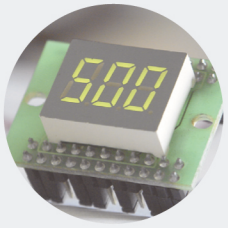
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Sistema de aquisição do PCG, miniaturizado, portátil

LUÍS MANUEL BAPTISTA PINHEIRO
Outubro de 2020

POLITÉCNICO
DO PORTO

Sistema de aquisição do PCG, miniaturizado, portátil

Luís Manuel Baptista Pinheiro

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização em Telecomunicações
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2020

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Luís Manuel Baptista Pinheiro, Nº 1141185, 1141185@isep.ipp.pt
Orientação científica: António Avelino Amorim Marques, aav@isep.ipp.pt

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização em Telecomunicações
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2020

Agradecimentos

Quero agradecer ao meu orientador, engenheiro António Avelino Amorim Marques, pela sua disponibilidade, orientação, profissionalismo, pelo rigor e por todo o apoio dado.

Agradeço também à minha família e amigos por todo o incentivo, pela amizade, paciência e todo o apoio incondicional durante estes anos de estudo.

Resumo

As doenças cardiovasculares são a principal causa de morte na Europa e em Portugal. A prática de auscultação dos sons emitidos pelo coração é de grande importância para identificação de doenças cardíacas e em caso de mau funcionamento permite estabelecer estratégias iniciais de combate à doença.

Este projeto teve como finalidade desenvolver um sistema que adquire e regista os batimentos cardíacos em tempo real de forma de identificar anomalias cardíacas.

O projeto desenvolvido é uma aplicação multiplataforma capaz de funcionar com conexão a um dispositivo externo do PCG para captura dos sinais emitidos pelo coração ou pode adquirir os sinais através da aplicação com auxílio de um microfone. O sinal recebido através do microfone é condicionado, através de filtros digitais de forma a obter o sinal com a melhor qualidade possível para análise do paciente. A aplicação representa graficamente o sinal sonoro capturado e a sua *Fast Fourier Transform*. Os sons capturados podem ser exportados sob forma de um ficheiro .wav para uma análise sonora do sinal.

Palavras-Chave: Fast Fourier Transform, Filtros digitais, PCG, sons do coração, ficheiros .wav.

Abstract

Cardiovascular diseases are the leading cause of death in Europe and Portugal. The practice of auscultation of the sounds emitted by the heart is of great importance for the identification of heart diseases and in case of malfunction, it is possible to establish initial strategies to combat the disease.

This project aimed to develop a system that acquires and records heartbeats in real time in order to identify cardiac anomalies.

The developed project is a multiplatform application capable of working with connection to an external device of the PCG to capture the signals emitted by the heart or it can acquire the signals through the application with the assistance of a microphone. The signal received through the microphone is conditioned, through digital filters in order to obtain the signal with the best quality possible for analysis of the patient.

The application graphically represents the captured sound signal and its Fast Fourier Transform. The captured sounds can be exported in the form of a .wav file for a sound analysis of the signal.

Keywords: Fast Fourier Transform, digital filters, PCG, heart sounds, .wav files.

Conteúdo

Agradecimentos **iii**

Conteúdo **ix**

Lista de Figuras **xi**

Lista de Tabelas **xiii**

Acrónimos **xv**

1 Introdução **1**
1.1 Contextualização 1
1.2 Objetivos 2
1.3 Calendarização 2
1.4 Organização do relatório 3

2 Estado da Arte **5**
2.1 Sistema cardiovascular 5
2.1.1 Coração 6
2.2 Fonocardiograma (PCG) 7
2.3 Sons cardíacos 8
2.4 Python 10
2.4.1 Kivy 11
2.4.2 Linguagem Kivy (Kv) 11
2.4.3 Criação de *builds* da aplicação 12
2.4.4 Kivy Launcher 13
2.4.5 PyAudio 13
2.4.6 NumPy 14
2.4.7 Socket 14
2.4.8 SciPy 15
2.4.9 Filtro Butterworth 15
2.4.10 Teorema de amostragem de Nyquist 16
2.4.11 Filtro filtfilt 16

2.5 Fourier Transform (FT) 16
2.5.1 Fast Fourier Transform (FFT) 17
2.6 Protótipo do sistema miniaturizado de PCG 17
2.7 Soluções de PCG miniaturizados existentes no mercado 18
2.7.1 Estetoscópio de teleconsulta CMS-VESD 18
2.7.2 Estetoscópio para cardiologia eKuore Pro 19

3 Desenvolvimento do sistema de aquisição do PCG 21
3.1 Objetivo e requisitos do sistema de aquisição do PCG 21
3.2 Aplicação de aquisição de processamento 22
3.3 Fluxograma da aplicação 22
3.3.1 Funcionamento da aplicação 22
3.3.2 Funcionamento do ecrã de menu 23
3.3.3 Funcionamento do ecrã de ligação 23
3.3.4 Funcionamento do ecrã de análise de sinal com conexão ao dispositivo externo 24
3.3.5 Funcionamento do ecrã de análise de sinal sem conexão a dispositivo externo 25
3.4 Condicionamento de sinal 27
3.5 Ligação ao dispositivo externo 28
3.6 Obtenção do sinal sonoro através do microfone 28

4 Validação e resultados 31
4.1 Validação da conexão da aplicação 31
4.2 Validação do condicionamento de sinal 32
4.3 Validação da FFT 35
4.4 Validação da captura do sinal sonoro 37
4.4.1 Validação do condicionamento de sinal aplicado ao sinal sonoro 39
4.5 Protótipo da aplicação 41
4.5.1 Ecrã de menu 41
4.5.2 Ecrã de ligação 42
4.5.3 Ecrã de análise de sinal 44

5 Conclusões 49
5.1 Conclusões do trabalho 49
5.2 Propostas de melhoria e funcionalidades 50

Bibliografia 51

Lista de Figuras

2.1 Representação do coração [1] 6
2.2 Diagrama de blocos PCG[2] 8
2.3 Focos de auscultação [3] 9
2.4 Sinais PCG de um sinal cardíaco [4] 10
2.5 Python [5] 11
2.6 Kivy [6] 11
2.7 Exemplo de uma ligação socket em Python[7] 14
2.8 Resposta do filtro Butterworth[8] 15
2.9 Protótipo do PCG miniaturizado (frente) [9] 17
2.10 Protótipo do PCG miniaturizado (back) [9] 18
2.11 Estetoscópio de teleconsulta CMS-VESD [10] 18
2.12 Estetoscópio para cardiologia eKuore Pro [11] 19

3.1 Arquitetura simplificada do sistema PCG 22
3.2 Fluxograma simplificado do funcionamento da Aplicação 23
3.3 Fluxograma do ecrã de ligação 24
3.4 Fluxograma do ecrã de análise de sinal com ligação 25
3.5 Fluxograma do ecrã de análise de sinal sem ligação 27

4.1 Validação da receção da mensagem do servidor 31
4.2 Validação da ligação do cliente (aplicação) ao servidor 31
4.3 Filtro Butterworth 32
4.4 Sinal de entrada e saída do filtro a 2 kHz 33
4.5 Sinal de entrada e saída do filtro a 1 kHz 34
4.6 Sinal de entrada e saída do filtro a 150 Hz 34
4.7 Sinal de entrada e saída do filtro a 100 Hz 35
4.8 Sinal f(t) 36
4.9 Sinal FFT 36
4.10 sinal s(t) 37
4.11 sinal FFT filtrado 37
4.12 Ficheiro .wav criado com nome de ”voice” 38
4.13 Confirmação do sinal gravado com 5 segundos 38
4.14 Representação do sinal sonoro capturado através do microfone. 39

4.15 Ficheiros .wav com condicionamento de sinal 40
4.16 Confirmação da duração dos ficheiros .wav com condicionamento de sinal 40
4.17 Representação do sinal sonoro capturado com condicionamento de sinal 41
4.18 Representação do ecrã de menu 42
4.19 Representação do ecrã de ligação 43
4.20 Representação do *pop-up* de erro de conexão 43
4.21 Representação Ecrã de análise de sinal 44
4.22 Representação Ecrã de análise de sinal S1 e S2 45
4.23 Representação do *pop-up* de ruído e alteração do estado do ”status” . . 46
4.24 Ficheiros .wav ”problemSound” e ”problemSound-filtered” 46
4.25 Representação do *pop-up* de exportação do som capturado 47
4.26 Ficheiros .wav ”problemHeartSound” e ”problemHeartSound-filtered” . 47

Lista de Tabelas

1.1 Calendarização 3

Acrónimos

APK Android Package

DFT Discrete Fourier Transform

ECG eletrocardiograma

FFT Fast Fourier Transform

FT Fourier Transform

GPU Graphics Processing Unit

IFT inverse Fourier Transform

I/O Input/Output

IP Internet Protocol

MIT Instituto de Tecnologia de Massachusetts

NDK Native Development Kits

NUI Natural User Interface

OS Sistema operativo

PCG Fonocardiograma

PF Par de Fourier

SDK Software Development Kits

TCP Transmission Control Protocol

TUIO Tangible User Interface Objects

UDP User Datagram Protocol

UI User Interface

Capítulo 1

Introdução

No 2º Ano de mestrado em Engenharia Eletrotécnica e de Computadores (MEEC) na área de especialização de Telecomunicações, no Instituto Superior de Engenharia do Porto (ISEP) é pedida uma dissertação cujo tema foi o desenvolvimento de um sistema PCG que permite a aquisição, registo e análise do sinal sonoro emitido pelo coração humano.

1.1 Contextualização

O desenvolvimento desta dissertação surgiu da motivação de elaborar um sistema na área da telemedicina, nomeadamente um sistema de monitorização de sons emitido pelo coração humano que permitisse analisar de forma simples o estado do coração e detetar anomalias cardíacas.

As doenças cardiovasculares continuam a ser a principal causa de morte na Europa e em Portugal. Em Portugal, morrem cerca de 35 mil portugueses anualmente vítimas de doença cardiovascular, representando assim um terço da mortalidade na população nacional.

A prática da auscultação é realizada normalmente através de um estetoscópio. É uma técnica médica indispensável no exame clínico nos hospitais e centros de saúde. Esta prática não invasiva é uma mais valia no custo-eficácia. A sua aprendizagem é de grande importância uma vez que as alterações identificadas no comportamento do coração são frequentemente indicadoras de doenças cardíacas e permitem estabelecer estratégias iniciais de combate à doença.

Um dos principais problemas do diagnóstico através do estetoscópio tradicional é a falta de sensibilidade do ouvido humano particularmente na identificação de sons de baixa frequência. Dado isto e com objetivo de melhorar a precisão da auscultação do som emitido pelo coração, surgiu o Fonocardiograma (PCG) que permite obter o som emitido pelo coração e separar cada componente de frequência individual do sinal produzido.

A alta taxa de transmissão de COVID-19 levou à necessidade da criação/otimização de novos modelos de assistência à saúde de forma a diminuir contacto entre o médico e o paciente. Neste contexto surge a telemedicina, uma tecnologia essencial nos sistemas de saúde para acompanhamento e diagnóstico de pacientes. Assim sendo, é necessário o desenvolvimento de ferramentas adicionais que possibilitem aperfeiçoar o atendimento médico, através de uma avaliação mais extensiva do estado de saúde dos pacientes.

Nesta dissertação é apresentado o desenvolvimento de uma tecnologia para auscultação e análise de sons emitidos pelo coração.

1.2 Objetivos

O objetivo principal deste projeto consiste no desenvolvimento de um sistema miniaturizado, portátil de aquisição fonocardiograma capaz de o registar e analisar em tempo real.
O sistema deverá ter uma interface capaz de mostrar ao utilizador o sinal emitido pelo coração e exportar o sinal sob forma de ficheiro sonoro. Para o bom funcionamento e uma boa otimização do projeto final, foi necessário subdividir o objetivo principal em diferentes tarefas, tais como:

- Estudo do sistema cardiovascular;
- Estudo dos sons emitidos pelo coração e as suas zonas de auscultação;
- Estudo do sistema PCG;
- Estudo de produtos existentes no mercado;
- Estudo das tecnologias para desenvolvimento do projeto;
- Estudo de sistemas de aquisição de sinal sonoro;
- Estudo de processamento de sinal;
- Desenvolvimento de um sistema de aquisição de sinal sonoro;
- Desenvolvimento de uma aplicação de PCG;
- Testes e validação.

1.3 Calendarização

Na calendarização fez-se referência aos processos que levaram ao desenvolvimento final da aplicação desenvolvida, na tabela 1.1 está representado o planeamento.

Tabela 1.1: Calendarização

	2020								
	Fevereiro	Março	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro
Estudo do projeto									
Estudo das tecnologias utilizadas no projeto									
Estudo do protótipo do sistema miniaturizado de PCG									
Estudo de produtos existentes no mercado									
Teste inicial desenvolvido									
Fase de implementação final									
Implementação de novas funcionalidades									
Testes finais									
Escrita do relatório									

1.4 Organização do relatório

O relatório encontra-se dividido em cinco capítulos:

- Introdução - é feita uma contextualização do problema, e uma exposição dos objetivos do trabalho desenvolvido; refere-se ainda à calendarização e à organização do relatório.
- Estado da Arte - é apresentada uma análise ao sistema cardiovascular, tecnologias de aquisição do sinal sonoro emitido pelo coração e sua análise, um estudo das tecnologias que levaram ao desenvolvimento do projeto, uma exposição do protótipo do sistema miniaturizado de PCG previamente desenvolvido e algumas soluções de PCG existentes no mercado.
- Desenvolvimento do sistema de aquisição do PCG - expõe os seus objetivos, fluxogramas da aplicação desenvolvida e explicando o funcionamento do condicionamento de sinal e como se procedeu a ligação ao dispositivo externo.
- Validação e Resultados - valida todas as implementações feitas na aplicação e demonstra o resultado final da aplicação e o seu funcionamento.
- Conclusão - que apresenta todas as conclusões do trabalho desenvolvido e possíveis melhorias ao projeto.

Capítulo 2

Estado da Arte

Neste capítulo são abordadas tecnologias que foram importantes para a elaboração do sistema de aquisição e análise do sinal cardíaco.

Para melhor interpretação dos dados foram estudadas a constituição e funções do sistema cardiovascular, e também foi realizado um estudo sobre análise sobre a constituição e funcionamento do coração.

Abordou-se o funcionamento, vantagens e constituição do Fonocardiograma (PCG).

Fez-se uma investigação aos sons e sinais cardíacos para uma melhor compreensão e interpretação dos dados recolhidos.

Apresentou-se a linguagem de programação que levou ao desenvolvimento deste projeto, bem como a *framework* utilizada e algumas bibliotecas que se considerou ter uma maior relevância.

Foi apresentado e estudado projeto de PCG, miniaturizado, portátil anteriormente desenvolvido.

Por fim, foram analisados produtos existentes que partilham o mesmo objetivo do projeto desenvolvido.

2.1 Sistema cardiovascular

Sistema cardiovascular, também conhecido por Sistema circulatório, é constituído pelo o coração e uma grande quantidade de vasos sanguíneos onde circula o sangue. Contribuí para equilibrar o organismo (homeostase) e tem como funções o transporte de oxigénio desde os pulmões e o dióxido de carbono em sentido inverso para ser expulso pelos pulmões e ser eliminado do organismo, movimentação de nutrientes que foram absorvidos e de substâncias produzidas em certos locais do organismo para os locais mais necessitados do organismo e regulação da temperatura corporal. [12]

2.1.1 Coração

O coração é o órgão central do sistema cardiovascular, localizado no centro do tórax (entre os dois pulmões, atrás do externo, ligeiramente à esquerda), tem como principal função bombear o sangue através dos vasos sanguíneos, para que forneça nutrientes e oxigénio a todas as células e que os resíduos sejam então levados a locais para a sua eliminação. [12] É um órgão oco de paredes musculosas, com um tamanho aproximado ao tamanho da mão fechada da própria pessoa. Na Figura 2.1 é apresentado um esquema da constituição do coração.

