Marco André Florindo Ferreira

Sistema de Visualização Ecográfica para Oftalmologia usando Dispositivos Android

Setembro de 2015

· U C ·

Universidade de Coimbra

Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

Dissertação de
Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Sistema de Visualização Ecográfica para Oftalmologia usando Dispositivos Android

Marco André Florindo Ferreira

Desenvolvido com a supervisão de:
Prof. Doutor Fernando Santos Perdigão
Prof. Doutor Marco Alexandre Cravo Gomes

Júri
Prof. Doutor Paulo José Monteiro Peixoto (Presidente)
Prof. Doutor Fernando Santos Perdigão (Orientador)
Prof. Doutor Gabriel Falcão Paiva Fernandes (Vogal)

Setembro de 2015

Agradecimentos

Em primeiro lugar gostaria de deixar um agradecimento especial ao Professor Doutor Fernando Perdigão pela sua grande ajuda no desenvolvimento desta dissertação, pela sua constante disponibilidade e pela grande contribuição que teve para o meu sucesso neste trabalho.

Agradeço também ao Professor Doutor Marco Gomes por toda a sua ajuda e disponibilidade ao longo deste projeto.

Quero deixar também um profundo agradecimento aos meus pais, por todo o seu trabalho e dedicação, determinantes para o meu sucesso académico.

Gostava também de deixar um enorme agradecimento à minha namorada, por todo o apoio e palavras de incentivo.

Finalmente, quero também agradecer a todos os meus amigos que tornaram estes anos numa das melhores fases da minha vida.

Resumo

Os ultrassons são sons com vibração de frequência tão alta que não são detetáveis pelo ouvido humano. Hoje em dia, estes sons são utilizados nas mais diversas áreas, contudo, destaca-se aqui a área da oftalmologia que estuda e trata doenças relacionadas com o olho. Existem atualmente nesta área vários instrumentos que recorrem ao uso de ultrassons para a visualização da estrutura interna do olho humano, permitindo uma análise rápida e eficiente. Na sua maioria, estes instrumentos apresentam-se com grande dimensão, e integram um sistema de visualização acoplado ao sensor de ultrassons. Com a tecnologia atual é possível separar a parte de instrumentação da de visualização, aumentando a mobilidade do operador. É aqui que os dispositivos móveis como os *smartphones* e *tablets* adquirem notoriedade. A sua utilidade parece não ter limite, sendo que cada vez é maior a aposta no desenvolvimento de aplicações para estes dispositivos. Assim, no âmbito desta dissertação, desenvolveu-se uma aplicação para o sistema operativo Android que, para além de permitir a visualização de imagens de estruturas internas do olho (imagens ecográficas), permite também a extração de dados biométricos sobre as mesmas.

Essencialmente, esta dissertação aborda o problema da visualização de um aparelho ultrassónico sob o ponto de vista do utilizador, um médico ou técnico ecográfico. A visualização contempla os modos de representação de imagem do tipo A-Scan e B-Scan, gere uma ligação cliente-servidor para atualização permanente das imagens e permite fazer medições sobre as imagens geradas de uma forma muito interativa.

A geração de imagens B-Scan é feita através de uma aplicação a correr no servidor (instrumento), detetando e interpretando os A-Scans individuais à medida que o sensor de ultrassons se desloca sobre o olho em análise. Como resultado deste processo é enviada uma imagem B-Scan para o visualizador (*smartphone* ou *tablet*). Permite-se também a visualização de imagens A-Scan individuais, identificação da região do cristalino, e ainda deteção e classificação de cataratas.

Para que tudo isto seja possível é necessário um estudo sobre diversas áreas de conhecimento. Em primeiro lugar, mencionamos a teoria dos ultrassons associada ao processo de aquisição de

dados que permite a criação de imagens ecográficas. Posteriormente, temos a utilização da linguagem de programação C++ para o desenvolvimento de um *software* que cria imagens ecográficas do tipo B-Scan. A finalidade deste *software* é a criação de uma imagem para posterior visualização num dispositivo Android e, por isso, é também necessário um conhecimento acerca da linguagem de programação Java orientada para o desenvolvimento de uma aplicação Android. Por sua vez, para o desenvolvimento desta aplicação, em particular das funcionalidades de dimensionamento e deslocação da imagem e ainda na medição de distâncias sobre a mesma, é necessário uma aprendizagem sobre geometria analítica. Finalmente, para que tudo isto faça sentido desenvolveu-se uma aplicação na linguagem de programação C# que é responsável pela transferência de dados entre o servidor (que simula o instrumento ultrassónico) e o seu cliente.

Palavras-chave: ultrassons, oftalmologia, A-Scan, B-Scan, Android, processamento digital de sinal, transformações geométricas

Abstract

Ultrassounds are sounds with such high frequency vibrations, that are not detectable by the human ear. Today, these sounds are used in several areas, however, stands out here the area of ophthalmology that studies and treats diseases related to eye. Currently exist, in this area, several instruments that resort to the use of ultrasound to visualize the internal structure of the human eye, allowing for a quick and efficient analysis. Most of these instruments are presented with large dimensions, and they integrate a display system coupled to the ultrasonic sensor. With current technology it is possible to separate the display from the instrumentation, increasing the operator's mobility. This is where mobile devices like smartphones and tablets acquire notoriety. Its usefulness seems to have no limit, and each time there is a greater focus on developing applications for these devices. In the context of this thesis, we developed an application for the Android operating system that, in addition to allowing viewing images of internal structures of the eye (ultrasound images), also allows the extraction of biometric data about them.

Essentially, this dissertation discusses the problem of an ultrasonic visualization device in the user's point of view, a physician or ultrasound technician. The display includes the image presentation modes A-Scan and B-Scan, manages a client-server connection for permanent updating of images and allows you to make measurements on the generated images in a very interactive way.

