e posiciona-o em frente mas também por trás do emissor.

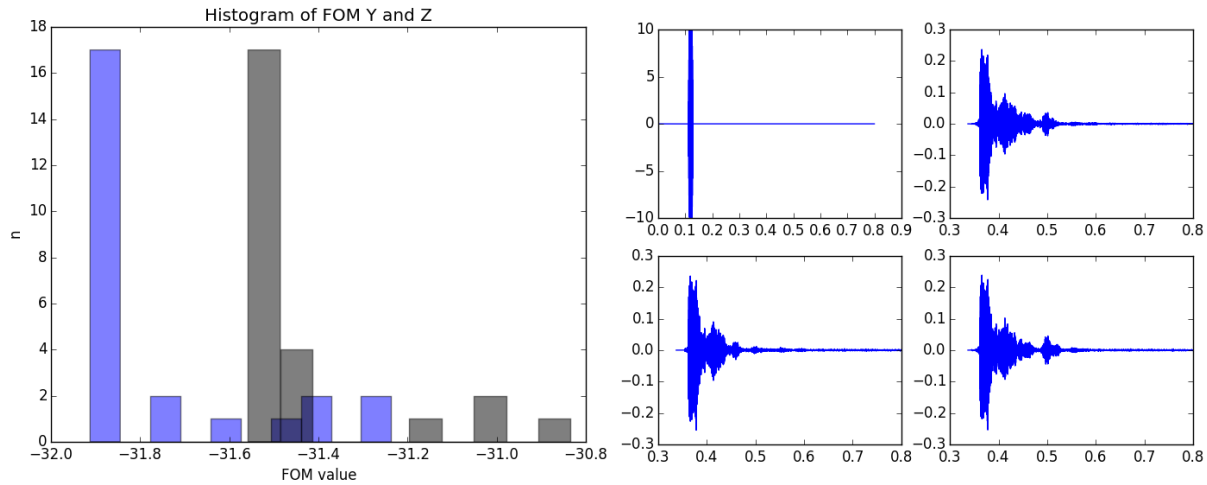

Figura 26: Experimento realizado com 25 pulsos e 1 pessoa atrás do emissor no terceiro dia de testes.

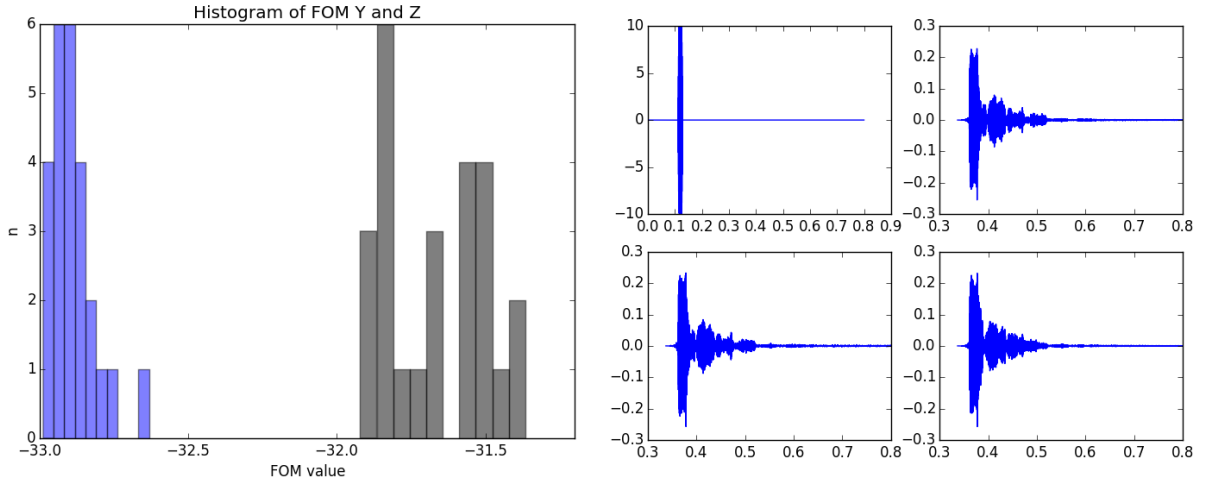

Figura 27: Experimento realizado com 25 pulsos e 1 pessoa em frente ao emissor no terceiro dia de testes.