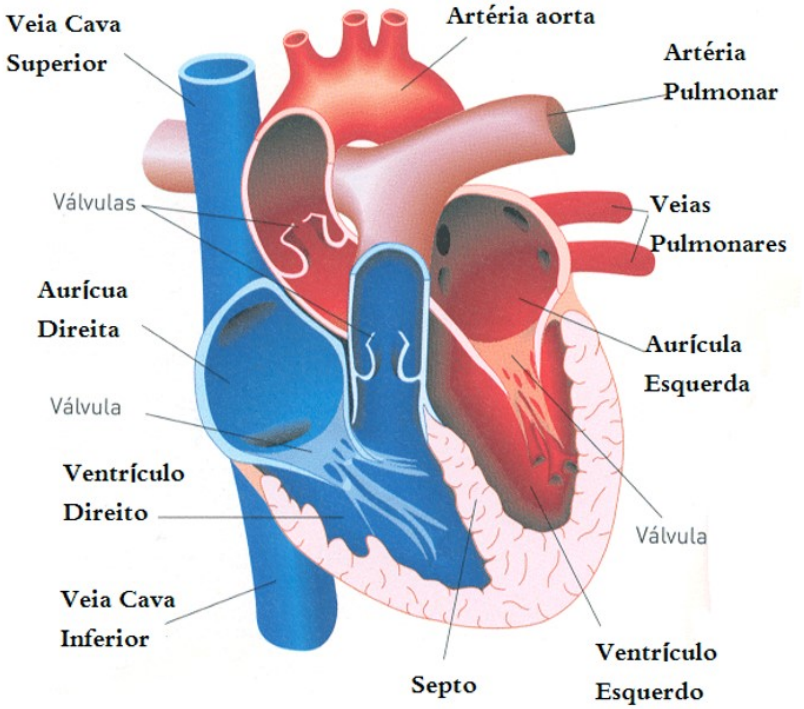


Figura 2.1: Representação do coração [1]

Na circulação do sangue podem ser distinguidos dois circuitos.

A pequena circulação, também conhecida por circulação pulmonar, o sangue venoso (pobre em oxigénio e ricos em resíduos celulares) chega ao coração pela veia cava superior (proveniente dos membros superiores) e pela veia cava inferior (oriundo dos membros inferiores), entra na aurícula direita e é empurrado do ventrículo direito para os pulmões passando na artérias pulmonares. Chegando aos pulmões o sangue é oxigenado (sangue arterial, rico em oxigénio e nutrientes) e, por fim, retorna o sangue oxigenado ao coração através da aurícula esquerda.

A grande circulação, pode também ser designada por circulação sistémica ou circulação periférica, representa o circuito posterior à chegada do sangue arterial ao coração, em que é encaminhado para o ventrículo esquerdo para ser bombeado para todos os órgãos e tecidos, recolhe o sangue venoso de volta ao coração através das veias cavas, entrando na aurícula direita.[12] [13]

O ciclo cardíaco é representado pelas sucessivas dilatações e contrações do órgão nas diversas câmaras cardíacas, indica a passagem do sangue de cada aurícula para o ventrículo do seu lado e do ventrículo para a artéria correspondente. A fase de dilatação, intitulada por diástole, significa que o coração se enche de sangue. Diminui-se a tensão das aurículas e recebem o sangue oriundo das veias. As válvulas auriculoventriculares abrem-se para que o fluxo de sangue passe de cada aurícula para o ventrículo correspondente. A fase de contração, denominada por sístole, assinala quando o coração expulsa o seu conteúdo. O fenómeno de sístole, acontece quando os ventrículos se enchem e as válvulas auriculoventriculares se fecham, para o fluxo não aconteça no sentido inverso fazendo com que o sangue volte até às aurículas. As paredes dos ventrículos dilatam, produzindo uma pressão no seu interior, o sangue acaba por provocar a abertura das válvulas arteriais. Por fim, as paredes ventriculares contraem e bombeiam o sangue para as artérias (do ventrículo direito para a artéria pulmonar e do ventrículo esquerdo para a artéria aorta).[13]

2.2 Fonocardiograma (PCG)

O PCG é uma ferramenta de análise e registo gráfico de sons e extra-sons que o coração produz no seu funcionamento, podendo detetar patologias sem ser necessário recorrer a outros exames, é um método não invasivo e não é necessário um grande tipo de especialização e habilidade técnica para utilizar. [14]

A atividade mecânica que ocorre dentro do coração e as variações de pressão do fluxo de sangue dentro do coração, dá origem a variações de toda a estrutura cardíaca que são audíveis na parede torácica e estes sons dão a indicação do estado da saúde do coração. Os sons produzidos podem ser captados na superfície do tórax, armazenados em forma de uma série temporal e exposto sob forma de um gráfico com diversas ondas. [15]

Na Figura 2.2 está representado o diagrama de blocos de um PCG, pode-se verificar que o sinal é recebido e segmentado, posteriormente pode-se gravar o ciclo, que termina com a classificação automática do sinal, ou proceder a uma classificação manual através uma representação gráfica ou auditiva do sinal.

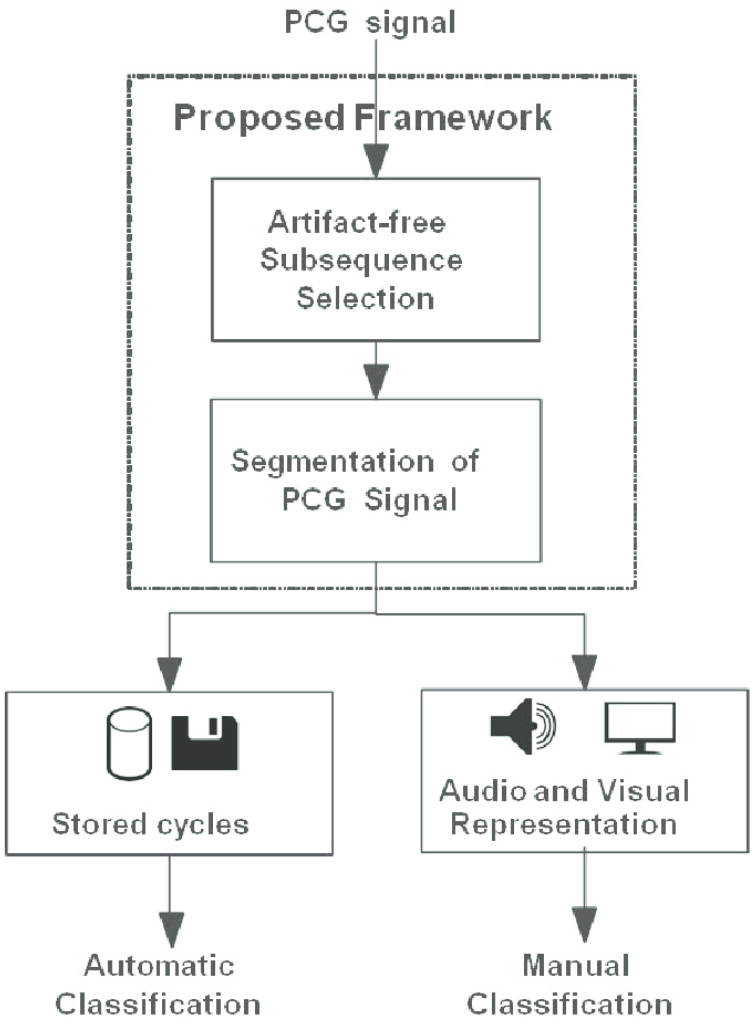


Figura 2.2: Diagrama de blocos PCG[2]

2.3 Sons cardíacos

Para auscultação de sons cardíacos são utilizados 4 focos: foco aórtico, foco pulmonar, foco tricúspide e foco mitral. Estes focos fazem referência aos comportamentos de cada uma das válvulas cardíacas. Pelo que se pode verificar pela Figura 2.3, o foco triscúpide (representado a cor de rosa) situa-se na extremidade inferior do externo ao nível do ventrículo direito. O foco mitral (representado a verde), localiza-se no quinto espaço intercostal na linha hemiclavicular que corresponde ao porção mais superficial do ventrículo esquerdo. O foco pulmonar (representado a amarelo) fica situado no segundo espaço intercostal à esquerda, ao nível do tronco pulmonar. por fim, o foco aórtico (representado a azul) situa-se no segundo espaço intercostal à direita, ao nível da aorta ascendente. Os focos

pulmonar e aórtico designam-se por focos da base. [14]

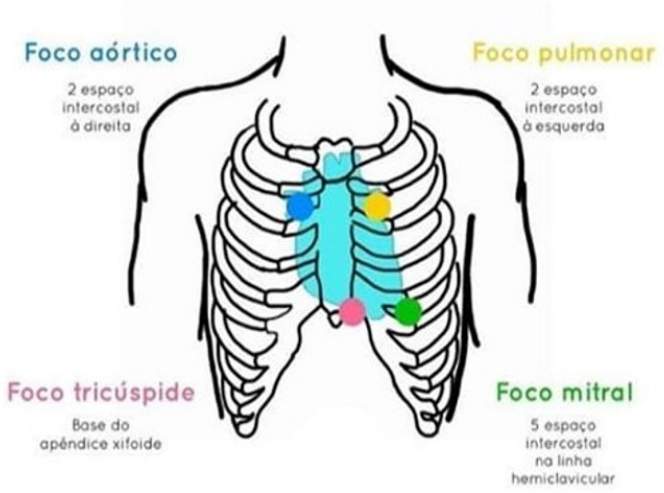


Figura 2.3: Focos de auscultação [3]

O coração em normal funcionamento, produz dois sinais distintos detetados pela auscultação, o primeiro som cardíaco (S1) e o segundo som cardíaco (S2) sendo eles sons de elevada frequência. Por outro lado, o terceiro (S3) e quarto (s4) som acústico são de baixa frequência e sons mais graves. Entre o S1 e S2 ocorre o fenómeno de sístole e entre S2 e S1 ocorre o fenómeno de diástole. O primeiro som cardíaco (S1) som de elevada frequência [50-150] Hertz (Hz) com uma duração de [70-150] milissegundos (ms), produzido quando ocorre o fecho das válvulas (início da sístole) auriculoventriculares audível em todos os focos auscultatórios. O segundo som cardíaco (S2) também um som de elevada frequência [50-200] Hz com uma duração de [60-120] ms, corresponde ao som gerado pela oclusão das válvulas aórticas e das válvulas pulmonares (fim da sístole). O terceiro som cardíaco (S3) é um som de baixa frequência [50-90] Hz com uma duração de [40-100] ms de fraca irradiação, surgindo no final da Fase de Enchimento Rápido (Protodiástole). Podendo ser produzido pelo ventrículo direito quando tem intensidade máxima no foco tricúspide, ou no ventrículo esquerdo quando tem intensidade máxima no foco mitral. Quando surge em circunstâncias patológicas designa-se Galope Protodiastólico ou Galope Ventricular. O quarto som cardíaco (S4) de baixa frequência [50-80] Hz com uma duração de [40-80] ms, ocorre cerca de 90 ms antes de S1, e fraca irradiação. Surge na Fase de contração auricular (Telediástole). Pode ter origem no ventrículo direito quando a intensidade é máxima no foco tricúspide, ou no ventrículo esquerdo, quando tem uma grande intensidade no foco mitral. O S4 pode ser designado por Galope Telediastólico ou Galope Auricular, podendo surgir num contexto de doença

cardíaca isquémica ou de diminuição da complacência ventricular. Na Figura 2.4 estão apresentados os diferentes sons cardíacos capturados por um PCG, pode-se verificar que o S1 ocorre entre o fim da diástole e o início da sístole, já o sinal S2 ocorre no fim da sístole até ao inicio da diástole. Os sinais S3 e S4 ocorrem durante a diástole.[4] [16]

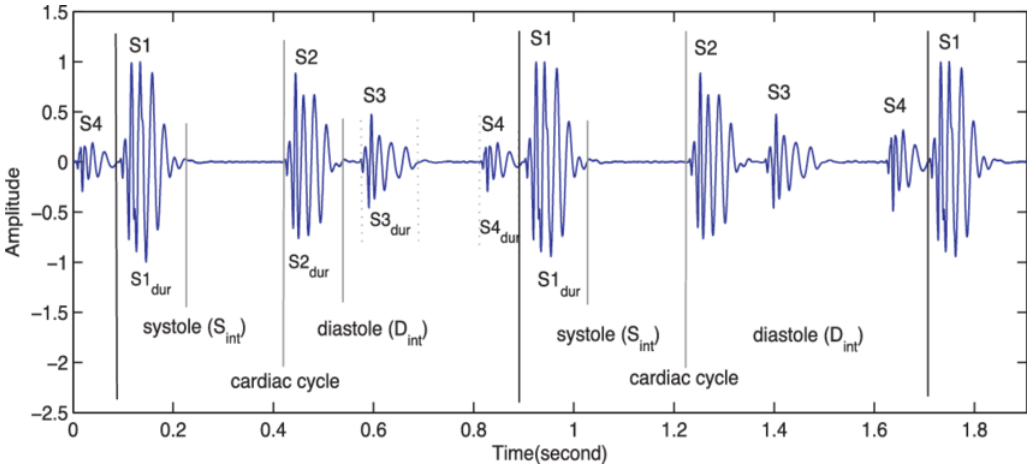


Figura 2.4: Sinais PCG de um sinal cardíaco [4]

2.4 Python

Neste subcapítulo é descrito a linguagem de programação Python, a *framework* utilizada para o desenvolvimento deste projeto e algumas das bibliotecas mais relevantes.

Python, apresentado na Figura 2.5, é uma linguagem de programação das mais utilizadas do mundo, criada por Guido van Rossumm e lançada em 1991. Orientada ao objeto e de alto nível, podendo ser utilizada no campo da programação numérica, inteligência artificial, processamento de imagem, entre outros. Tem uma sintaxe similar à língua Inglesa, com uma linguagem interpretativa significando que o código pode ser executado à medida que é escrito, ou seja, é possível executar um protótipo muito rapidamente. A linguagem tem grande suporte de bibliotecas, integrações e recursos para os desenvolvimentos de projetos. É de fácil utilização e desenvolvimento, dá suporte a multiplataforma (Windows, Mac, Linux, Raspberry Py, etc.), com um enorme número de *frameworks* para facilitar o desenvolvimento.[17][18][19]

Figura 2.5: Python [5]

2.4.1 Kivy

Kivy, apresentado na Figura 2.6, é uma *framkework* de Python para um desenvolvimento de aplicações sendo elas *desktop* ou móveis, com licença do MIT (Instituto de Tecnologia de Massachusetts) que faz com que possa ser utilizada sem custos e comercializar o *software*. Foi lançada no inicio de 2011, permite o desenvolvimento em Windows, Mac, Linux e Raspberry Pi e a aplicação desenvolvida pode ser executada e implementada em qualquer sistema operativo de computadores *desktop* (macOS, Linux ou Windows), dispositivos iOS (iPad e iPhone), dispositivos Android (*tablets* e *smartphones*) e qualquer outro tipo de dispositivo que suporte TUIO (Tangible User Interface Objects). Fornece suporte de de eventos *multitouch*, eventos de teclado e rato. Os gráficos são construídos utilizando OpenGL ES2, então suporta a aceleração dos gráficos através do*Graphics Processing Unit* (GPU). Na criação de uma aplicação com Kivy está-se a criar uma *Natural User Interface* (NUI), o utilizador pode facilmente utilizar o *software* com poucas instruções. Todos os controlos e *widgets* são desenhos personalizados o que faz com que sejam iguais e estejam dispostos de maneira igual em todas as plataformas.[20][21]

Figura 2.6: Kivy [6]

2.4.2 Linguagem Kivy (Kv)

A linguagem Kv, também denominada de kvlang, permite uma separação lógica entre a aplicação e a interface do utilizador e a criação de *widges* atribuindo-

lhe funções, propriedades e interações entre si de modo a facilitar a implementação e mudanças na *User Interface* (UI) muito rapidamente.[22]

2.4.3 Criação de builds da aplicação

Para a criação de uma *build* para Windows só pode ser feito através de uma máquina que tenha o sistema operativo (OS) Windows, em que a versão pode ser 32 ou 64 bits dependendo da versão de Python instalada e requer o auxilio do PyInstaller.[21][23]

Para criação de uma *build* para macOS pode também ser utilizado o PyInstaller e só pode ser feito por uma máquina que contenha o macOS. [21][24]

O PyInstaller agrupa todos os ficheiros e dependências do projeto num único executável, esta *build* criada pode ser distribuído e comercializado. Os novos utilizadores não precisam de ter um interpretador de Python ou qualquer módulo instalado para possam instalar o projeto.[25]

A criação da *build* de instalação pode ser feito através da seguinte linha do comandos:

```
$ pyinstaller [Nome-do-ficheiro-principal.py] -w --onefile
```

O argumento -w diz ao PyInstaller que a aplicação é uma aplicação em janela e o argumento –onefile significa que vai ser criado um único ficheiro executável.