The generation of B-Scan images is made using an application running on the server (instrument), detecting and interpreting the individual A-Scans as the ultrasound sensor moves on the eye under examination. The result of this process is a B-Scan image, that is sent to the mobile device (smartphone or tablet). It also allows viewing individual A-Scan images, identification of the lens region, and cataracts detection and classification.

For all this to be possible a study of the most diverse areas of knowledge is required. In the first place, we mention the theory of ultrassounds associated to the acquisition of data that allows the creation of ultrasound images. Subsequently, we use the C++ programming language

for developing a software which creates echographic images of type B-Scan. The purpose of this software is to create an image for later viewing on an Android device, so you also require a knowledge of the Java programming language oriented to developing an Android aplication. In turn, the development of the application, in particular the features of scaling and moving the image and the distance measurement, require a learning process about analytical geometry. Finally, so that everything makes sense, we developed an application in the C# programming language that is responsible for transferring data between the server and the client.

Keywords : ultrasound, ophthalmology, A-Scan, B-Scan, Android, Digital Signal Processing, Geometric Transformations

Índice

Lista de Figuras **iii**

Lista de Tabelas **vi**

Lista de Acrónimos **ix**

1 Introdução **1**

1.1 Objetivos 2

1.2 Estrutura da Dissertação 3

2 Tecnologias de Suporte **5**

2.1 Introdução à Teoria dos Ultrassons em Ecografia 5

2.2 Introdução ao Android 8

2.2.1 O Sistema Operativo 8

2.2.2 Ferramentas de Desenvolvimento 9

2.2.3 Componentes de uma Aplicação 9

2.2.4 Anatomia de uma Aplicação 10

3 Criação de Imagens B-Scan **11**

3.1 O Sistema de Aquisição 12

3.2 Deteção dos Pulsos de Emissão 13

3.3 Processamento dos Sinais Ultrassónicos 14

3.3.1 Filtragem Passa Banda 16
3.3.2 Filtragem Hilbert 18
3.3.3 Realização de Médias 20
3.4 A Matriz Final da Imagem 26
3.5 Determinação de Dados Biométricos 27
3.6 Envio da Imagem Ecográfica para o Cliente 28

4 Arquitetura Servidor Cliente **29**
4.1 ASP.NET 29
4.2 A Aplicação Web 30

5 A Aplicação Android **31**
5.1 A Interface Gráfica 31
5.2 Transferência, Construção e Desenho das Imagens 32
5.2.1 Transferência dos Dados 32
5.2.2 Construção das Imagens 34
5.2.3 Desenho das Imagens 35
5.3 Transformações Geométricas 36
5.3.1 Dimensionamento e Deslocação da Imagem 37
5.4 Medir Distâncias sobre a Imagem 41
5.5 Considerações sobre as Imagens A-Scan 42

6 Resultado Final **43**

7 Conclusão **45**

Bibliografia **48**

A Transformações Geométricas Iniciais **49**

B Transformações Geométricas Utilizando Gestos **53**

Lista de Figuras

1.1 Dois dispositivos para visualização de imagens ecográficas. Retirado de [14] e [17]. 2

2.1 Esquema representativo da emissão e receção de ultrassons, utilizando um sensor piezoelétrico. 6

2.2 Representação dos modos de aquisição e criação de imagens ecográficas. Editado de [20]. 7

2.3 Diagrama da arquitetura do Android. 9

3.1 Sistema utilizado na aquisição de sinais ultrassónicos. 12

3.2 Representação dos níveis de amplitude utilizados para deteção de um pulso de emissão. 14

3.3 Representação do módulo (a) e da fase (b) da resposta em frequência do filtro passa banda. Em (c) está representado o atraso de grupo deste filtro. 16

3.4 Diagrama de fluxo do sistema para filtragem IIR. 17

3.5 Representação do módulo (a) e da fase (b) da resposta em frequência do filtro de Hilbert. 18

3.6 Representação do atraso de grupo do filtro de Hilbert. 19

3.7 Diagrama de fluxo do sistema para filtragem FIR. 19

3.8 Estrutura de dados do *buffer* circular pertencente à classe `CircularBuffer`. . . . 22

3.9 Gráfico representativo do processo de avanço da janela de médias na horizontal. . 24

3.10 Exemplo ilustrativo da realização de médias na horizontal. 25

4.1 Comunicação entre o servidor Web e a aplicação Android. 30

5.1 Interface gráfica da aplicação. 32

5.2 Representação do sistema de coordenadas utilizado pela classe `Canvas`. 36

5.3 Representação de três gestos de toque. Editado de [11]. 38

6.1 Demonstração para obtenção de uma imagem B-Scan. 43

6.2 Demonstração da medição de distâncias em imagens B-Scan. 44

6.3 Demonstração para obtenção de uma imagem A-Scan 44

6.4 Demonstração da medição de distâncias em imagens A-Scan. 44

A.1 Referênciais da imagem e do *Canvas*. 49

A.2 A imagem é dimensionada de forma a preencher os limites do *Canvas* em altura. . 50

A.3 A imagem é dimensionada de forma a preencher os limites do *Canvas* em largura. 51

A.4 Escalamento dos referenciais da imagem para os referenciais do *Canvas*. 52

B.1 Referenciais da imagem e do *Canvas*. 53

B.2 Deslocação de uma imagem no *Canvas*. 54

B.3 Dimensionamento de uma imagem no *Canvas*. 55

Lista de Tabelas

3.1 Membros e métodos da classe `CircularBuffer`. 21

Lista de Acrónimos

PRF: Pulse Repetition Frequency

PRP: Pulse Repetition Period

SO: Sistema Operativo

IDE: Integrated Development Environment

SDK: Software Development Kit

ADT: Android Development Tools

IIR: Infinite Impulse Response

FIR: Finite Impulse Response

ASP: Active Server Pages

IIS: Internet Information Services

Capítulo 1

Introdução

É cada vez maior o avanço da tecnologia ao serviço da humanidade. Atualmente, na área da medicina, a tecnologia tem desempenhado um papel fundamental no auxílio ao tratamento e prevenção de determinadas doenças. É neste contexto que surgem os ultrassons, e a sua utilidade para a medicina. Em particular destaca-se aqui a sua contribuição para a área da oftalmologia – "especialidade da medicina que investiga, estuda, diagnostica e trata as doenças relacionadas com os olhos, com a visão e estruturas afins" [18].