Podemos lembrar que o pulso está sendo emitido por alto-falantes simples posicionados em um ângulo perpendicular à parede, então, perturbações que acontecem atrás desse emissor podem não ser detectadas nessa configuração, devido às reflexões não chegarem ao receptor com tempo e potência suficientes. É o que podemos observar na dessemelhança entre as figuras 26 e 27. Entretanto, esta última figura nos mostra que mesmo com uma perturbação menor que a anterior (2 pessoas), o software ainda apresentou um ótimo resultado.

5 CONCLUSÕES E TRABALHOS FUTUROS

O sistema se mostrou eficaz e apresentou bons resultados nos experimentos realizados em frente ao emissor e sem muita perturbação no ambiente. A utilização de “recursos amortizados”, como computadores e celulares, é o que se destaca nessa aplicação. Após o desenvolvimento, constatamos que o sistema se adequa a ambientes privados como laboratórios ou escritórios, por exemplo, onde esses equipamentos já se encontram disponíveis. Alia-se então, um sistema de segurança a um baixo ou até mesmo nulo investimento.

Quando se fala da perturbação, pôde-se observar que mesmo com alguns resultados não tão satisfatórios, o sistema respondeu como esperado a cada mudança dos parâmetros, como foi o caso da perturbação atrás do emissor, por exemplo. Nos experimentos dos dias 1 e 3, notamos que quanto maior o tamanho do objeto perturbador, maior a chance de detecção. Nesses experimentos houve uma melhor concentração das FOMs na medida que o objeto se tornou também maior. Acredita-se que a área, o volume e o material do objeto também interfira no resultado de forma que as ondas propagadas terão dificuldade em atravessá-lo e serão mais refletidas.

A sensibilidade do software à ruídos externos e pequenas perturbações foi um dos quesitos de cuidado e que mais preocuparam a execução dos experimentos. Conforme fomos realizando os experimentos onde tínhamos o laboratório e a sala do professor como ambiente de teste, podemos ver que os ruídos externos dos corredores e das salas vizinhas poderiam interferir no sinal recebido. No dia 2 os experimentos realizados em casa não demonstraram resultado contrário pois foram feitos na madrugada onde todos os moradores da casa estavam em repouso.

Em adição, seria essencial uma calibração fina dos parâmetros e poderia ser um ótimo tema para trabalhos futuros. A análise sistemática dos parâmetros, variando um enquanto os outros estão fixos e observando seus resultados, dada as

condições da estrutura do ambiente, pode trazer resultados ainda melhores. A escolha do sinal cossenoidal emitido também pode ser estudado e comparado a outros tipos como, por exemplo, o CHIRP. Este, por ser baseado na emissão simultânea de frequências baixas e altas, possui características únicas que podem ser viáveis ao projeto se aplicadas corretamente. É válida uma discussão sobre a relação entre os parâmetros do objeto perturbador e como eles influenciariam nos resultados.

Por fim, um trabalho utilizando técnicas de inteligência artificial, como *Deep Learning*, capaz de aprender e calibrar os seus parâmetros automaticamente nas primeiras etapas, possivelmente tornaria a aplicação adaptável a qualquer estrutura física.

6 REFERÊNCIAS

1 “**What’s CHIRP Sonar?**”. West Marine. 2019. Disponível em: https://www.westmarine.com/WestAdvisor/Understanding-CHIRP-Scanning-Sonar. Acesso em 11/09/2019.

2 “**Python**”. Wikipedia. 2019. Disponível em: https://pt.wikipedia.org/wiki/Python#Implementa%C3%A7%C3%B5es. Acesso em 11/09/2019.

3 “**numpy.std — NumPy v1.17 Manual**”. Scipy.org. 2019. Disponível em: https://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html. Acesso em 11/09/2019.

4 “**python-sounddevice**”. 2019. Disponível em: https://python-sounddevice.readthedocs.io/en/0.3.13/usage.html. Acesso em 11/09/2019.