Para a a criação de uma *build* de instalação para Android deve ser feito através de uma máquina com um OS Linux ou macOS com a contribuição do buildozer. Buildozer é uma ferrramenta que automatiza o processo de construção, prepara todos os pré-requisitos para a criação do executável de python para android, incluindo o Android SDK (Software Development Kits) e NDK (Native Development Kits), por fim cria um APK (Android Package) que pode ser enviado automaticamente para o dispositivo Android e executado. [21][26] [27]

Para criação do APK de instalação para Android deve ser feito através dos seguintes etapas:

Criação na pasta do projeto um ficheiro buildozer.spec para configurar o APK:

```
$ buildozer init
```

Depois de criado o ficheiro buildozer.spec, pode-se alterar o ficheiro de forma personalizar a aplicação, definir domínios, entre outros. De seguida, tem que se instalar as dependências do Buildozer dependendo do OS que a máquina tem instalado (macOS ou Linux). [21]

Por fim, pode-se iniciar a compilação para a criação do APK pelo seguinte comando:

```
$ buildozer [target] debug
```

O argumento target indica para qual sistema será feita a *build* e o arguento debug indica que a aplicação será feita do modo *debug* (prevenir que o programa seja corrido com erros de *software*). [21][27]

Para criar uma *build* para iOS tem que se primeiro instalar algumas dependências e pré-requisitos e só pode ser criada através de uma máquina que contenha o OS macOS, depois de ter todos os pré-requisitos instalados é necessário compilar a distribuição pelos seguintes comandos:

```
$ git clone git://github.com/kivy/kivy-ios
$ cd kivy-ios
$ ./toolchain.py build python3 kivy
```

Depois de compilado com sucesso, pode-se criar um projeto Xcode. Para a criação do projeto Xcode é necessário ter um ficheiro de entrada para a aplicação com nome de main.py, por fim pode-se utilizar o seguinte comando para criar o projeto Xcode utilizando o *script* toolchain:

```
$ ./toolchain.py create <title> <appdirectory>
```

O argumento title não pode ter espaços ou caracteres especiais, no argumento appdirectory deve-se indicar o *path* do diretório onde se encontra o projeto. [21][28]

2.4.4 Kivy Launcher

Kivy Launcher é uma aplicação Android que executa qualquer aplicação Kivy armazenado na memória do *smartphone*, pode ser instalado através da Google Play Store. Esta aplicação permite fazer testes de funcionamento da aplicação Android, sem necessitar de gerar uma *build* da aplicação. [26]

2.4.5 PyAudio

PyAudio é uma biblioteca I/O de áudio, multi-plataforma, que fornece ligações Python para a PortAudio. Esta biblioteca pode ser utilizada para gravação e reprodução áudio em qualquer plataforma.

O PortAudio é uma biblioteca de reprodução e gravação de áudio, podendo ser utilizada em diferentes plataformas e OS.

Para operar o PyAudio é necessário inicializar utilizando pyaudio.PyAudio(), este comando configura a PortAudio do sistema.
Para gravação ou reprodução de áudio tem que se se iniciar uma *stream* com as definições e parâmetros desejados, é utilizado pyaudio.PyAudio.open(<definições e parametrizações desejadas>) com isto é configurada uma *stream* para gravação ou reprodução de áudio.

Para parar a gravação ou parar a reprodução áudio utiliza-se pyaudio.Stream.stop_stream() e para fechar a *stream* aplica-se o comando pyaudio.Stream.close().
Por fim, é necessário fechar a sessão de PortAudio, pode ser feito através do comando pyaudio.PyAudio.terminate(). [29]

2.4.6 NumPy

NumPy é uma biblioteca de Python que fornece um grande número de funções e operações para sejam realizados os cálculos numéricos que podem ser utilizados em tarefas como modelos de *Machine Learning*, processamento de imagem e computação gráfica, tarefas matemáticas, entre outros. [30]

2.4.7 Socket

Socket permite que programas enviem e recebam informação entre eles, é uma comunicação bidirecional (num único nó) entre dois programas em execução dentro de uma rede, podendo ser configurados como servidor ou ser configurado como cliente e conectar a outras aplicações.

Os sockets podem ser de dois tipos, SOCK_DGRAM, transporte usualmente associado com o *User Datagram Protocol* (UDP), que não fornece uma entrega confiável de mensagens, frequentemente utilizadas quando a ordem da mensagem não é muito importante, como o envio da mesma informação para múltiplos clientes. Podem também ser do tipo SOCK_STREAM, transporte habitualmente associado ao Transmission Control Protocol (TCP), proporcionando uma entrega confiável e ordenada de bytes entre dois utilizadores, com controlo, sendo geralmente utilizada para implementar em aplicações que transfiram uma grande quantidade de informação entre elas. [31] [32]

A Figura 2.7 apresenta um exemplo de uma comunicação sob o protocolo TCP com uma conexão *full-duplex* continua entre o cliente e o servidor.

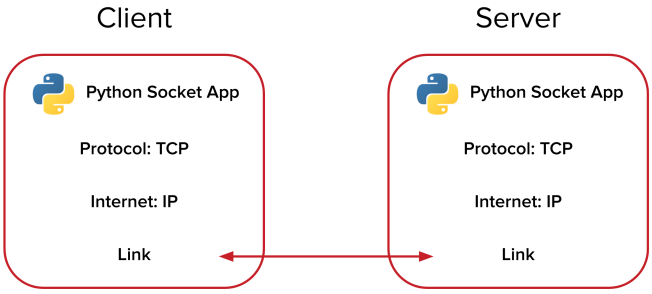


Figura 2.7: Exemplo de uma ligação socket em Python[7]

2.4.8 SciPy

SciPy é um conjunto de ferramentas científicas e numéricas *open source* para Python, com suporte para funções especiais, integrações, otimizações, processamento de sinais e imagens, entre outros. Tem como base o NumPy, permitindo assim manipular e visualizar uma grande quantidade de dados com comandos de alto nível. [33] [34]

2.4.9 Filtro Butterworth

Um filtro Butterworth é desenvolvido de forma a obter uma resposta em frequência muito plana (não possuir *ripple*, ou ondulações) na banda passante e é aproximadamente zero nas bandas rejeitadas.

Para criação de um filtro Butterworth passa-banda é necessário combinar dois filtros, um passa-baixo e outro passa-alto. [35]

Este tipo de filtro é definido pela sua ordem, n, e a sua frequência de corte, w_c e a sua resposta é dada pela Equação 2.1.

$$|H(jw)| = \frac{1}{\sqrt{1 + (\frac{w}{w_c})^{2n}}} \tag{2.1}$$

Na Figura 2.8 apresenta a resposta do filtro de diferentes ordens (n = 3, n = 5 e n = 7) para uma frequência de corte w_c.[8]

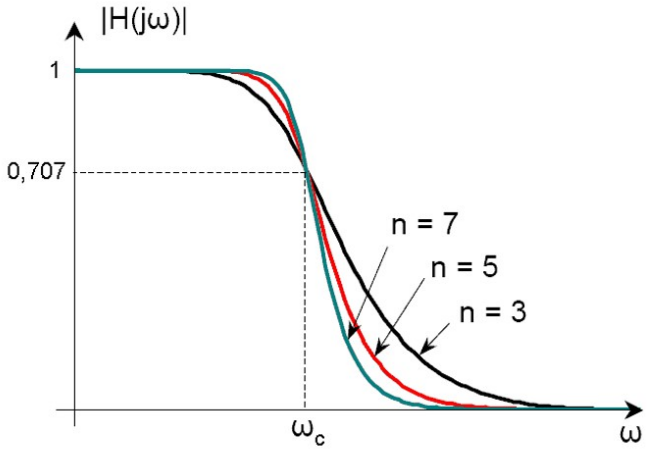


Figura 2.8: Resposta do filtro Butterworth[8]

2.4.10 Teorema de amostragem de Nyquist

O teorema de amostragem de Nyquist clarifica a relação entre a taxa de amostragem e a frequência do sinal. A taxa de amostragem deve ser maior do que o dobro da componente de maior frequência de análise no sinal medido. Essa frequência de amostragem mínima é designada por frequência de Nyquist. [36]

2.4.11 Filtro filtfilt

Filtro filtfilt é um filtro linear que pertence biblioteca SciPy, que corresponde a um filtro de fase zero, ou seja, não altera a relação de fase entre as várias componentes frequênciais do sinal durante o processo de filtragem. Processa os dados de entrada na direção direta, de seguida ele inverte a sequência filtrada e executa-a de volta através do filtro. A sequência final tem distorção de fase precisamente zero à custa da aplicação do mesmo filtro duas vezes.

Este filtro minimiza os transientes de inicialização e de término ao combinar as condições iniciais. Funcionando para entradas reais ou complexas. [37]

2.5 Fourier Transform (FT)

A FT é uma ferramenta que transforma uma forma de onda, pode ser função ou sinal, numa representação alternativa formada por senos e cossenos e faz a transição entre o domínio do tempo para o domínio da frequência. Para uma função g(t), a sua FT é definida pela Equação 2.2.

$$TF: \quad \mathscr{F}\{g(t)\} = G(f) = \int_{-\infty}^{\infty} g(t)e^{-2\pi ift}\,dt \tag{2.2}$$

A função resultante G(f) é o espetro da função g(t). A função original g(t) pode ser obtida através da inverse Fourier transform (IFT) da função G(f), definida pela Equação 2.3.

$$IFT: \quad \mathscr{F}^{-1}\{G(f)\} = \int_{-\infty}^{\infty} G(f)e^{-2\pi ift}\,df = g(t) \tag{2.3}$$

As funções g(t) e G(f) formam um par de Fourier (PF), podendo esse par ser representado pela Equação 2.4. [38]

$$PF: \quad g(t) \xleftarrow{\mathscr{F}^{-1}} \xrightarrow{\mathscr{F}} G(f) \tag{2.4}$$

2.5.1 Fast Fourier Transform (FFT)

Fast Fourier Transform (Transformada rápida de Fourier) é um algoritmo de computação rápida para a Discrete Fourier Transform (DFT), tem como funcionalidade pegar num vetor de amostras de forma de onda no domínio dos tempos, processando-o para produzir um vetor de amostras do espetro no domínio das frequências. Uma função FFT(f) é de ordem $f(nlog(n))$, designado de escala rápida, quase linear em n, enquanto que uma função DFT (f1) é de ordem $f1(n^2)$. É uma ferramenta utilizada para processamento de sinal digital, alterar, filtrar e descodificação digital de áudio, vídeo e imagens. [39]

2.6 Protótipo do sistema miniaturizado de PCG

A placa do protótipo do sistema miniaturizado de PCG foi desenvolvida para ser o mais compacta e otimizada possível e que ofereça o melhor custo-benefício. Nas Figuras 2.9 e 2.10 estão representados o protótipo final do sistema PCG, a placa possuí 7.5 centímetros (cm) de diâmetro. Pela Figura 2.9 é possível verificar uma divisão em três partes organizadas, o condicionamento de sinal, o processamento de sinal e a zona que diz respeito à alimentação do sistema. Com uma interface de programação do microcontrolador sob forma de conector com 6 pinos e uma interface de alimentação e dados sob forma de um conector de 2x4 pinos. O sistema possui uma porta Micro USB para alimentação e carregamento da bateria. Na Figura 2.10 é possível visualizar a parte traseira da placa com uma bateria agregada e um microfone de eletreto [9]. O projeto desenvolvido deverá ser compatível com o protótipo do sistema miniaturizado de PCG, ou seja, deverá conectar-se, possibilitar a receção e a interpretação dos dados sonoros captados.

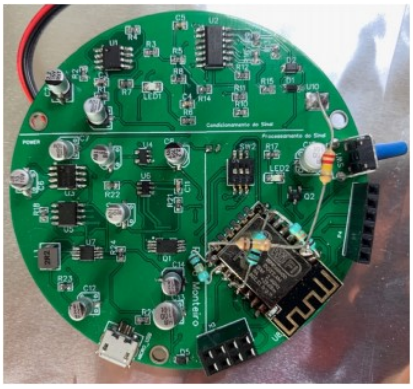


Figura 2.9: Protótipo do PCG miniaturizado (frente) [9]

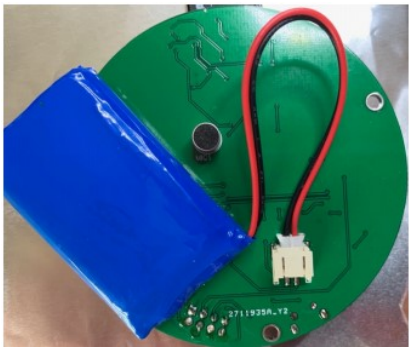


Figura 2.10: Protótipo do PCG miniaturizado (back) [9]

2.7 Soluções de PCG miniaturizados existentes no mercado

Neste subcapítulo serão expostas algumas soluções de PCGs existentes no mercado e uma breve descrição sobre o produto e o seu funcionamento.

2.7.1 Estetoscópio de teleconsulta CMS-VESD

CMS-VESD é um estetoscópio visual com diversas funcionalidades como auscultações, coletar dados de eletrocardiograma (ECG), entre outros. Como se pode verificar pela Figura 2.11 exibe uma forma de onda de ECG, frequência cardíaca, entre outros... Possuí uma saída de auscultadores e volume ajustável. Registo em tempo real e os dados podem ser transferidos para um computador e analisados pelo *software* de análise. [10]

Figura 2.11: Estetoscópio de teleconsulta CMS-VESD [10]

2.7.2 Estetoscópio para cardiologia eKuore Pro

O estetoscópio eKuore Pro permite ouvir remotamente através de auscultadores sem fios ou através de uma aplicação no *smartphone*. Útil para profissionais de saúde monitorizar pacientes crônicos remotamente, evitando visitas desnecessárias ao hospital. Pela Figura 2.12 é possível observar o aparelho de auscultação (à direita) e os dados recebidos e a sua conexão ao *smartphone* com os dados da auscultação. [11]

Figura 2.12: Estetoscópio para cardiologia eKuore Pro [11]

Capítulo 3

Desenvolvimento do sistema de aquisição do PCG

Neste capítulo apresenta-se as diferentes fases de implementação do sistema de aquisição do PCG. Começa-se por apresentar o objetivo e requisitos do projeto desenvolvido exibindo uma arquitetura simplificada do sistema PCG, de seguida apresenta-se diagrama de blocos geral e, por fim, apresenta-se um diagrama de blocos especificamente para cada parte do projeto.

3.1 Objetivo e requisitos do sistema de aquisição do PCG

O sistema de aquisição do PCG teve em atenção alguns requisitos como:

- Fácil utilização e baixo custo;
- Facilidade de ligação e compatibilidade ao sistema de aquisição do PCG externo;
- Compatibilidade com qualquer dispositivo eletrónico com ou sem ligação a uma rede externa;
- Auscultação do coração;
- Análise do som auscultado e verificação do seu funcionamento;
- Visualização do sinal emitido pelo coração.

Na Figura 3.1 é apresentada uma arquitetura simplificada do sistema PCG. O sistema pode funcionar com uma ligação ao sistema miniaturizado PCG anteriormente desenvolvido (apresentado no subcapítulo 2.6), a ligação é feita do sentido aplicação (cliente) - sistema miniaturizado PCG (servidor), depois de estabelecida começa o envio dos dados para a ligação. A aplicação também pode

proceder à recolha dos dados diretamente através de um microfone, em que o sinal sonoro depois de recebido é tratado digitalmente.

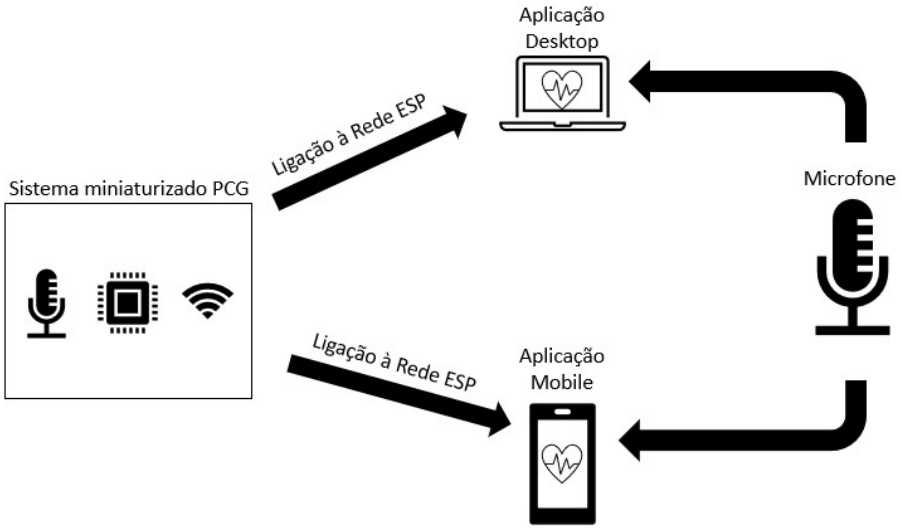


Figura 3.1: Arquitetura simplificada do sistema PCG

3.2 Aplicação de aquisição de processamento

A aplicação desenvolvida está dividida em três partes, designadas de ecrãs (*screens*). A primeira parte corresponde ao ecrã de menu que dá ao utilizador a possibilidade de se conectar a um dispositivo externo ou continuar sem a conexão ao dispositivo. A segunda parte diz respeito ao ecrã de ligação, só é mostrada quando o utilizador escolher a opção de conectar a um dispositivo externo, fica à espera que a ligação esteja estabelecida com o dispositivo externo. Por fim, a terceira parte é referente ao ecrã de análise de sinal.