Atualmente, existem vários instrumentos que recorrem ao uso de ultrassons para a visualização da estrutura interna do olho humano. Entre as suas várias funcionalidades, salienta-se a realização de biometrias e ecografias, que auxiliam num diagnóstico mais rápido e eficiente de determinados problemas da visão. Essencialmente, estes instrumentos podem ser divididos em duas partes. Uma parte diz respeito à aquisição e processamento da informação necessária para a criação de uma imagem. A outra parte é relativa ao sistema de visualização para a imagem criada. A título de exemplo, a figura 1.1 ilustra alguns dos equipamentos oftalmológicos de ultrassons, utilizados hoje em dia. O primeiro (lado esquerdo) é um instrumento portátil e possui uma sonda acoplada, permitindo a sua utilização em qualquer lugar, apresentando assim a portabilidade como fator principal. Em contrapartida, o segundo constitui exatamente o oposto. Tem um aspeto robusto e difícil mobilidade, contudo, apresenta a seu favor, o maior número de funcionalidades que possui. Como fator comum a estes dois instrumentos, temos o facto do sistema de visualização estar fisicamente incorporado com a instrumentação relativa à aquisição e processamento dos dados. Atualmente, esta característica encontra-se presentes em muitos dos equipamentos existentes, contudo, com a tecnologia atual é possível separar a parte de instru-

Figura 1.1: Dois dispositivos para visualização de imagens ecográficas. O dispositivo da esquerda é um Biómetro/Paquímetro portátil, modelo AP2000 (retirado de [14]). À direita apresenta-se o Biómetro/Paquímetro US-500, não portátil, e de muito maiores dimensões (retirado de [17]).

mentação da de visualização, aumentando a mobilidade do operador. Propõe-se então a criação de uma alternativa aos aparelhos oftalmológicos já existentes. Com a crescente evolução dos dispositivos móveis como os *smartphones* e *tablets*, porque não desenvolver um sistema de visualização ecográfica para oftalmologia, assente neste crescimento tecnológico? Surgiu assim a ideia de desenvolver um protótipo de uma aplicação para a visualização de imagens ecográficas, com funcionalidades semelhantes às dos equipamentos atuais, contudo redirecionada para dispositivos móveis e suportada pelo sistema operativo Android.

1.1 Objetivos

O objetivo principal consiste em criar uma aplicação Android, para *smartphones* e *tablets*, que constitua um sistema de visualização para oftalmologia, com a particularidade de estar fisicamente separado do sistema de aquisição e processamento dos dados, dando assim um verdadeiro sentido à mobilidade destes dispositivos. Desta maneira, o seu utilizador poderá visualizar imagens ecográficas, aquando estas estão a ser criadas, em qualquer lugar e a qualquer hora.

No presente trabalho, uma vez que não era necessário dispor de um instrumento ecográfico, este é simulado por um servidor que tem acesso a dados ecográficos, os processa e envia para o visualizador (*smartphone* ou *tablet*). Estes dados ecográficos encontram-se presentes no servidor sob a forma de ficheiros binários. Um destes ficheiros contém os dados adquiridos em modo B-

Scan, sendo que o outro contém os dados adquiridos em modo A-Scan. Estes modos representam formas de criar imagens ecográficas e serão abordados no próximo capítulo. É de salientar que o último ficheiro resulta de uma dissertação de Mestrado [12]. Este, para além de conter os dados adquiridos em modo A-Scan, já processados, prontos para serem visualizados, contém ainda outras informações relevantes. Entre estas salienta-se a localização de determinadas estruturas do olho e ainda a identificação automática de cataratas e respetiva classificação.

Para a visualização de imagens ecográficas do tipo B-Scan, foi criada uma solução eficiente em C++, com base num projeto já existente, desenvolvido em MATLAB. A ideia é que, a pedido da aplicação, este programa seja executado pelo servidor, processando os dados presentes no ficheiro adquirido em modo B-Scan, e, de seguida, envie o resultado desse processamento para a aplicação.

Do outro lado, a aplicação Android, desenvolvida em Java, será responsável por dar algum significado a todos estes dados, permitindo essencialmente a sua visualização, através de imagens representativas dos modos de aquisição A-Scan e B-Scan.

O sistema desenvolvido foi testado com dados B-Scan de cristalinos de suíno e dados A-Scan de olhos de rato.

1.2 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma: o capítulo 2 contém uma breve introdução sobre a utilização dos ultrassons em ecografia, isto é, explica alguns conceitos teóricos relacionados com o processo da obtenção de imagens ecográficas a partir de ondas ultrassónicas. Para além disso contém também uma breve introdução ao sistema operativo Android, utilizado pela aplicação desenvolvida. No capítulo 3 é apresentado inicialmente o sistema utilizado para a aquisição de sinais ultrassónicos. De seguida explicamos o programa, desenvolvido em C++, que faz o processamento desses sinais, e do qual resulta um conjunto de dados que permitem a visualização de uma imagem ecográfica do tipo B-Scan. Posteriormente surge o capítulo 4 que explica a arquitetura de comunicação entre o servidor, que contém os dados adquiridos pelo sistema de aquisição, e a aplicação, que poderá ser executada num *tablet* ou num *smartphone*. O capítulo 5 apresenta o desenvolvimento da aplicação Android. Nele é explicada a estrutura geral da aplicação, o processo que permite a visualização das imagens a partir dos dados recebidos,

e também os algoritmos desenvolvidos para as várias funcionalidades que esta possui. Seguidamente apresenta-se no capítulo 6 o resultado final obtido, que neste caso constitui a aplicação que foi desenvolvida e as suas várias funcionalidades. Finalmente, no capítulo 7 será feita uma conclusão final sobre o trabalho desenvolvido e serão apresentadas propostas para trabalho futuro.