5 KLEIN DOS SANTOS, Alexandro. Os IDE's (Ambientes de Desenvolvimento Integrado) como ferramentas de trabalho em informática. **Universidade Federal de Santa Maria.** Disponível em: http://www-usr.inf.ufsm.br/~alexks/elc1020/artigo-elc1020-alexks.pdf. Consultado em 11/09/2019.

6 CASAVELLA, Eduardo. “**Ambientes Integrados de Desenvolvimento em Linguagem C**”. Intellectuale. Disponível em: http://linguagemc.com.br/ambientes-integrados-de-desenvolvimento-em-linguagem-c/. Acesso em 11/09/2019.

7 “**GitHub - spyder-ide/spyder: Official repository for Spyder - The Scientific Python Development Environment**”. GitHub. 2019. Disponível em: https://github.com/spyder-ide/spyder. Acesso em 11/09/2019.

8 “**John Cage**”. Wikipedia. 2019. Disponível em: https://pt.wikipedia.org/wiki/John_Cage. Acesso em 11/09/2019.

9 KOZINN, Allan. “**John Cage, 79, a Minimalist Enchanted With Sound, Dies**”. The New York Times. 1992. Disponível em: *https://archive.nytimes.com/www.nytimes.com/learning/general/onthisday/bday/0905.html*. Consultado em 11/09/2019.

10 “**No noise, only Sound**”. Guernica. 2015. Disponível em: https://www.guernicamag.com/no-noise-only-sound/. Acesso em 11/09/2019.

11 PONTES, A. C. F.. “ENSINO DA CORRELAÇÃO DE POSTOS NO ENSINO MÉDIO”. 2010. **19º SINAPE**. Universidade Federal do Acre. Consultado em 11/09/2019.

12 FALCÃO, A. J. T.. “Detecção de Correlação e Causalidade em Séries Temporais não Categóricas“. 2012. **Universidade Nova de Lisboa**. Acesso em: 11/09/2019.

13 FILHO, L. M. A. L.. “Correlação e Regressão” (PDF). Universidade Federal da Paraíba. Pág.6. Disponível em: http://www.de.ufpb.br/~luiz/AED/Aula9.pdf. Consultado em 11/09/2019.

14 Gel'fand, I.M.; Yaglom, A.M.. 1957. “**Calculation of amount of information about a random function contained in another such function**”. American Mathematical Society Translations: Series 2. **12**: 199–246 English translation of original in Uspekhi Matematicheskikh Nauk **12** (1): 3-52.

15 Peng, H.C., Long, F., and Ding, C.. 2005. “**Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy”**. IEEE Transactions on Pattern Analysis and Machine Intelligence. **27** (8): 1226–1238.

16 “**Informação mútua**”. Wikipedia. 2018. Disponível em: https://pt.wikipedia.org/wiki/Informa%C3%A7%C3%A3o_m%C3%BAtua. Acesso em 11/09/2019.

17 SILVA, D. C. M.. "**Interferência de ondas**". Brasil Escola. Disponível em: https://brasilescola.uol.com.br/fisica/interferencia-ondas.htm. Acesso em 11/09/2019.

7 APÊNDICE A - CÓDIGOS-FONTE

1. Criação do pulso

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 03 18:56:08 2019

@author: Danillo
"""

import numpy as np
import matplotlib.pyplot as plt

Ts = 0.01 # período de amostragem
f = 5.0  # frequência da portadora
L = 3.0 # meia largura
twopi = 2*np.pi

x = np.arange(-4,4, Ts)
envelope = np.exp(-(x/L)**10)
signal = np.cos(twopi*f*x)
pulse = envelope*signal