3.3 Fluxograma da aplicação

Neste subcapítulo é exposto os diversos fluxogramas de funcionamento da aplicação desenvolvida nos seus diversos modos.

3.3.1 Funcionamento da aplicação

Na Figura 3.2 está apresentado o fluxograma simplificado.

Ao iniciar a aplicação são dadas duas opções ao utilizador, se quer conectar a um dispositivo externo ou continuar sem conectar. Se o utilizador escolher a opção de conectar a um dispositivo externo, vai para o ecrã de ligação e tenta estabelecer uma conexão ao dispositivo externo. Se a ligação foi feita sucesso vai para o ecrã final, o ecrã de análise de sinal. Por outro lado, se o utilizador escolher a opção de continuar sem dispositivo externo, a captura do sinal é feita através do microfone do dispositivo que a aplicação esteja a correr, de seguida vai para o ecrã de análise do sinal. No ecrã de análise de sinal é possível voltar ao ecrã inicial ao pressionar o botão "Go back to menu".

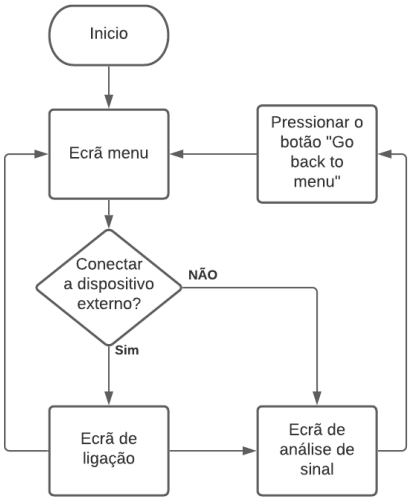


Figura 3.2: Fluxograma simplificado do funcionamento da Aplicação

3.3.2 Funcionamento do ecrã de menu

O ecrã de menu tem como funcionalidades de dar a escolha ao utilizador de se conectar ao dispositivo externo ou caso não tenha, dar a possibilidade de continuar sem a necessidade de ligação. Este ecrã como se pode ver pela Figura 3.2 encaminha a aplicação para o próximo ecrã, o de ligação caso queira conectar ou diretamente para o ecrã de análise de sinal.

3.3.3 Funcionamento do ecrã de ligação

Na Figura 3.3 está apresentado o fluxograma do ecrã de ligação. Se o utilizador escolher a opção de conectar a um dispositivo externo, é iniciada uma *thread* para tratar da ligação em *background*. Se conseguir conectar com sucesso ao dispositivo externo a aplicação avança para o ecrã de análise de sinal. Por outro lado, se não conseguir estabelecer uma conexão, volta a tentar conectar-se

de novo, ao fim de cinco tentativas é apresentado um *pop-up* de erro de ligação e volta ao ecrã menu.

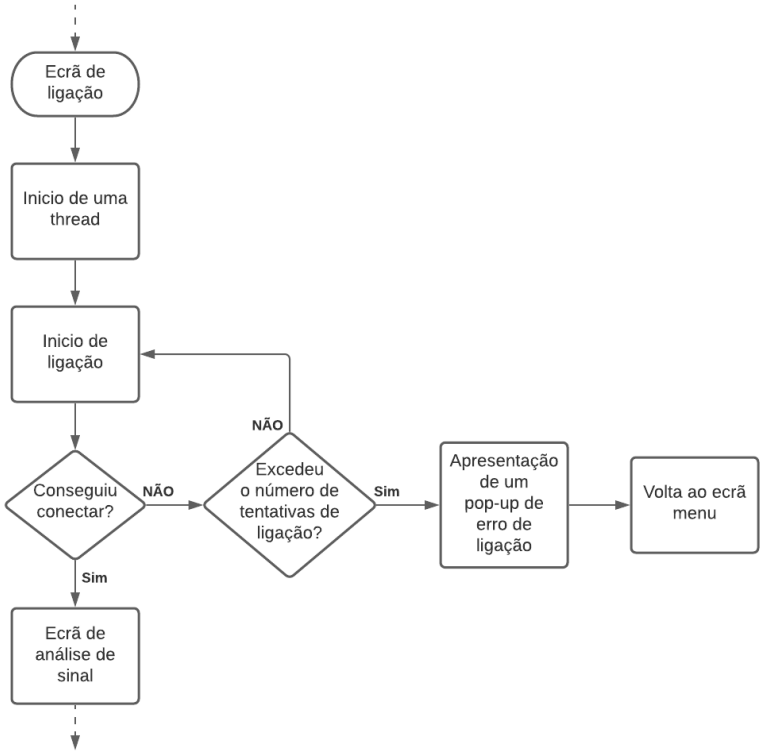


Figura 3.3: Fluxograma do ecrã de ligação

3.3.4 Funcionamento do ecrã de análise de sinal com conexão ao dispositivo externo

Na Figura 3.4 está representando um fluxograma do ecrã de análise de sinal com ligação ao dispositivo externo. Ao estabelecer ligação no ecrã anterior, é iniciada uma *Thread* para receção de informação, essa informação é um sinal filtrado proveniente do dispositivo externo, é calculada a FFT do sinal. Os vetores que armazenam o sinal filtrado e a FFT têm um limite de de 25 * 1024 (1024 representa o tamanho de cada amostra recolhida por *buffer*) e vetor de cache guarda duas vezes esse valor. Esses limites são verificados e se forem atingidos é avaliado o grau de ruído do sinal, se verificar um grau elevado é apresentado um pop-up de erro, o valor de status altera-se de ”OK” para ”NOK” e, por fim, é gravado um áudio com os valores de cache. Se o sinal não apresentar um ruído significativo, os vetores são limpos e os novos valores do sinal filtrado e FFT são guardados e representados graficamente. Se os limites não tiverem sido atingidos

o valor do sinal filtrado e a sua FFT são armazenados nos seus respetivos vetores e representados gráficamente.
Se o utilizador pressionar o botão ”Go back to menu” é fechada a ligação ao dispositivo externo e as variáveis globais são limpas, de seguida volta ao ecrã menu.
Caso o utilizador pressione o botão ”Export sound” é gravado um áudio do tipo .wav com os valores armazenados em cache e exportado para a pasta da aplicação.

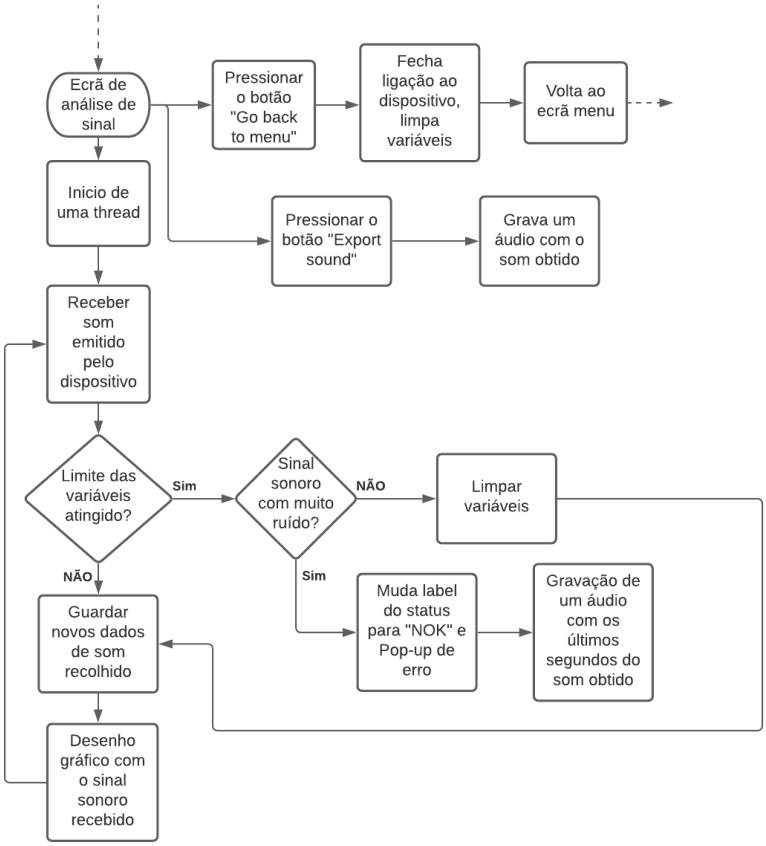


Figura 3.4: Fluxograma do ecrã de análise de sinal com ligação

3.3.5 Funcionamento do ecrã de análise de sinal sem conexão a dispositivo externo

O fluxograma do ecrã de análise de sinal sem ligação ao dispositivo externo está representado na Figura 3.5. No ecrã inicial ao ser escolhida a opção de não querer conectar, abre diretamente o ecrã de análise de sinal. Neste ecrã é iniciada uma *thread* que vai recolher os dados do microfone, depois de recolhidos

é aplicado um filtro ao sinal através de um filtro passa-banda e calcula a sua FFT esses valores são ajustados para estarem entre um intervalo [-1, 1] de amplitude, os valores são armazenados em vetores e representados graficamente. É guardado também o sinal original para um vetor de cache. O vetor do sinal filtrado e o vetor da FFT têm um limite de 25 * 1024 (1024 representa o tamanho de cada amostra recolhida por *buffer*) enquanto que o vetor de cache guarda duas vezes esse valor. É feita uma verificação se o limite dos vetores foi atingido, caso os limites tenham sido atingidos é avaliado o nível de ruído do sinal sonoro e se tiver muito ruído é alterado o valor do status de "OK" para "NOK" ativando um pop-up de erro e é exportado um ficheiro com o valor de cache guardado. Ao fechar o pop-up o valor do status volta a "OK" e volta a receber dados através do microfone. Se não tiver muito ruído os vetores do FFT e do sinal filtrado são limpos, depois são guardados os novos dados adquiridos e são desenhados os gráficos do sinal sonoro e a sua FFT. O vetor de cache é limpo quando atinge o seu limite e guarda os novos valores.
À semelhança do funcionamento do ecrã de análise de sinal com conexão ao dispositivo externo tem um botão "Go back to menu" que ao ser pressionado são limpos os vetores e volta ao ecrã menu. Ao pressionar o botão "Export sound" grava-se um áudio com o som guardado em cache e exportado para a pasta da aplicação.

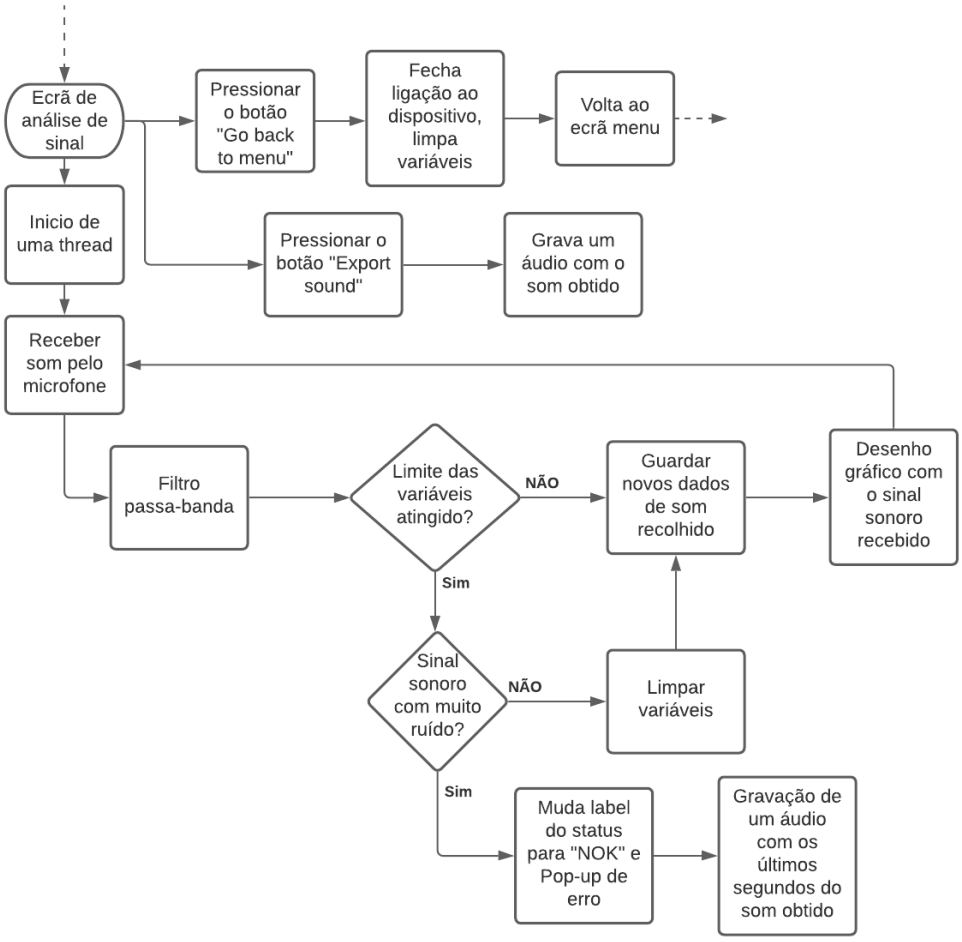


Figura 3.5: Fluxograma do ecrã de análise de sinal sem ligação

3.4 Condicionamento de sinal

O condicionamento de sinal para o dispositivo externo utilizado é feito no próprio dispositivo externo. No condicionamento de sinal da aplicação para quando a aplicação não está conectada a um dispositivo externo é feito através de um filtro passa-banda digital feito após receção do sinal sonoro. Este filtro passa-banda está projetado para permitir a passagem de frequências entre 50 Hz e 200 Hz. Primeiramente é criado um filtro *Butterworth* com a seguinte sintaxe:

```
b, a = scipy.signal.butter(
order,
[lowCut, highCut],
'bandpass',
analog=False)
```

A resposta b, a representam respetivamente o numerador e denominador da função transferência, *order* é a ordem do filtro, de seguida é definido o tipo de filtro que se pretende criar neste caso um filtro passa-banda o ultimo parâmetro diz respeito ao sinal que se pretende filtrar, um sinal digital. Para obtenção os parâmetros lowCut e highCut é necessário calcular a frequência de Nyquist apresentada na Equação 3.1. A frequência de Nyquist corresponde a metade da frequência de amostragem (fs). De seguida, o valor de lowCut é obtido como é mostrado na Equação 3.2 em que fl corresponde à frequência de corte inferior dividida o valor da frequência de Nyquist, por fim, o valor de highCut é obtido segundo a Equação 3.3, fh corresponde à frequência de corte superior divida por a frequência de Nyquist

$$Nyquist: \quad nyq = 0.5 * fs \tag{3.1}$$

$$lowCut: \quad lowCut = \frac{fl}{nyq} \tag{3.2}$$

$$highCut: highCut = \frac{fh}{nyq} \tag{3.3}$$

Após a criação do filtro *Butterworth* é aplicado ao sinal pela função spicy.signal.filtfilt() da seguinte forma:

```
y = scipy.signal.filtfilt(b, a, signal, axis=0)
```

Por fim, o y corresponde ao sinal filtrado na gama de frequências desejado.

3.5 Ligação ao dispositivo externo

A aplicação inicialmente tenta conectar-se ao dispositivo externo, através da função s.connect(HOST_IP, PORT) em que HOST_IP corresponde ao IP do dispositivo e PORT a porta que o dispositivo externo atribuiu para a comunicação. Para receção do conteúdo é necessário definir o tamanho de informação que se irá receber (*buffer*) a mensagem é recebida pela função s.recv(*buffer*) e é necessário definir um tamanho mínimo de mensagem (*HEADER*) que se irá receber, este tamanho mínimo deverá ser o mesmo na aplicação e no dispositivo externo. O socket não recebe nenhuma outra informação até a mensagem estar completa, ao ficar completa a aplicação fica pronta para receber mais informação.