Capítulo 2

Tecnologias de Suporte

Como já referido, esta dissertação centra-se essencialmente no desenvolvimento de uma aplicação Android que por si só constitua um sistema de visualização de imagens ecográficas para oftalmologia. Não tem a pretensão de abranger todos os conceitos relacionados com ecografia, muito menos com oftalmologia, contudo importa mencionar alguns aspetos fundamentais, que possibilitem ao leitor algum conhecimento sobre os diferentes temas que iremos abordar. Para além disso será feita também uma ligeira introdução ao sistema operativo que suporta a aplicação, o Android.

2.1 Introdução à Teoria dos Ultrassons em Ecografia

O som é uma vibração, transmitida num determinado meio, com uma determinada frequência. Para os humanos, o som é percetível para uma gama de frequências de 20Hz a 20kHz. Logo a seguir a esta faixa de frequências estão os ultrassons (acima de 20kHz e até 2MHz), que não são percetíveis pelo ouvido humano. É através dos ultrassons, que os morcegos, por exemplo, veem o mundo. Eles emitem som, que ao incidir em objetos, é refletido de volta para eles, permitindo-lhes saber a localização desses objetos. Conseguem assim mapear o meio que os envolve através dos sons que emitem.

Os ultrassons são também utilizados para obtermos representações das estruturas internas do corpo humano, neste caso do olho, mas para isso precisamos de algo que nos permita emitir este tipo de som e conseguir obter as informações desejadas. Existe um material, o qual designamos por cristal piezoelétrico, cujas propriedades apresentam-se como uma solução para este problema.

Este cristal é utilizado por um sensor piezoelétrico, presente nos aparelhos de ultrassons. Quando uma tensão é aplicada ao cristal, ele expande-se, voltando ao estado normal quando a tensão que lhe foi inicialmente aplicada deixa de o ser. Com base neste princípio, se conseguirmos aplicar e retirar uma tensão, com uma determinada frequência, conseguimos emitir ultrassons. Os sensores piezoelétricos podem também funcionar como recetores de ondas sonoras. Uma vez mais, derivado às propriedades deste cristal, quando as ondas sonoras são refletidas, e o atingem, ele comprime-se e consequentemente gera uma tensão. Esta tensão corresponde à intensidade da onda refletida que o atingiu.

Resumidamente, com base nas propriedades do cristal piezoelétrico, podemos criar um instrumento que consiga emitir ultrassons, e que receba as respetivas reflexões, criando assim um sinal cuja amplitude representa a intensidade das diferentes reflexões recebidas ao longo do tempo. Estas reflexões denominam-se ecos, e é a partir destes que conseguimos visualizar a estrutura interna do olho.

A título ilustrativo, atente na figura 2.1. O sensor emite uma onda sonora, que ao incidir no objeto é refletida de volta para ela, o que gera uma tensão. Um sistema de aquisição irá interpretar as diferentes tensões recebidas. Cria-se assim um sinal com uma única variação de amplitude, que corresponde ao momento em que a sonda recebeu a onda refletida pelo objeto.

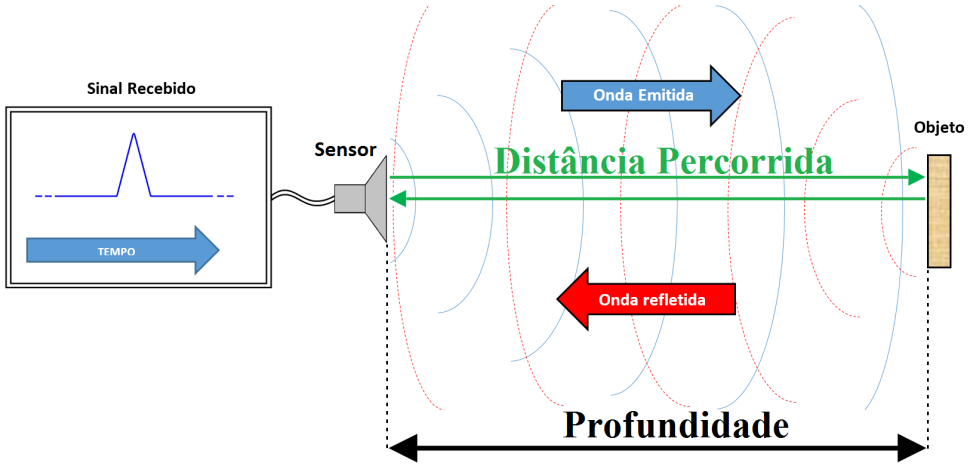


Figura 2.1: Esquema representativo da emissão e receção de ultrassons, utilizando um sensor piezoelétrico.

É aqui que surge o conceito de ecografia. Como o nome sugere, é uma representação dos ecos produzidos pelo som à medida que este se propaga. A partir destes ecos podemos obter uma representação da estrutura interna do olho humano, mas para isso estes precisam de ser interpretados. Esta interpretação é feita a partir de um conjunto de variáveis que nos permitem

representar as intensidades dos diferentes ecos recebidos ao longo do tempo/profundidade. Estas variáveis definem as equações 2.1 e 2.2 apresentadas de seguida.

$$t = \frac{d}{v} \Leftrightarrow t = \frac{2 \times \text{Profundidade}}{v}[s] \tag{2.1}$$

$$\text{Profundidade} = \frac{t \times v}{2}[m] \tag{2.2}$$

A partir da equação 2.1, calcula-se o tempo decorrido desde a transmissão até à receção do eco, ao qual chamamos tempo de voo e designamos por t. Note-se que v designa a velocidade de propagação das ondas ultrassónicas, que nesta dissertação é determinada em função da temperatura da solução onde é imerso o olho/cristalino, e d a distância percorrida pelas ondas (ida e volta). De seguida, a partir do intervalo de tempo determinado, calcula-se a profundidade da superfície na qual ocorreu a reflexão, como demonstrado na equação 2.2.