plt.figure(0)
plt.plot(x, envelope, label = 'Supergauss')
plt.plot(x, signal, label = 'Cos signal')
plt.legend()
plt.figure(1)
plt.plot(x, pulse, label = 'Pulse')
plt.legend()
```

2. Simulação em ambiente ideal

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 27 00:49:51 2019

@author: Danillo
"""

import numpy as np
import matplotlib.pyplot as plt
from numpy.random import normal

fs = 48000
t0 = 0.028
pulse_duration = 0.007
waveform_duration = 0.050 #1.05
f = 2200.0
```

```
Ts = 1.0/fs
N = int(waveform_duration/Ts)+1
A = 10.0
t = np.linspace(0.0, waveform_duration, N)
noise_amp = 0.01      #Natural noise amplitude (1%)
noise_amp_dist = 0.1    #Disturbed noise amplitude (10%)
N_samples = 25   #número de séries


def pulse_gen(t, A):
  return A*np.cos(2*np.pi*f*t)*np.exp(-((t-t0)/pulse_duration)**10)

def xcorr(a1, a2):
  """ Calcula a função de correlação de Pearson de dois arrays a1 e a2:
    C = Sum a1[i]*a2[i-j] /(sqrt(sigma_a1 *sigma_a2)) """
  Num_points = np.amin([len(a1), len(a2)])
  mu_a1 = np.mean(a1)
  mu_a2 = np.mean(a2)
  a1_local = a1-mu_a1
  a2_local = a2-mu_a2
  A1 = np.std(a1_local)
  A2 = np.std(a2_local)
  X = np.correlate(a1_local, a2_local, mode = 'same')/(Num_points*A1*A2)
  return X


### CALIBRATION STEP
X = np.zeros((N,N_samples))   #Calibration group
Y = np.zeros((N,N_samples))   #Control group
Z = np.zeros((N,N_samples))   #Test group
corrsY = np.zeros(N_samples)  #Y correlations
corrsZ = np.zeros(N_samples)  #Z correlations
FOM_Y = np.zeros(N_samples)   #Y FOM's
FOM_Z = np.zeros(N_samples)   #Z FOM's
for i in range(N_samples):
  X[:,i] = pulse_gen(t, A) + normal(0, noise_amp*A, N)
  Y[:,i] = pulse_gen(t, A) + normal(0, noise_amp*A, N)
  Z[:,i] = pulse_gen(t, A) + normal(0, noise_amp_dist*A, N)

#Standard deviation and mean of calibration group
sigma = np.std(X)
u = np.mean(X)

### TEST STEP
for i in range(N_samples):
  corrsY[i] = np.amax(xcorr(X[:,0], Y[:,i]))    #correlation of each Y with
reference signal X_0
  FOM_Y[i] = (u - corrsY[i])/sigma              #Figure Of Merit of each Y
  corrsZ[i] = np.amax(xcorr(X[:,0], Z[:,i]))    #correlation of each Z with
reference signal X_0
  FOM_Z[i] = (u - corrsZ[i])/sigma              #Figure Of Merit of each Z
```

```
num_bins = 10

# the histogram of the data
plt.hist(FOM_Y, num_bins, facecolor='blue', alpha=0.5)
plt.hist(FOM_Z, num_bins, facecolor='black', alpha=0.5)

plt.xlabel('FOM value')
plt.ylabel('n')
plt.title(r'Histogram of FOM Y and Z')
plt.legend()
# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()
```

3. Aplicação em ambiente real

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 02 11:41:49 2019

@author: Danillo
"""

import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt

fs = 48000
t0 = 0.120
pulse_duration = 0.007
waveform_duration = 0.800    #800ms
f = 2200.0
Ts = 1.0/fs
N = int(waveform_duration/Ts)+1
A = 10.0
t = np.linspace(0.0, waveform_duration, N)
N_samples = 25       #número de séries
channels = 1        #número de canais
record_amp = 100.0     #amplitude da gravação


#Set default sample frequency and channel for sd.play and sd.rec
sd.default.samplerate = fs
sd.default.channels = channels

def pulse_gen(t, A):
  return A*np.cos(2*np.pi*f*t)*np.exp(-((t-t0)/pulse_duration)**10)

def xcorr(a1, a2):
  """ Calcula a função de correlação de Pearson de dois arrays a1 e a2:
    C = Sum a1[i]*a2[i-j] /(sqrt(sigma_a1 *sigma_a2)) """
```

```
  Num_points = np.amin([len(a1), len(a2)])
  mu_a1 = np.mean(a1)
  mu_a2 = np.mean(a2)
  a1_local = a1-mu_a1
  a2_local = a2-mu_a2
  A1 = np.std(a1_local)
  A2 = np.std(a2_local)
  X = np.correlate(a1_local, a2_local, mode = 'same')/(Num_points*A1*A2)
  return X