3.6 Obtenção do sinal sonoro através do microfone

Para obter o sinal sonoro foi utilizado uma biblioteca denominada PyAudio, foi iniciado conforme a seguinte sintaxe:

```
stream = p.open(format = FORMAT,
                channels = CHANNELS,
                rate = RATE,
                input = True,
                frames_per_buffer = chunk
               )
```

Formato indica qual o formato da recolha de dados, tendo sido escolhido o formato "paInt16" (16 bit int), o parâmetro *channels* indica quantos canais se pretende a recolha de dados, se tiver mais do que um microfone configurado, a variavél CHANNELS tem o valor de 1. A rate especifica a frequência de captura desejada, é necessário verificar a compatibilidade do dispositivo e dependendo da variedade das frequências procede-se à sua escolha, para o desenvolvimento do projeto foi escolhida a frequência 44100 Hz. O parâmetro input indica se a aplicação vai capturar ou emitir áudio, como se pretende capturar esse parâmetro fica a True. Por fim, frames_per_buffer especifica o número de *frames* por captura, no projeto desenvolvido serão capturadas 1024 *frames* por *buffer*.

Capítulo 4

Validação e resultados

Neste capítulo serão validadas as implementações desenvolvidas e demonstrado o resultado final da aplicação.

4.1 Validação da conexão da aplicação

Para validar a conectividade da aplicação foi criado um servidor que simula o dispositivo externo e envia dados para a aplicação. O servidor é iniciado localmente na porta 8080, do tipo SOCK_STREAM pois é necessário uma entrega confiável, ordenada e a transferência de uma grande quantidade de informação e pode aceitar até cinco ligações. Ao ser iniciado imprime uma mensagem "start server". Ao ser conectado o servidor envia a sua primeira mensagem "Welcome to the server!" e imprime os dados do cliente. Na Figura 4.1 está apresentada a mensagem enviada pelo servidor ao cliente depois de se conectar.

Figura 4.1: Validação da receção da mensagem do servidor

Na Figura 4.2 primeiramente está apresentada a mensagem "start server" indicando que o servidor foi iniciado, seguida da mensagem "Connection from ('192.168.1.8', 62633) has been established!", indicando que a conexão foi efetuada com sucesso e imprime de onde foi feita a conexão.

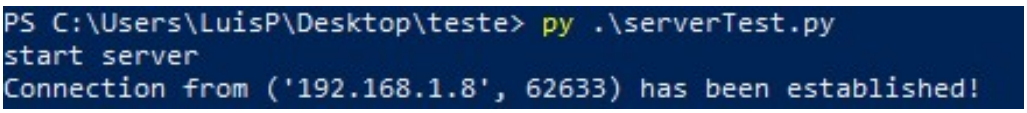


Figura 4.2: Validação da ligação do cliente (aplicação) ao servidor

4.2 Validação do condicionamento de sinal

Para validação do condicionamento de sinal foram feitos testes com diversas frequências à entrada e validar o sinal à saída. O condicionamento está configurado de forma a permitir a passagem de frequências entre 50 Hz até 200 Hz, com um *ripple* de -3 dB (decibéis) e uma atenuação de -20 dB na rejeita-faixa definida de 200 Hz até 400 Hz. Para a escolha da ordem (order) que melhor se adequa para a filtrar o sinal sonoro do projeto para dados amostrados a 44100 Hz, então para fazer essa escolha foi necessário analisar a Figura 4.3 que está representado o comportamento do filtro *Butterworth* implementado nas diferentes ordens.

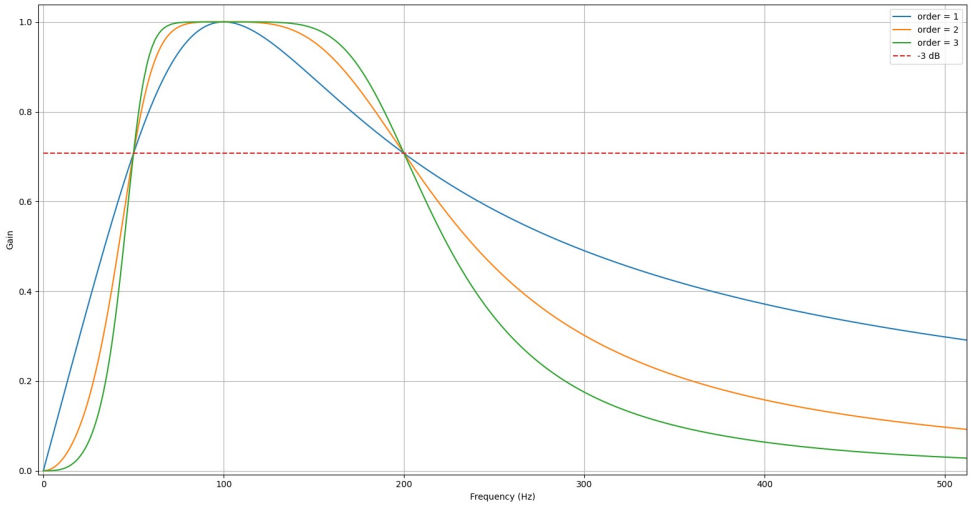


Figura 4.3: Filtro Butterworth

Através da equação 4.1 é possível determinar a ordem adequada para a implementação do filtro *Butterworth*. O parâmetro que Am representa a atenuação máxima na rejeita-faixa (20 dB), W representa a frequência de corte superior (200 Hz) e a Wp representa a frequência de corte inferior (50 Hz).

$$Ordem: \quad N = \frac{\log_{10} 10^{\frac{Am}{10}} - 1}{2 \log_{10} \frac{W}{Wp}} \tag{4.1}$$

$$\Leftrightarrow N = \frac{\log_{10} 10^{\frac{20}{10}} - 1}{2 \log_{10} \frac{200}{50}} \approx 2 \tag{4.2}$$

Pode-se concluir que a ordem mais adequada com uma resposta em frequência mais plana possível para implementação do filtro será a 2.

De forma a testar todo o condicionamento de sinal foram simulados diversos sinais sinusoidais de diferentes frequências, através da função cosseno. A função cosseno, f(t), está representada pela Equação 4.3, A representa a amplitude do sinal e f0 indica a frequência.

$$f(t) = A * cos(2 * \pi * f0 * t) \tag{4.3}$$

Foram testadas com o valor de amplitude de 0.2 as seguintes frequências: 2000 Hz, 1000 Hz e 150 Hz. A primeira frequência testada foi de 2 kHz cujo resultado é apresentado na Figura 4.4, a azul está representado o sinal de entrada e a cor-de-laranja está representado o sinal de saída filtrado.

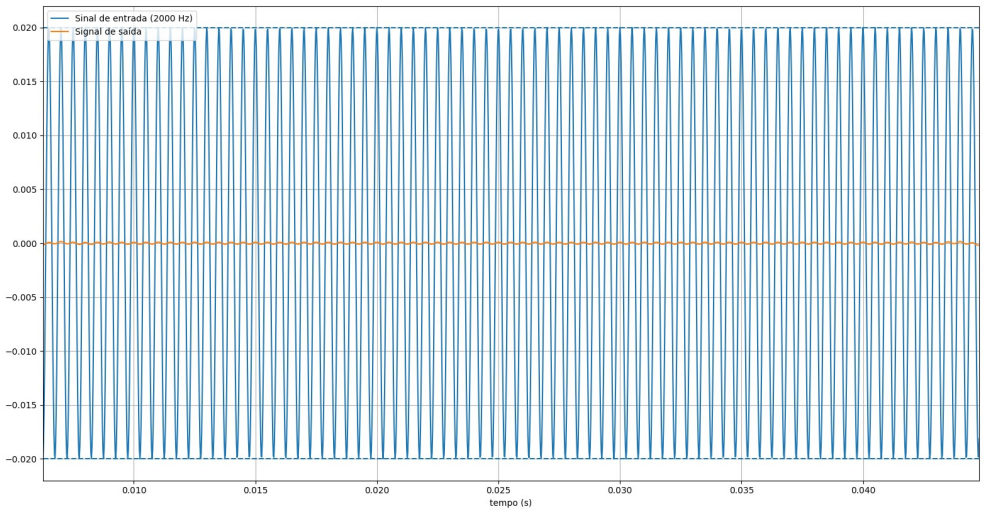


Figura 4.4: Sinal de entrada e saída do filtro a 2 kHz

Pode-se confirmar que corta todas as frequências de elevado ruído e permite a passagem de todas as frequências inferiores a 200 Hz e superiores a 50 Hz. Na Figura 4.5 está representada a segunda frequência testada, 1 kHz.

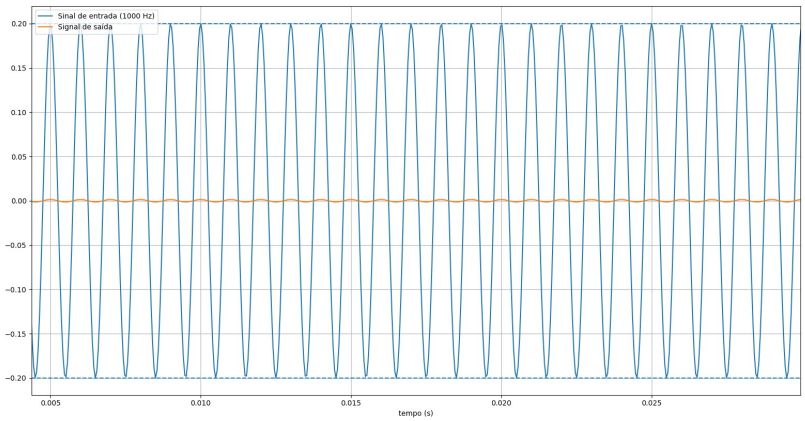


Figura 4.5: Sinal de entrada e saída do filtro a 1 kHz

Confirma-se que tem um comportamento semelhante ao filtrar a onda de 2 kHz, permite a passagem de todas as frequências na banda passante entre 50 Hz e 200 Hz. Na Figura 4.6 está representado um sinal com uma frequência de 150 Hz.

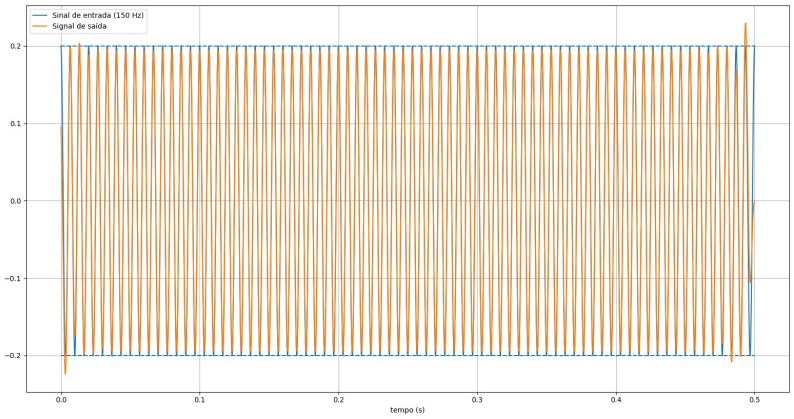


Figura 4.6: Sinal de entrada e saída do filtro a 150 Hz

Os dois sinais encontram-se quase totalmente sobrepostos. Na Figura 4.7 está representado o teste realizado para o condicionamento de sinal de uma onda de 100 Hz.

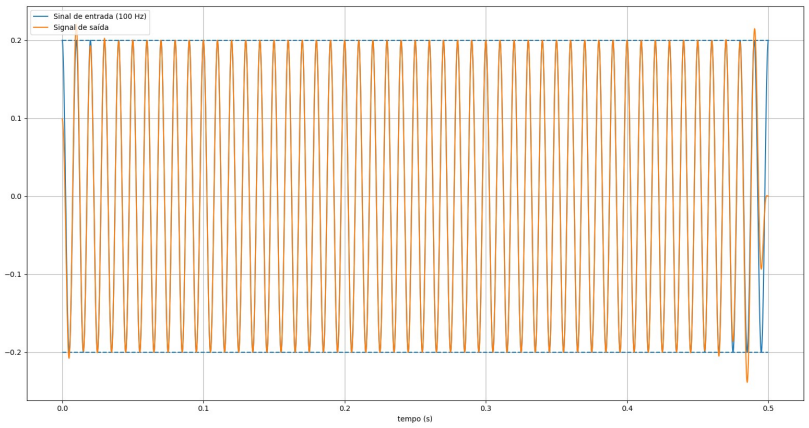


Figura 4.7: Sinal de entrada e saída do filtro a 100 Hz

4.3 Validação da FFT

Para validação da FFT foram desenvolvidos testes com a soma de dois sinais de frequências e amplitudes diferentes. O primeiro sinal (f1(t)) tem uma frequência de 300 Hz e uma amplitude de 2 está representado na Equação 4.4:

$$f1(t): \quad f1(t) = 2.0 * cos(2 * \pi * 300.0 * t) \tag{4.4}$$

O segundo sinal (f2(t)) tem uma frequência de 500 Hz e uma aplitude de 1, está representado na Equação 4.5:

$$f2(t): \quad f2(t) = 1.0 * cos(2 * \pi * 500.0 * t) \tag{4.5}$$

O sinal f(t) provém da soma dos dois sinais, f1(t) e f(2), e a sua soma é apresentada na Figura 4.8, o eixo das ordenadas representa a amplitude do sinal e das abcissas representa o tempo.

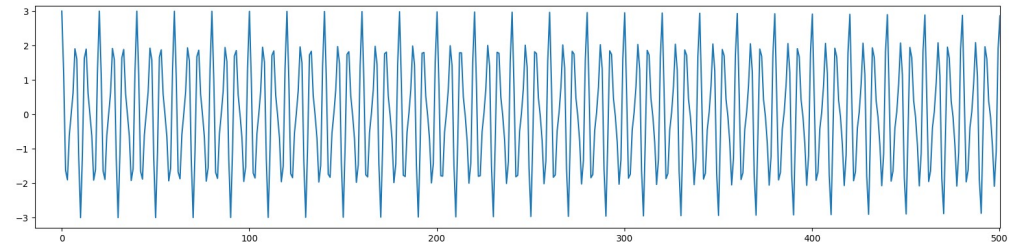


Figura 4.8: Sinal f(t)

O cálculo da FFT é feito em Python através da função fft da biblioteca scipy.fftpack, são retirados os valores espalhados e menores do que zero. De seguida, é necessário tirar o valor absoluto da transformada dividido pelo número de pontos e multiplicar-se por dois para eliminar os valores negativos na amplitude. A Figura 4.9 apresenta a FFT do sinal f(t).

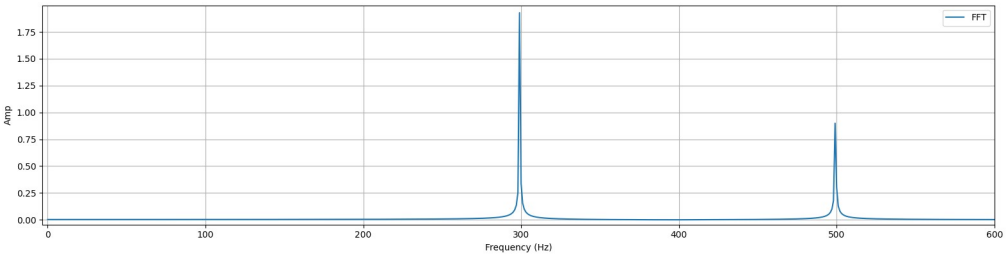


Figura 4.9: Sinal FFT

Pode-se confirmar a resposta da FFT, pois na Figura 4.9 estão representados as duas frequências, de 300 Hz e 500 Hz e as suas respetivas amplitudes de 2 e 1.

Para implementação da FFT no projeto desenvolvido é necessário verificar a sua validade no sinal filtrado entre 50 Hz e 200 Hz, então para confirmar foi utilizada o mesmo sinal f(t) proveniente da soma dos sinais apresentados nas Equações 4.6 e 4.7 com as frequências 150 Hz e 300 Hz, e amplitudes de 2.0 e 3.0, respetivamente.

$$s1(t): \quad f1(t) = 2.0 * cos(2 * \pi * 150.0 * t) \tag{4.6}$$

$$s2(t): \quad f2(t) = 3.0 * cos(2 * \pi * 300.0 * t) \tag{4.7}$$

A soma dos sinais s1(t) e s2(t) sinais dá origem a função s(t) representada na Figura 4.10, o eixo das ordenadas representa a amplitude do sinal e das abcissas

representa o tempo. É aplicado a este sinal o filtro passa-banda desenvolvido e aplicado o calculo da FFT.