Existem alguns modos para a representação de uma imagem ecográfica, mas segundo o interesse desta dissertação, falaremos apenas de dois. São eles os modos de *scan* tipo A e tipo B, também conhecidos como A-Scans e B-Scans. Na figura 2.2 apresentada a seguir, é ilustrado o funcionamento destes modos.

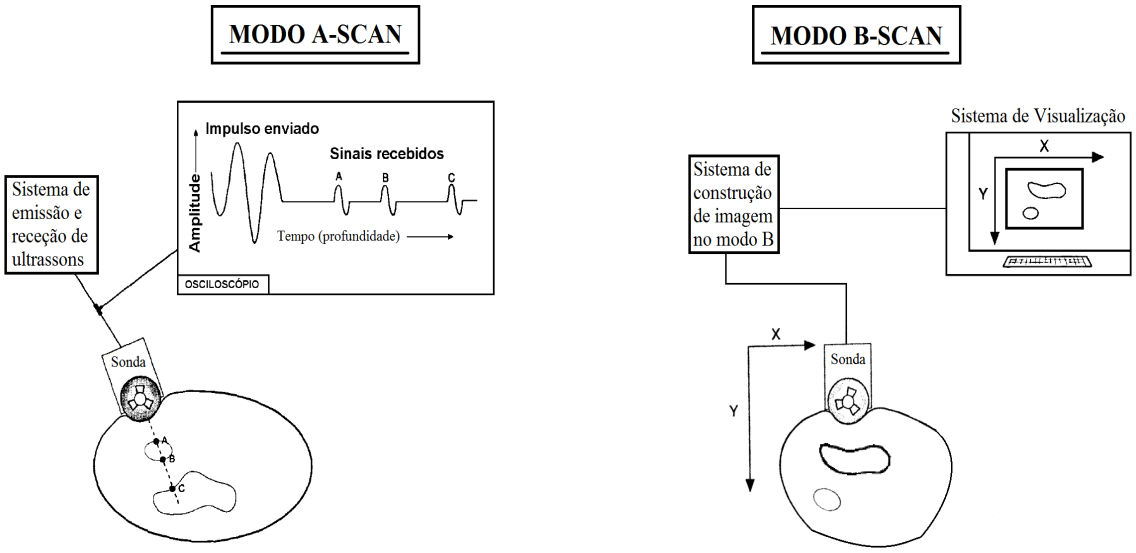


Figura 2.2: Representação dos modos de aquisição e criação de imagens ecográficas. Editado de [20].

O modo A-Scan é a forma mais simples de representação de imagem em ultrassons. Trata-se de um sinal variante no tempo ou em profundidade, representativo das diferentes tensões geradas pelos ecos recebidos. É por isto que é um modo de *scan* tipo A, um *scan* de amplitudes. Por outro lado, no modo B-Scan obtém-se uma imagem propriamente dita, o que envolve um modo de varrimento do sensor ultrassónico. No caso da figura 2.2 (modo B-Scan), o sensor desloca-se

horizontalmente emitindo sinais ultrassónicos, a uma determinada frequência, e recebendo os respetivos ecos. Por sua vez, a intensidade de cada eco recebido é representada em termos de brilho (daí a letra B) em duas dimensões (x e y), criando assim uma imagem. Em [20] pode ser encontrada uma descrição detalhada acerca destes modos.

Relativamente ao modo de representação B-Scan, a frequência a que são gerados os pulsos de emissão, isto é, os ciclos de emissão de ultrassons, é chamada de *Pulse Repetition Frequency* (PRF), que se traduz como sendo a frequência de repetição de um pulso, e que é medido em pulsos por segundo ou Hertz. O seu valor pode ser calculado a partir da equação 2.3.

$$\text{PRF} = \frac{\text{Velocidade de Propagação}}{\text{Distância entre Pulsos}}\,[Hz] \tag{2.3}$$

Supondo que a aquisição do sinal ecográfico é feita de forma contínua, sabendo que os pulsos são gerados a um ritmo fixo, é possível calcular o PRF através da deteção dos instantes de emissão dos pulsos. Isto é, o intervalo entre estes pulsos de emissão, que é designado por PRP (Período de Repetição de Pulso), tem valor igual ao inverso do PRF.

2.2 Introdução ao Android

O Android é um Sistema Operativo (SO) para dispositivos móveis, desenvolvido pela Open Handset Alliance, liderada pela Google e outras empresas. De entre estes dispositivos destacam-se os *smartphones* e os *tablets*.

2.2.1 O Sistema Operativo

Como descrito em [11], a arquitetura deste sistema operativo é composta por cinco secções agrupadas em quatro camadas de *software* (figura 2.3):

Linux Kernel - O Android usa uma versão modificada do *kernel* do Linux. Esta é a camada mais baixa do SO.

Libraries - Esta camada contém o código necessário para suportar as principais características deste SO como, por exemplo, gráficos, Web, *media*, etc.

Android runtime - Disponibiliza um conjunto de bibliotecas que permitem aos programadores desenvolver aplicações Android usando a linguagem de programação Java. Inclui também

a máquina virtual Dalvik onde as aplicações são executadas.

Application Framework - Esta camada é desenvolvida quase toda em Java, e faz a interface com as aplicações Android.

Applications - É aqui que ficam as aplicações (desenvolvidas em Java) para o SO Android.