### CALIBRATION STEP
x = N_samples*[N*[0.0]]       #Calibration group
y = N_samples*[N*[0.0]]       #Control group
z = N_samples*[N*[0.0]]       #Test group
corrsY = np.zeros(N_samples)  #Y correlations
corrsZ = np.zeros(N_samples)  #Z correlations
FOM_Y = np.zeros(N_samples)   #Y FOM's
FOM_Z = np.zeros(N_samples)   #Z FOM's

pulse = pulse_gen(t, A)

print("VAI COMEÇAR...")
sd.sleep(10000)
print("GRUPO DE CALIBRAÇÃO")
for i in range(N_samples):                    #Grupo de calibração da sala
  x[i] = sd.playrec(pulse)                   #executa e grava o pulso
  print("Sinal " + str(i+1))
  sd.sleep(int(waveform_duration*1100))     #tempo entre uma execução e
outra
X = np.array(x)
print("GRUPO DE CONTROLE")
for i in range(N_samples):                    #Grupo de controle da sala
  y[i] = sd.playrec(pulse)                   #executa e grava o pulso
  print("Sinal " + str(i+1))
  sd.sleep(int(waveform_duration*1100))     #tempo entre uma execução e
outra
Y = np.array(y)
sd.sleep(40000)
print("GRUPO DE TESTES")
for i in range(N_samples):                    #Grupo de controle da sala
  z[i] = sd.playrec(pulse)                   #executa e grava o pulso
  print("Sinal " + str(i+1))
  sd.sleep(int(waveform_duration*1100))     #tempo entre uma execução e
outra
Z = np.array(z)

i_cut = int(0.337/Ts)      #indice de corte do tempo morto X, Y e Z
X = X[:, i_cut:]
Y = Y[:, i_cut:]
Z = Z[:, i_cut:]

plt.figure(1)
```

```
plt.subplot(221)
plt.plot(t, pulse, label = 'Pulse (array)')
plt.subplot(222)
plt.plot(t[i_cut:], X[0], label = 'Pulse (array)')
plt.subplot(223)
plt.plot(t[i_cut:], Y[0], label = 'Pulse (array)')
plt.subplot(224)
plt.plot(t[i_cut:], Z[0], label = 'Pulse (array)')
plt.savefig("Pulses.png")

#Standard deviation and mean of calibration group
sigma = np.std(X)
u = np.mean(X)

### TEST STEP
for i in range(N_samples):
  corrsY[i] = np.amax(xcorr(X[0].flatten(), Y[i].flatten()))
#correlation of each Y with reference signal X_0
  FOM_Y[i] = (u - corrsY[i])/sigma              #Figure Of Merit of each Y
  corrsZ[i] = np.amax(xcorr(X[0].flatten(), Z[i].flatten()))
#correlation of each Z with reference signal X_0
  FOM_Z[i] = (u - corrsZ[i])/sigma              #Figure Of Merit of each Z

num_bins = 10

# the histogram of the data
plt.figure(2)
plt.hist(FOM_Y, num_bins, facecolor='blue', alpha=0.5)
plt.hist(FOM_Z, num_bins, facecolor='black', alpha=0.5)
plt.xlabel('FOM value')
plt.ylabel('n')
plt.title(r'Histogram of FOM Y and Z')
plt.savefig("HIST.png")
# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()

np.savetxt("FOM_Y.txt", FOM_Y);
np.savetxt("FOM_Z.txt", FOM_Z);
```

8 APÊNDICE B - ESTRUTURA DOS DIRETÓRIOS DE TESTE

- **Dia 1 - Sala do professor**
    - ↳ 5 pulsos
    - ↳ 25 pulsos
        - ↳ vol50
        - ↳ vol100
    - ↳ 50 pulsos
- **Dia 2 - Casa**
    - ↳ 5 pulsos
        - ↳ teste1
        - ↳ teste2
    - ↳ 25 pulsos
        - ↳ teste1
        - ↳ teste2
        - ↳ teste3
    - ↳ 50 pulsos
        - ↳ teste1
        - ↳ teste2
- **Dia 3 - LABEC**
    - ↳ 25 pulsos
        - ↳ 0 pessoas
        - ↳ 1 pessoas
            - Na frente da caixa de som
            - Por trás da caixa de som
        - ↳ 2 pessoas
            - teste1
            - teste2