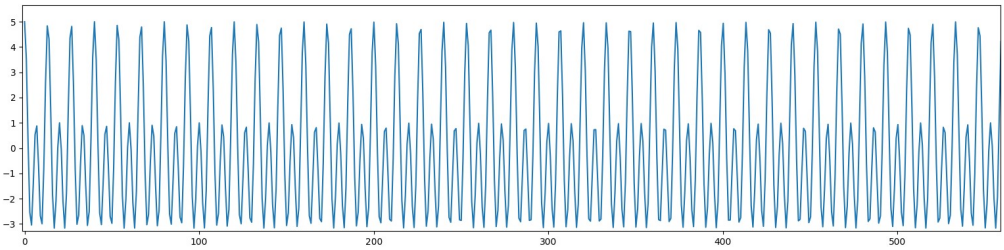


Figura 4.10: sinal s(t)

Este sinal s(t) é filtrado através do filtro passa-banda implementado e calculado o FFT do sinal filtrado, o FFT encontra-se representado Na Figura 4.11.

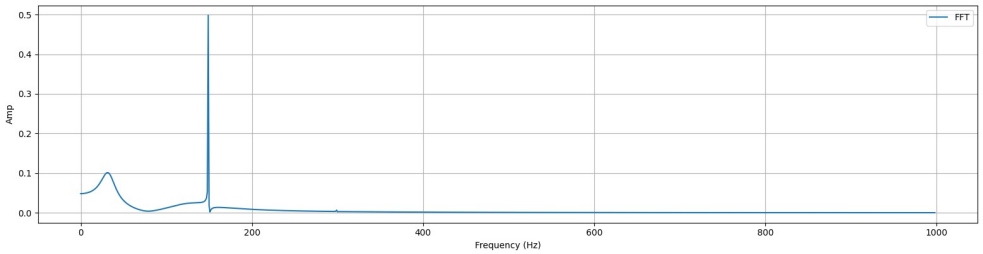


Figura 4.11: sinal FFT filtrado

Confirma-se que o sinal s2(t) é totalmente filtrado e é só apresentado o sinal que se encontra gama de frequências de aceitação do filtro passa-banda sendo só apresentado o sinal s1(t).

4.4 Validação da captura do sinal sonoro

Para validar a captura do sinal sonoro, foi efetuada uma pequena implementação com o objetivo de perceber se o sinal é capturado com sucesso, para confirmar foi feito uma gravação através da biblioteca de PyAudio durante 5 segundos e exportar o conteúdo gravado para um ficheiro .wav e ao reproduzir o ficheiro criado entendeu-se a gravação foi feita com sucesso. Ao iniciar a gravação é apresentado uma mensagem de aviso de começo de gravação ”* recording” e o fim de gravação ”* done recording”.

Na Figura 4.12 está representados o ficheiro .wav criado com o nome ”voice” e na Figura 4.13 confirmação do ficheiro se encontrar com 5 segundos.

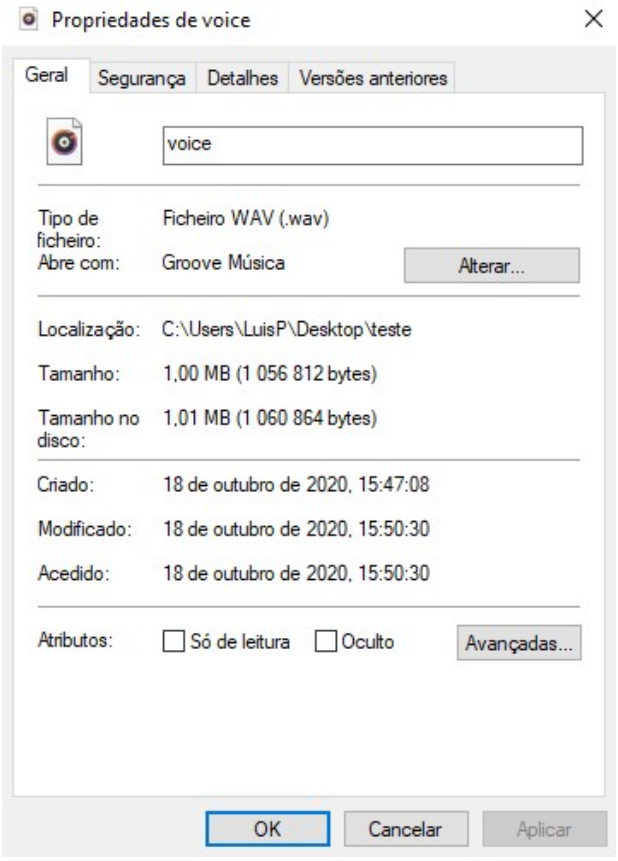


Figura 4.12: Ficheiro .wav criado com nome de ”voice”

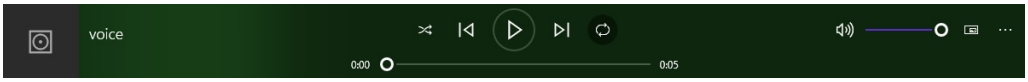


Figura 4.13: Confirmação do sinal gravado com 5 segundos

Com o sinal foi capturado com sucesso, foi implementado o sinal recolhido através do PyAudio e representado graficamente em tempo real. Na Figura 4.14[1] está representado, a azul, a captura do sinal sonoro.

[1]A Figura representa uma implementação de teste, não é a implementação final.

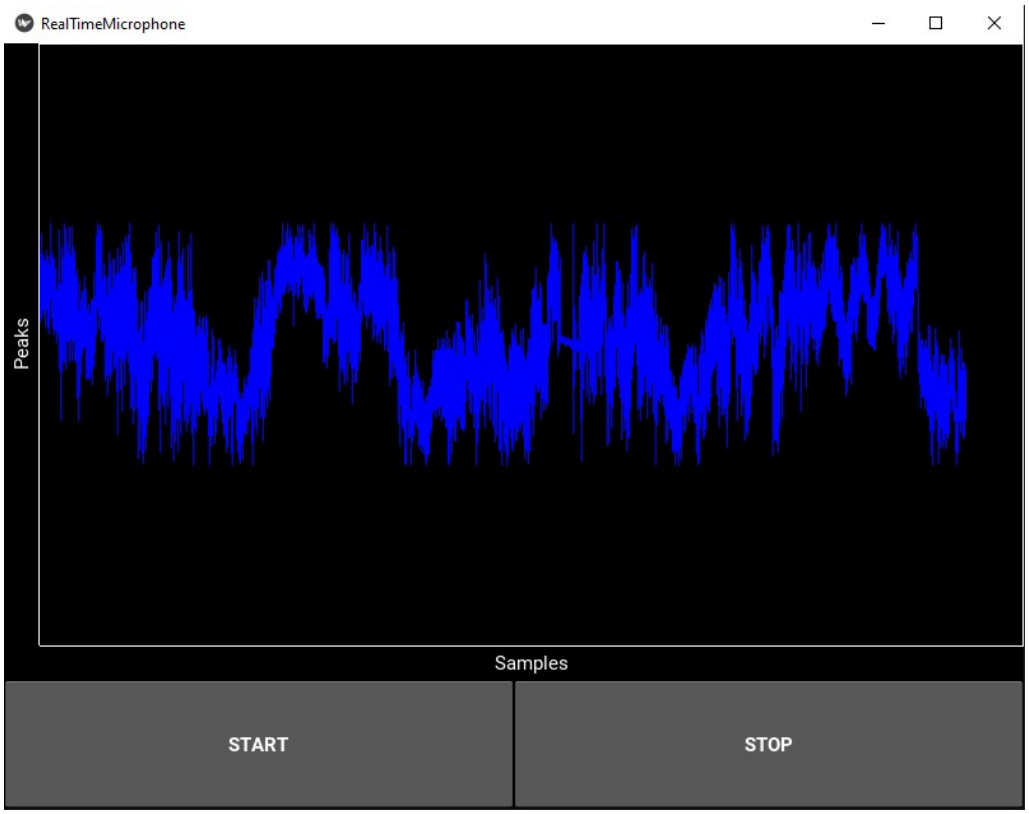


Figura 4.14: Representação do sinal sonoro capturado através do microfone.

Ao pressionar o botão "START" é iniciado a captura do sinal através do microfone, o botão "STOP" pára a gravação e o gráfico fica estático, pode confirmar-se que é possível obter som através do microfone e representar graficamente o seu sinal.

4.4.1 Validação do condicionamento de sinal aplicado ao sinal sonoro

Na implementação da captura do sinal para implementação no projeto é necessário condicionar o sinal, à semelhança dos processos de captura de sinais também se fez dois testes para validação do condicionamento de sinal aplicado ao sinal sonoro capturado.

É aplicado o filtro desenvolvido à gravação produzida através da biblioteca PyAudio durante 5 segundos, posteriormente são realizados testes para perceber qual é o melhor valor de amplificar o sinal sem distorcer ou adicionar ruído ao seu conteúdo e, por fim, é exportado para um ficheiro .wav. Os valores testados de amplificação (32767 é o mínimo para arquivos de 16 bits) foram 32767, 65534 e 131068. Na Figura 4.15 representa a criação dos ficheiros para cada valor de amplificação e na Figura 4.16 a confirmação da sua duração.

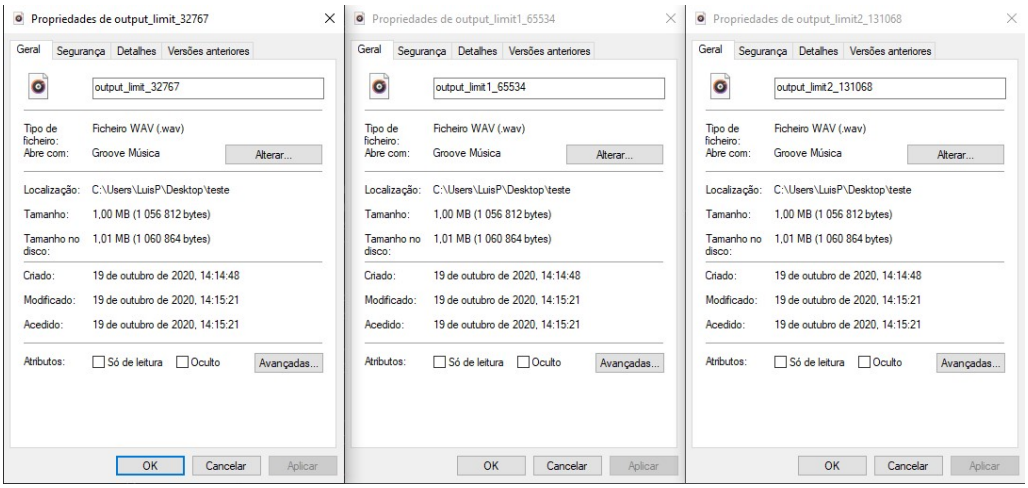


Figura 4.15: Ficheiros .wav com condicionamento de sinal

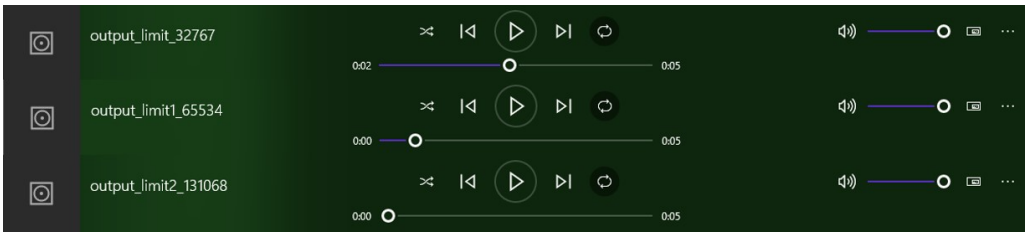


Figura 4.16: Confirmação da duração dos ficheiros .wav com condicionamento de sinal

Após uma análise rigorosa ao ficheiro .wav, confirma-se que o sinal encontra-se filtrado com sucesso. O valor de amplificação escolhido foi o de 65534 pois o valor 131068 encontra-se distorcido e com ruído e o valor de 32767 amplifica o sinal mas não é suficiente para ter um bom nível de amplificação.

Foi necessário proceder à mesma implementação do condicionamento do sinal para representar graficamente o sinal recolhido através do PyAudio. Na Figura 4.17[2] está representado, a azul o sinal original e a vermelho o sinal de saída do condicionamento de sinal.

[2]A Figura representa uma implementação de teste, não é a implementação final.

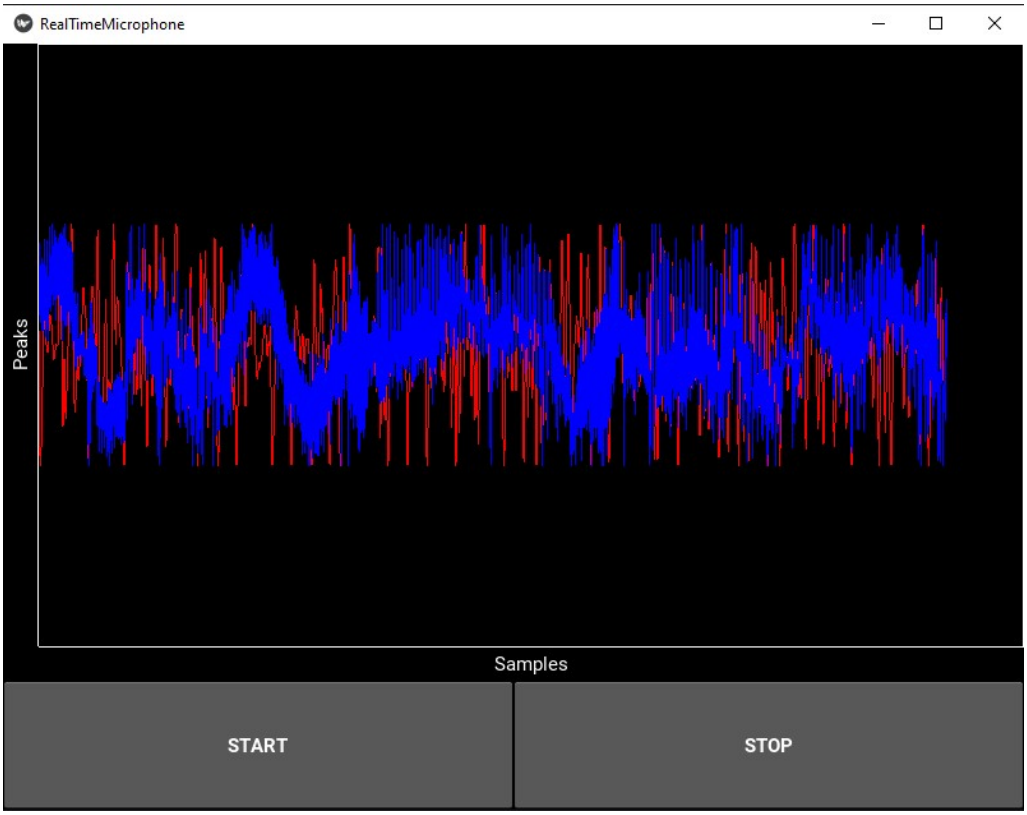


Figura 4.17: Representação do sinal sonoro capturado com condicionamento de sinal

Os sinais estão redimensionados para estarem apresentados entre -1 e 1. É possível verificar que o sinal a azul tem muito ruído enquanto que o sinal representado a vermelho tem muito menos ruído e as suas linhas estão mais visíveis.

4.5 Protótipo da aplicação

Neste subcapítulo será apresentado o projeto final desenvolvido, desenvolvido com as tecnologias desenvolvidas e validadas anteriormente.

4.5.1 Ecrã de menu

Ao iniciar a aplicação o primeiro ecrã é alusivo ao ecrã de menu, apresentado na Figura 4.18. Este ecrã dá ao utilizador a possibilidade de escolher se pretende conectar a um dispositivo externo através do botão ”Connect to external device”, ou então continuar sem necessidade de conexão ao pressionar o botão ”Continue without external device”. Apresenta o título da aplicação ”Phonocardiogram” com uma representação de o que é um PCG, uma imagem no canto superior

direito alusivo ao Instituto Superior de Engenharia do Porto e no canto inferior direito informação do criador da aplicação.