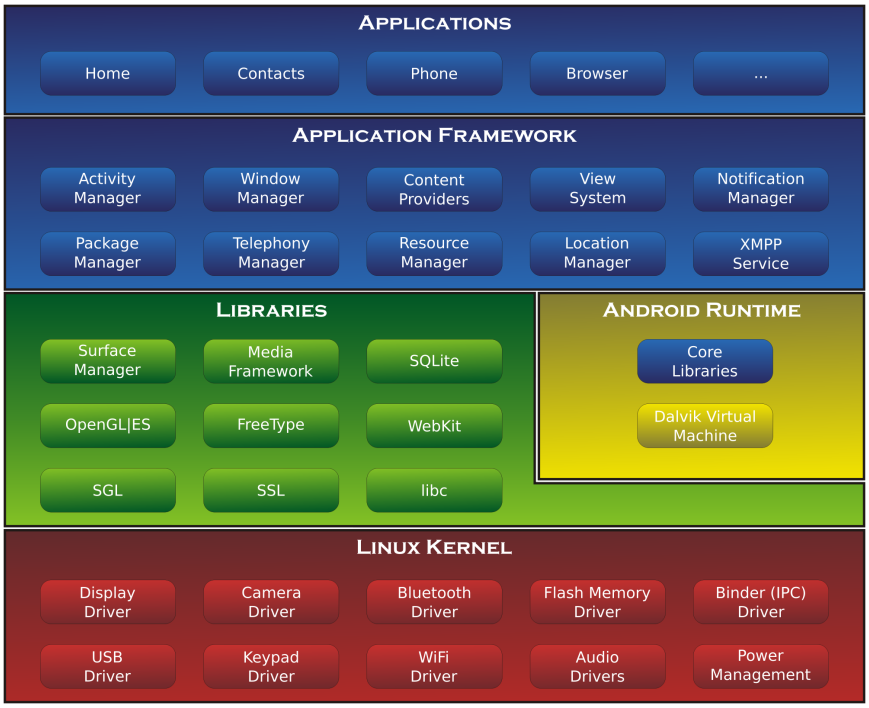


Figura 2.3: Diagrama da arquitetura do Android.

2.2.2 Ferramentas de Desenvolvimento

Uma aplicação desenvolvida para o SO Android é programada em Java, mas antes de iniciar esse processo é necessário preparar o computador com um conjunto de ferramentas de *software*. No nosso caso o ambiente de desenvolvimento integrado (IDE - *Integrated Development Environment*) utilizado foi o Eclipse. Utilizamos ainda o Android SDK (*Software Development Kit*), que integra um conjunto de ferramentas necessárias para criar, compilar, testar e distribuir aplicações Android. Finalmente, utilizamos também um *plugin* para o Eclipse denominado ADT(*Android Development Tools*), que permite a criação, compilação, depuração e distribuição de aplicações.

2.2.3 Componentes de uma Aplicação

As aplicações são compostas por um ou mais componentes, [3]. Estes componentes podem ser:

Atividade - Representa o ecrã de uma aplicação com uma interface gráfica de utilizador.

Serviço - Componente executado em segundo plano para operações de longa duração ou controladas por processos remotos. Um serviço não fornece uma interface gráfica.

Recetores de *Broadcast* - Respondem à transmissão de mensagens do sistema. Um exemplo de transmissão do sistema poderia ser uma transmissão a dizer que a bateria está fraca.

Fornecedores de Conteúdo - Gerem um conjunto de dados na aplicação. Estes dados podem ser armazenados no sistema de ficheiros, numa base de dados SQLite, na Web, ou em qualquer outro local de armazenamento persistente a que a aplicação possa aceder.

2.2.4 Anatomia de uma Aplicação

O projeto associado a uma aplicação Android contém uma série de pastas e ficheiros. Entre eles, salientamos aqui o ficheiro *androidmanifest.xml* e a pasta dos recursos da aplicação, *res*.

O ficheiro *androidmanifest.xml* é escrito na linguagem XML, e é automaticamente criado no projeto desta aplicação. Como enunciado em [2], ele contém a informação essencial para a execução da aplicação: descreve os componentes da aplicação, define os nomes para as atividades, define os modos de orientação do ecrã, declara permissões para acesso a determinados recursos, entre outras informações que fornece ao sistema.

Por outro lado, a pasta *res* contém os recursos da aplicação (interface gráfica, imagens, menus, *strings*, etc.). Em [10], pode ser encontrada uma descrição detalhada acerca desta pasta e do seu conteúdo.

Capítulo 3

Criação de Imagens B-Scan

Usualmente o processo de criação de imagens ecográficas do tipo B-Scan, no seu ponto de origem, pressupõe um tipo de aquisição em que é utilizado um único sensor, que se desloca a uma velocidade constante sobre o objeto. Noutros casos a sonda tem um *array* de sensores, cada um detetando diferentes partes do objeto, podendo não existir movimento. No presente caso consideramos apenas a situação em que um sensor varre o cristalino de um olho da esquerda para a direita, linearmente, a uma velocidade constante de 1cm/s. Além disso, o sensor emite pulsos na direção perpendicular ao seu movimento com um PRF fixo. Desta forma, o sistema de aquisição adquire o sinal em contínuo durante o varrimento do sensor e coloca os dados num ficheiro. Posteriormente, este ficheiro é interpretado por um programa, desenvolvido em C++, de forma a produzir uma imagem ecográfica do tipo B-Scan.

Pretende-se que o utilizador deste programa, tenha controlo sobre determinados parâmetros, de tal forma que a imagem seja o mais clara possível. Assim, o utilizador pode optar por diferentes configurações para a criação da imagem final pretendida. Poderá escolher a profundidade inicial e final, a percentagem do deslocamento efetuado pelo sensor e ainda a resolução da imagem em altura e largura. O significado destes parâmetros e a sua implicação para a imagem ecográfica final serão abordados no desenrolar deste capítulo.

Inicialmente faremos uma pequena abordagem ao sistema de aquisição utilizado. Posteriormente, explicaremos com um maior detalhe o processo de criação da imagem B-Scan, no que diz respeito ao código desenvolvido.

3.1 O Sistema de Aquisição

O sistema de aquisição constitui o processo de recolha de sinais ultrassónicos, do qual resultam um conjunto de dados representativos desses sinais, que são armazenados num ficheiro binário. Como anteriormente referido, o processo mecânico de aquisição pressupõe um deslocamento do sensor piezoelétrico a uma velocidade constante, enquanto os dados são adquiridos.

Essencialmente, importa aqui referir as características principais deste sistema. A placa de aquisição de dados utilizada é da marca Adlink, modelo PCIe-9842, com frequência de amostragem até 200 MHz, 14 bits de resolução e 1 canal, [19]. Dispõe de *software* de aquisição que permite guardar os dados adquiridos à frequência máxima de 200MHz. Os valores das amostras adquiridas são definidos com inteiros de 16 bits de resolução (de -32768 a 32767). A figura 3.1 ilustra o sistema utilizado para a aquisição de sinais ultrassónicos.

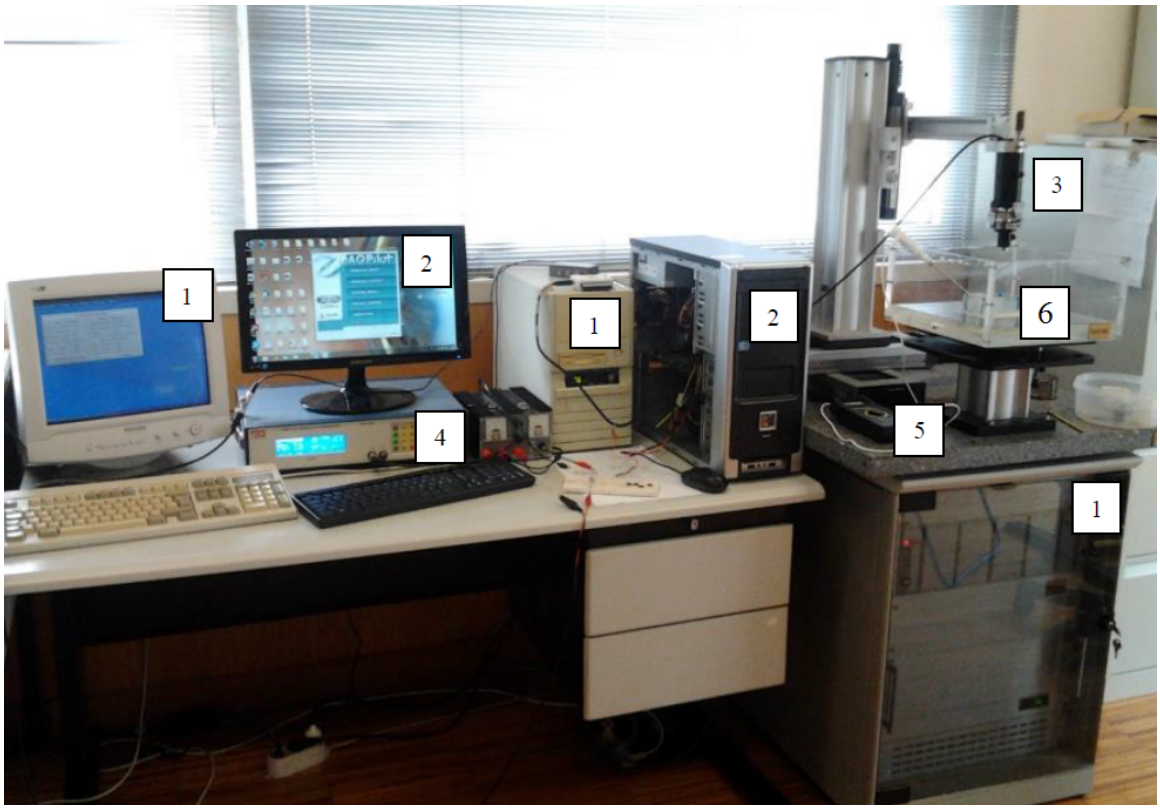


Figura 3.1: Sistema utilizado na aquisição de sinais ultrassónicos, onde 1 identifica a instrumentação associada ao sistema de posicionamento, 2 a instrumentação associada à aquisição, 3 o sensor piezoelétrico, 4 o gerados de sinais, 5 o termómetro e 6 a tina e suporte para colocação do cristalino.

Com os dados adquiridos, inicia-se agora o processo de criação da imagem ecográfica B-Scan.

3.2 Deteção dos Pulsos de Emissão

O ficheiro resultante do processo de aquisição, mencionado anteriormente, tem um cabeçalho de 60 bytes e os dados restantes constituem o sinal adquirido. Essencialmente o cabeçalho do ficheiro permite-nos saber a frequência de amostragem, o tamanho em bytes das amostras e o número total de amostras que o ficheiro contém. Contudo, precisamos ainda de determinar qual o valor do PRF e o início dos pulsos de emissão, para assim podermos ter acesso aos A-Scans. Inicialmente o valor do PRF não é obtido a partir dos dados presentes no ficheiro, mas sim a partir da sua nomenclatura, que obedece a um determinado conjunto de regras especificadas no protocolo de aquisição destes sinais. Para além disto, a partir do nome do ficheiro conseguimos também extrair a velocidade de deslocamento do sensor piezoelétrico e a temperatura do líquido em que o olho está submerso. Caso esta nomenclatura esteja incorreta, é necessário atribuir valores a estes parâmetros. Então, por defeito, atribuem-se 10 mm/s para o deslocamento do sensor. Por outro lado, a temperatura do líquido de imersão é calculada em função da velocidade de propagação das ondas ultrassónicas, cujo valor determinado foi de 1482.5 m/s para uma temperatura de 27°C. Finalmente, para um PRF omitido ou mal especificado, na nomenclatura do ficheiro, desenvolvemos um método capaz de o determinar através da deteção dos pulsos de emissão.