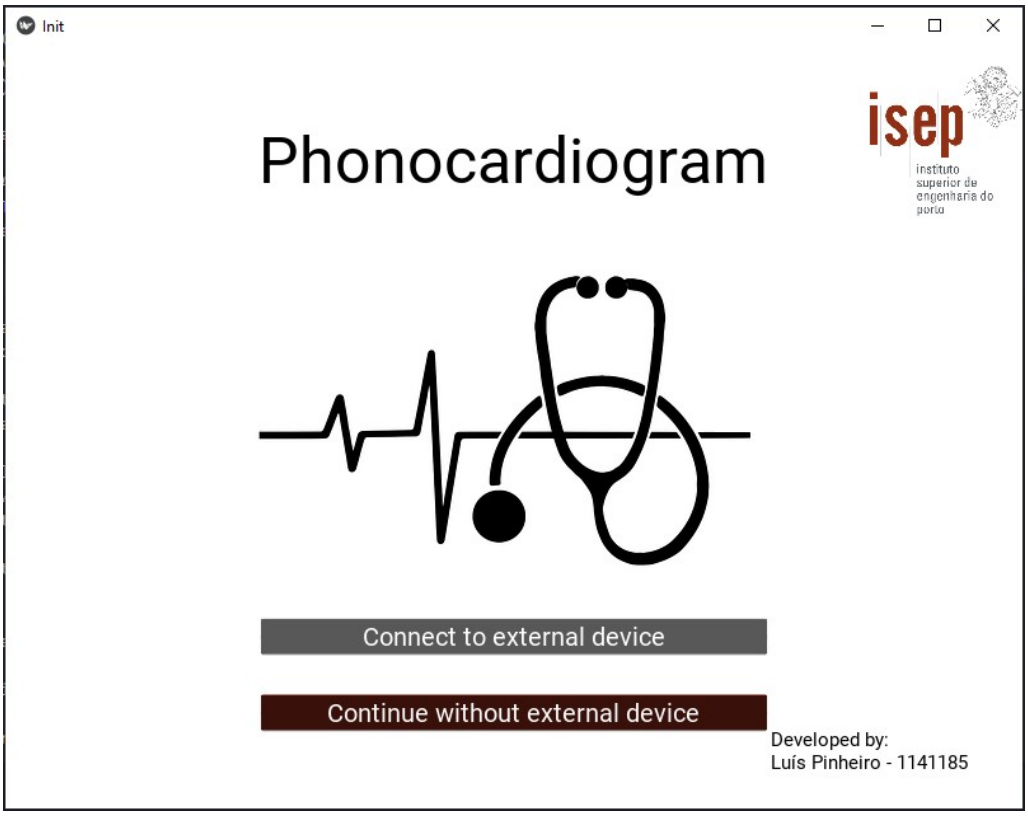


Figura 4.18: Representação do ecrã de menu

Ao pressionar o botão ”Connect to external device” este direciona a aplicação para o ecrã de ligação para estabelecer ligação ao dispositivo externo, enquanto ao pressionar o botão ”Continue without external device” a aplicação é redirecionada para o ecrã de análise de sinal.

4.5.2 Ecrã de ligação

O ecrã ligação só apresentado quando o utilizador escolhe a opção ”Connect to external device” no ecrã de menu. Este ecrã tenta conectar-se ao dispositivo externo. Na Figura 4.19, está apresentado o ecrã de ligação, este ecrã tem um *spinner* no centro enquanto se estabelece a ligação, com um texto ”Waiting for connection”. Ao conectar-se com sucesso avança para o ecrã de análise de sinal, caso contrário volta para o ecrã de menu com o *pop-up* ilustrado na Figura 4.20, ao pressionar o *pop-up* este fecha.

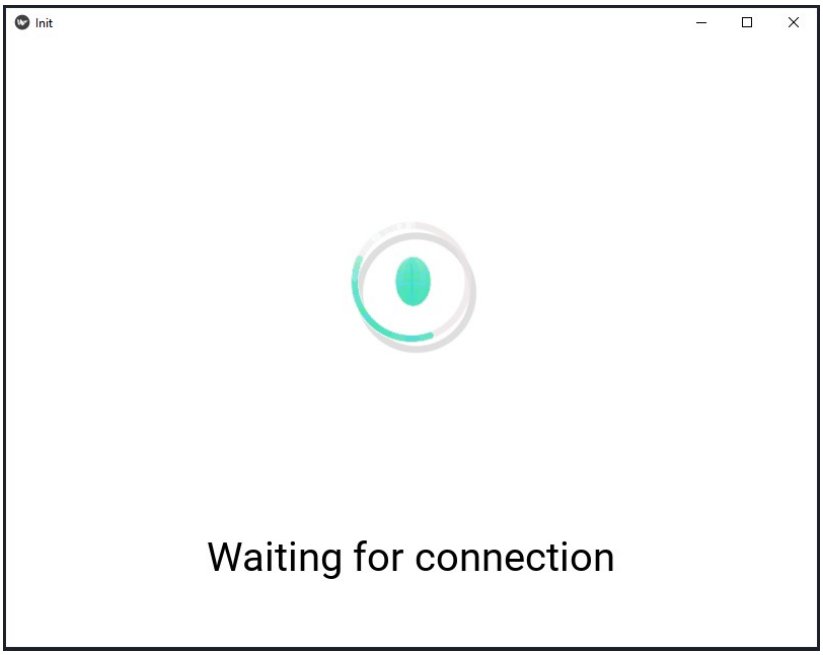


Figura 4.19: Representação do ecrã de ligação

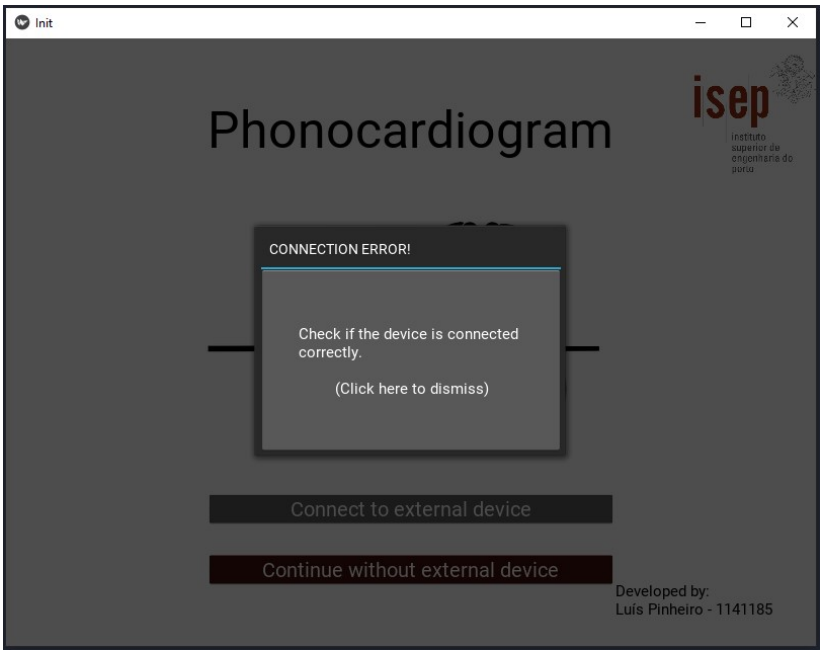


Figura 4.20: Representação do *pop-up* de erro de conexão

4.5.3 Ecrã de análise de sinal

O ecrã de análise de sinal pode analisar do sinal recebido pelo dispositivo externo, ou analisar os dados recolhidos pelo microfone do dispositivo que a aplicação se encontra. O sinal recolhido pelo dispositivo externo não necessita de condicionamento de sinal, pois o sinal é condicionado no próprio dispositivo. O sinal recolhido pelo microfone do dispositivo que a aplicação se encontra é condicionado na própria aplicação. Os FFTs de ambos os sinais são calculados na aplicação.

Este ecrã está apresentado na Figura 4.21. Este ecrã mostra um gráfico confinado em amplitude entre -1 e 1, o status que mostra como está a clareza do sinal recebido, se tem muito ruído. Se este se encontrar "OK" o sinal recebido não tem níveis de ruído elevado. Encontra-se um botão para exportar o sinal sonoro para o dispositivo sob forma de um ficheiro .wav, e um botão "Go back to menu" que volta ao ecrã de menu.

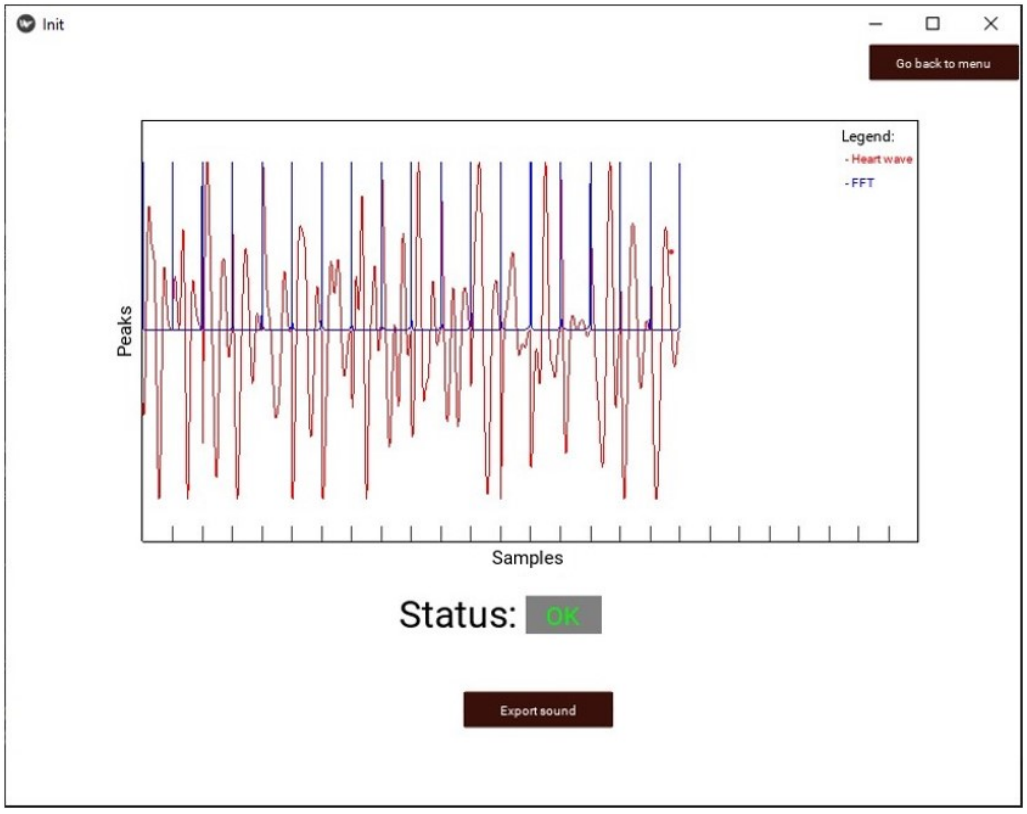


Figura 4.21: Representação Ecrã de análise de sinal

No gráfico deste ecrã podem ser analisados o sinal sonoro (a vermelho) e o FFT (a azul), podendo então, de forma fácil, analisar o comportamento do coração e

verificar a existência dos seus componentes S1 e S2 através da visualização das ondas. Na Figura 4.22 estão assinalados onde se encontra o S1 e de seguida o S2.

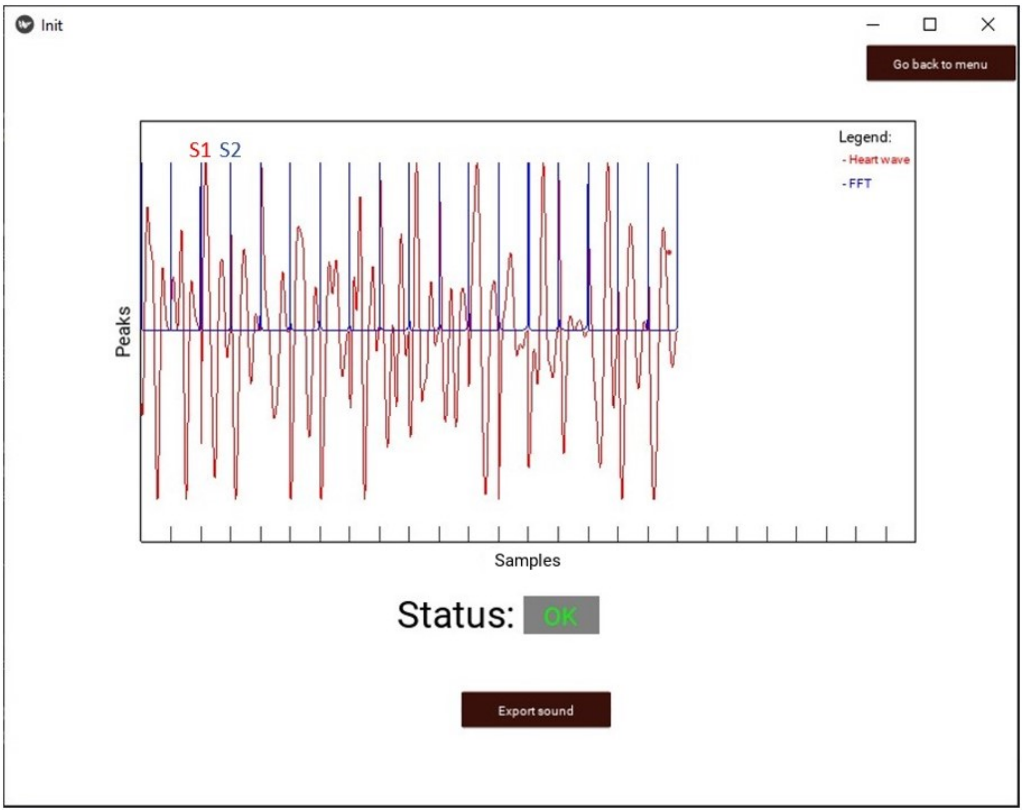


Figura 4.22: Representação Ecrã de análise de sinal S1 e S2

Se o sinal sonoro captado apresentar muito ruído, por vias de má posição do microfone ou microfone com muita pouca sensibilidade, dá-se um *pop-up* de erro ilustrado na Figura 4.23, o estado do status altera-se para ”NOK” a vermelho e é exportado um ficheiro .wav denominado de ”problemSound” com o som capturado sem alterações e um ficheiro .wav denominado de ”problemSound-filtered” com o condicionamento de sinal implementado, estes dois ficheiros estão apresentados na Figura 4.24.

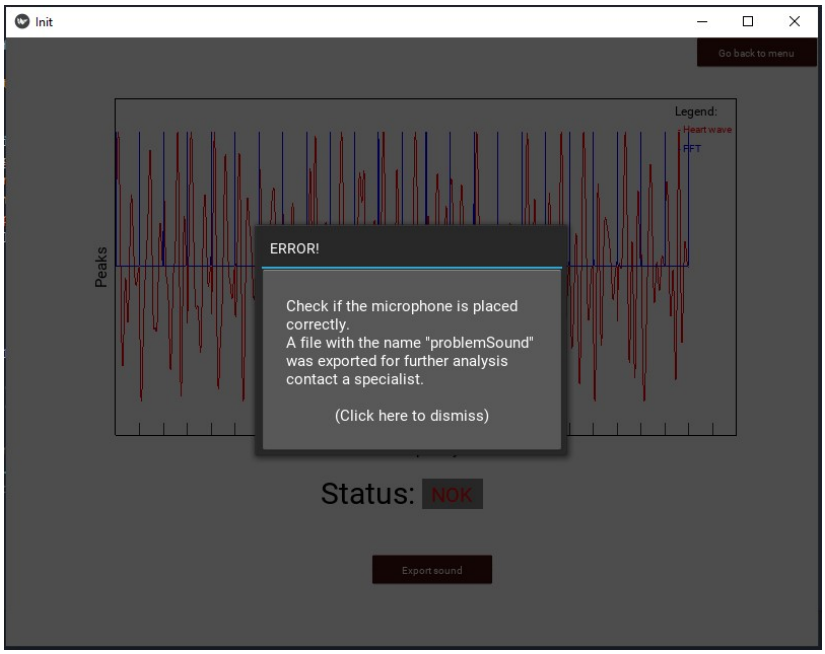


Figura 4.23: Representação do *pop-up* de ruído e alteração do estado do ”status”

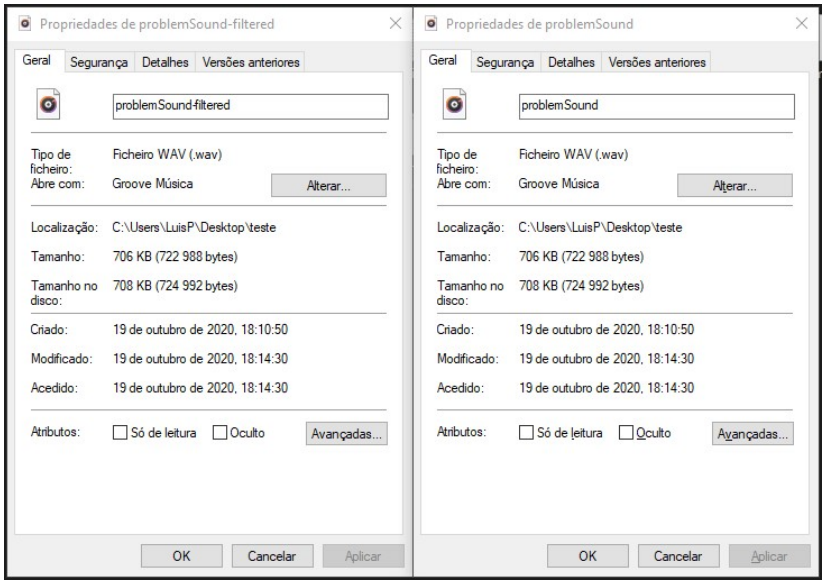


Figura 4.24: Ficheiros .wav ”problemSound” e ”problemSound-filtered”

Ao pressionar o botão ”Export sound” dá-se um *pop-up* de aviso que o ficheiro sonoro foi exportado com sucesso, este comportamento pode ser analisado na Figura 4.25. São exportados dois ficheiros do tipo .wav, um dos ficheiros é denominado de ”problemHeartSound” com o som capturado sem alterações e o outro

ficheiro ficheiro é denominado de ”problemSound-filtered” com o condicionamento de sinal implementado, estes dois ficheiros estão apresentados na Figura 4.26.

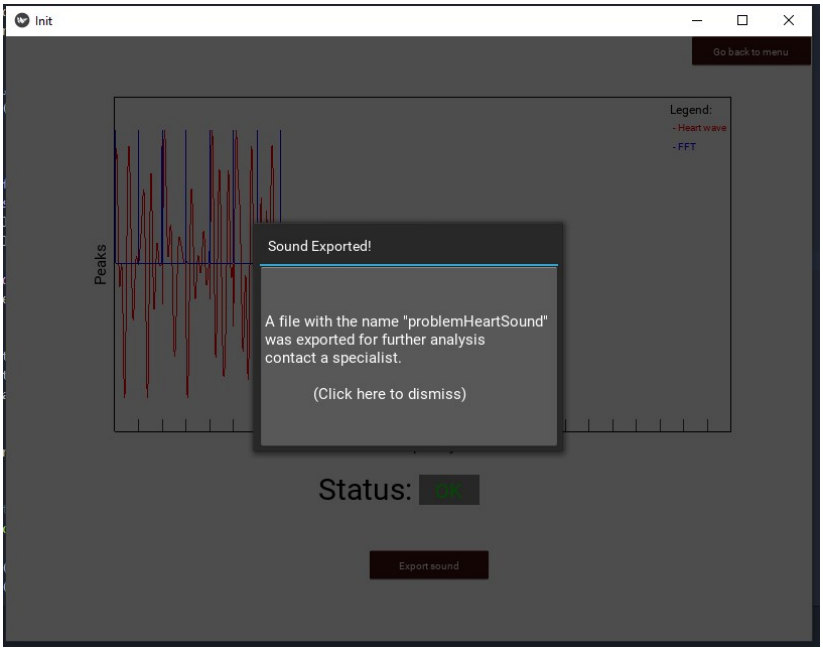


Figura 4.25: Representação do *pop-up* de exportação do som capturado

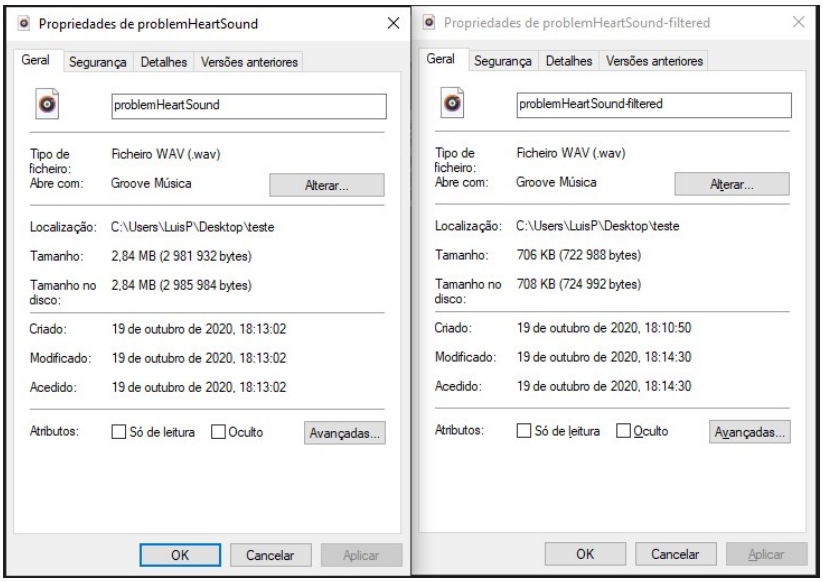


Figura 4.26: Ficheiros .wav ”problemHeartSound” e ”problemHeartSound-filtered”

Capítulo 5

Conclusões

Neste capítulo pretende-se apresentar as conclusões finais do projeto da dissertação desenvolvido, analisando os objetivos propostos e validar se foram cumpridos. São também apresentadas propostas de melhorias futuras e funcionalidades a implementar.

5.1 Conclusões do trabalho

Na presente dissertação é feita uma descrição pormenorizada de todo o processo do desenvolvimento de um sistema de monitorização e análise de sons emitidos pelo coração humano, um sistema de aquisição do PCG.

O protótipo final do sistema de aquisição do PCG é uma aplicação multiplataforma, desenvolvida em Python com recurso à *framework* Kivy. A aplicação divide-se em 3 ecrãs, sendo o primeiro o ecrã de menu que dá a possibilidade ao utilizador se querer conectar a um dispositivo de aquisição externo compatível ou continuar sem ligação a dispositivo; o segundo ecrã é o ecrã de ligação é executado caso o utilizador pretenda conectar a um dispositivo externo; o terceiro ecrã é de análise de sinal, sendo que neste ecrã o utilizador não se conecte a um dispositivo externo, os sinais sonoros são recolhidos pela própria aplicação através da biblioteca PyAudio e de um microfone conectado ao dispositivo onde a aplicação se encontra em execução. Se o utilizador se conectar os dados dos sinais sonoros são recolhidos pelo dispositivo externo e enviados para a aplicação.

O protótipo e o seu funcionamento foram validados em diferentes partes começando pela validação da conexão da aplicação, de seguida foi feita uma validação do condicionamento de sinal, posteriormente a validação da FFT, seguidamente a validação da aquisição do sinal sonoro através da aplicação e por fim a validação integração das diferentes implementações. Por fim, desenvolveu-se o protótipo final da aplicação com uma UI iterativa e de fácil utilização. Na aplicação é possível exportar ficheiros sonoros do tipo .wav, com o registo das leituras

dos sinais emitidos pelo coração, também é possível fazer uma análise gráfica ao sinal capturado. Conta também com boa manipulação de erros, sejam eles de má conexão ao dispositivo externo, má qualidade do sinal recebido ou má colocação do microfone. O utilizador pode utilizar a aplicação livremente, ou seja, pode navegar sobre os ecrãs da aplicação livremente sem necessidade de a fechar para retroceder algum passo.

O protótipo desenvolvido é de fácil utilização de baixo custo sem necessidade de recorrer a aparelhos externos, sendo os requisitos mínimos a utilização de qualquer máquina com um sistema operativo instalado. É recomendado utilização de um microfone externo para captura de sinal sonoro, caso se utilize sem dispositivo externo conectado.

Pode-se concluir que os objetivos foram cumpridos com sucesso, cumprindo com todos os requisitos. Porém, foram encontradas algumas dificuldades no desenvolvimento do projeto, tais como:

- Falta de testes da aplicação desenvolvida com conexão ao dispositivo externo, só foi testada por simulações;
- Falta de testes em ambientes macOS e iOS;
- Microfone utilizado durante o desenvolvimento da aplicação pouco sensível.

5.2 Propostas de melhoria e funcionalidades

Para desenvolvimento futuro da aplicação são propostas as seguintes implementações:

- Separação do gráfico FFT e o sinal sonoro emitido pelo coração;
- Importar ficheiros .wav de forma a acompanhar o registo do paciente;
- Integração de um ECG e representar graficamente o seu sinal de forma a ter uma avaliação do paciente mais completa;
- Deteção automática dos sons cardíacos S1, S2, S3 e S4;
- Adicionar ferramentas de anotação e manipulação do gráfico apresentado.

Bibliografia

[1] "Curiosidades sobre coração humano." http://estelajoaosara.blogspot.com/2014/06/curiosidades-sobre-o-coracao.html. Accessed: 2020-03-28. [cited on p. xi, 6]

[2] K. Kumar, "Block diagram of system flow for pcg signal analysis.." https://www.researchgate.net/figure/Block-diagram-of-system-flow-for-PCG-signal-analysis_fig2_274641376. Accessed: 2020-03-29. [cited on p. xi, 8]

[3] "Focos de ausculta cardíaca." https://scontent-yyz1-1.cdninstagram.com/v/t51.2885-15/sh0.08/e35/s640x640/100929467_285232552617332_2952753609079513063_n.jpg?_nc_ht=scontent-yyz1-1.cdninstagram.com&_nc_cat=108&_nc_ohc=DMlYVo-WFoAAX94jfN3&oh=abf12267221b58e355b097b206e75e5f&oe=5F83F687. Accessed: 2020-03-29. [cited on p. xi, 9]

[4] A. Kist, M. Baldasso, and M. Valk, "A novel heart sound activity detection framework for automated heart sound analysis." https://www.researchgate.net/publication/262806530_A_novel_heart_sound_activity_detection_framework_for_automated_heart_sound_analysis. Accessed: 2020-03-30. [cited on p. xi, 10]

[5] "Comece a programar em python." https://python.softonic.com.br/. Accessed: 2020-10-15. [cited on p. xi, 11]

[6] "Texture management in kivy using atlas." https://www.codementor.io/@kiok46/theming-in-kivy-0-yt8c94mbb. Accessed: 2020-10-15. [cited on p. xi, 11]

[7] "Socket programming in python: Client, server, and peer examples." https://hackernoon.com/socket-programming-in-python-client-server-and-peer-examples-a25c9782b584. Accessed: 2020-10-15. [cited on p. xi, 14]

[8] B. Mihai and P. Mihai, "Labview modeling and simulation of the digital filters." https://www.researchgate.net/publication/280878245_LabVIEW_Modeling_and_Simulation_of_The_Digital_Filters. Accessed: 2020-10-15. [cited on p. xi, 15]

[9] P. E. de Barbosa Monteiro, "Sistema de aquisição do pcg, miniaturizado, portátil," Outubro 2019. Instituto Superior de Engenharia do Porto. [cited on p. xi, 17, 18]

[10] "Estetoscópio de teleconsulta cms-vesd." https://www.medicalexpo.com/pt/prod/contec-medical-systems/product-68095-430707.html. Accessed: 2020-07-18. [cited on p. xi, 18]

[11] "Estetoscópio para cardiologia ekuore pro." https://www.medicalexpo.com/pt/prod/ekuore/product-107265-732567.html. Accessed: 2020-07-18. [cited on p. xi, 19]

[12] "Sistema cardiovascular: como funciona?." https://www.hospitaldaluz.pt/pt/guia-de-saude/dicionario-de-saude/S/157/sistema-cardiovascular-como-atua. Accessed: 2020-03-28. [cited on p. 5, 6, 7]

[13] Medipédia, "Coração - anatomia e fisiologia." https://www.medipedia.pt/home/home.php?module=artigoEnc&id=97. Accessed: 2020-03-28. [cited on p. 7]

[14] A. F. D. E. Santo, "Caraterização funcional cardíaca por fonocardiografia," Março 2016. Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria. [cited on p. 7, 9]

[15] A. Kist, M. Baldasso, and M. Valk, "Classicação de doenças cardíacas através de eletrocardiogramas e fonocardiogramas," Outubro 2017. Universidade Federal do Rio Grande do Sul. [cited on p. 7]

[16] D. R. R. Albuquerque, D. T. H. Coelho, and D. A. L. M. Valk, "Auscultação cardíaca." https://turmafcml0612.files.wordpress.com/2010/09/ausccard.pdf. Accessed: 2020-03-30. [cited on p. 10]

[17] "Python introduction." https://www.w3schools.com/python/python_intro.asp. Accessed: 2020-07-11. [cited on p. 10]

[18] A. Kist, M. Baldasso, and M. Valk, "Tiobe index for october 2020." https://www.tiobe.com/tiobe-index/. Accessed: 2020-07-11. [cited on p. 10]

[19] "What is python? executive summary." https://www.python.org/doc/essays/blurb/. Accessed: 2020-07-11. [cited on p. 10]

[20] "Kivy." https://kivy.org/#home. Accessed: 2020-07-11. [cited on p. 11]

[21] M. Driscoll, "Build a mobile application with the kivy python framework." https://realpython.com/mobile-app-kivy-python/. Accessed: 2020-07-11. [cited on p. 11, 12, 13]

[22] "Kv language." https://kivy.org/doc/stable/guide/lang.html. Accessed: 2020-10-15. [cited on p. 12]

[23] "Create a package for windows." https://kivy.org/doc/stable/guide/packaging-windows.html. Accessed: 2020-07-11. [cited on p. 12]

[24] "Creating packages for os x." https://kivy.org/doc/stable/guide/packaging-osx.html. Accessed: 2020-07-11. [cited on p. 12]

[25] G. Bajo, H. Goebel, D. Vierra, D. Cortesi, and M. Zibricky, "pyinstaller 4.0." https://pypi.org/project/pyinstaller/. Accessed: 2020-07-11. [cited on p. 12]

[26] "Creating packages for android." https://kivy.org/doc/stable/guide/packaging-android.html. Accessed: 2020-07-12. [cited on p. 12, 13]

[27] "Buildozer." https://github.com/kivy/buildozer. Accessed: 2020-07-12. [cited on p. 12, 13]

[28] "Creating packages for ios." https://kivy.org/doc/stable/guide/packaging-ios.html#. Accessed: 2020-07-13. [cited on p. 13]

[29] "Pyaudio documentation." https://people.csail.mit.edu/hubert/pyaudio/docs/. Accessed: 2020-07-13. [cited on p. 14]

[30] "Numpy." https://numpy.org/. Accessed: 2020-07-13. [cited on p. 14]

[31] "socket — low-level networking interface." https://docs.python.org/3/library/socket.html. Accessed: 2020-07-17. [cited on p. 14]

[32] "Python socket communication." https://medium.com/python-pandemonium/python-socket-communication-e10b39225a4c. Accessed: 2020-07-17. [cited on p. 14]

[33] "Scipy: Scientific library for python." https://pypi.org/project/scipy/. Accessed: 2020-07-17. [cited on p. 15]

[34] "What is scipyn." https://www.scipy.org/scipylib/faq.html#what-is-scipy. Accessed: 2020-07-17. [cited on p. 15]

[35] S. F. B. Hussin, G. a/p Birasamy, and Z. B. Hamid, "Design of butterworth band-pass filter," 2016. Polytechnic Tuanku Sultanah Bahiyah. [cited on p. 15]

[36] "Aquisição de sinais analógicos: largura de banda, teorema de amostragem de nyquist e aliasing." https://www.ni.com/pt-pt/innovations/white-papers/06/acquiring-an-analog-signal--bandwidth--nyquist-sampling-theorem-.html. Accessed: 2020-07-18. [cited on p. 16]

[37] "filtfilt." https://www.mathworks.com/help/signal/ref/filtfilt.html. Accessed: 2020-07-18. [cited on p. 16]

[38] "The fourier transform .com." http://www.thefouriertransform.com/. Accessed: 2020-10-15. [cited on p. 16]

[39] S. L. Brunton and J. N. Kutz, "Data driven science engineering - machine learning, dynamical systems, and control," 2017. [cited on p. 17]