Quando os pulsos são emitidos (por vezes com centenas de Volt) o amplificador de entrada é desligado. Mesmo assim, no sinal adquirido a amplitude observada é muito elevada, quase sempre saturando o conversor analógico-digital. Esta característica permite a deteção dos pulsos de emissão e, por isso, analisando os primeiros pulsos de emissão do ficheiro, consegue-se estimar um valor para o PRF. Assim, desenvolveu-se uma solução que percorre as amostras dessa porção inicial do ficheiro, procurando por aquelas que correspondem ao início dos pulsos de emissão. Para isso, definimos três níveis de amplitude no sinal, como ilustra a figura 3.2. O nível baixo assumia um valor de amplitude inferior a 4000, o nível alto superior a 30000 e o nível médio situava-se entre estes dois níveis. Para além disto, sabíamos que para o nível alto, o pulso de emissão encontrava-se muito saturado, logo assumimos que se encontrássemos mais de 10 amostras consecutivas, no nível alto, estaríamos na presença de um pulso de emissão. Então, segundo esta abordagem, e a partir do número de pulsos de emissão obtidos, calculamos o período de pulso (PRP) pela média dos períodos detetados, e finalmente o PRF.

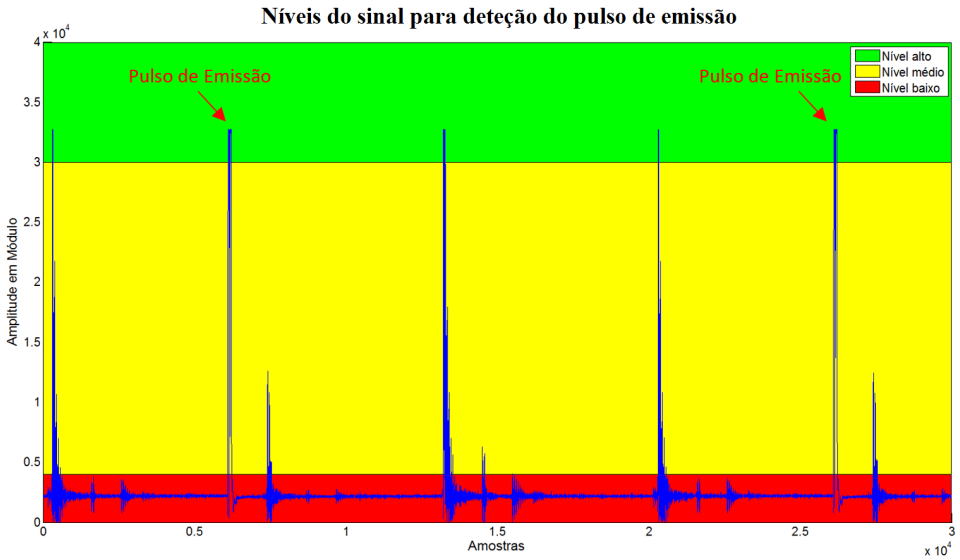


Figura 3.2: Representação dos diferentes níveis de amplitude utilizados para detetar os pulsos de emissão. A título ilustrativo, o sinal apresentado aqui, contém as primeiras trinta mil amostras adquiridas de um ficheiro.

Posto isto, estamos finalmente em condições de processar o sinal obtido. Inicia-se assim uma fase de processamento e manipulação dos dados segmentados, de forma a ter todos os A-Scans perfeitamente alinhados.

3.3 Processamento dos Sinais Ultrassónicos

O sinal recebido pelo sensor piezoelétrico é um sinal modulado pela frequência de ressonância do sensor, logo a informação útil (de banda base) é a envolvente do sinal. Assim, uma imagem ecográfica do tipo B-Scan é construída a partir da envolvente do sinal recebido, presente no ficheiro adquirido. Esta imagem representa uma imagem do olho a duas dimensões, ou seja, uma secção transversal, por isso é representada por uma matriz onde cada coluna corresponde aos ecos resultantes do pulso de emissão (um A-Scan) relativo a uma posição do sensor no varrimento linear, e o número total de colunas corresponde ao número de pulsos de emissão pretendidos (o varrimento efetuado pelo sensor). Logicamente, o número total de linhas em cada coluna está relacionado com a máxima profundidade que o utilizador pretende visualizar.

O processamento das amostras presentes no ficheiro adquirido é feito de forma cíclica. Isto significa que em cada ciclo de processamento são processadas as amostras relativas aos ecos resultantes da emissão de um pulso. Desta forma, o resultado do processamento dos vários ciclos

é usado para criar a matriz representativa da imagem. Salienta-se ainda que, em cada ciclo de processamento os dados são inicialmente convertidos de *short* para *double*[1].

Para determinar a envolvente do sinal ultrassónico adquirido deve ser usado um filtro de Hilbert. Como o sinal apresenta um *offset* introduzido pela placa presente no sistema de aquisição, efetuamos uma filtragem passa banda (secção 3.3.1) para remover esse *offset*, e só depois é que realizamos uma filtragem de Hilbert (secção 3.3.2). Definimos assim uma classe, denominada `Filter`, com métodos eficientes na realização destas filtragens.

Como referido no início deste capítulo, o utilizador tem controlo sobre determinados parâmetros que afetam o processamento dos dados, por isso apresenta-se de seguida uma descrição dos mesmos.

Profundidade inicial e final - estes parâmetros permitem ao utilizador escolher a profundidade inicial e final que pretende visualizar na imagem final da secção transversal (do cristalino). Para além de definirem o tamanho da matriz final (número total de linhas da matriz), estes parâmetros têm ainda influência na duração do processo de filtragem, visto que apenas filtramos as amostras relativas à diferença entre estas profundidades.

Percentagem do deslocamento efetuado pelo sensor - define a distância de varrimento do sensor que o utilizador pretende visualizar. Este valor percentual é sempre referenciado em relação à posição inicial do sensor. Essencialmente, este valor define o número de A-Scans que o utilizador pretende visualizar.

Resolução da imagem em altura e largura - no processo de criação da imagem final, estes parâmetros definem o tamanho da matriz a ser criada. Na ausência destes dois parâmetros, teríamos uma matriz final cujo número de colunas seria o número de pulsos de emissão presentes no ficheiro adquirido, e o número de linhas, o número de total de amostras num período de pulso. Com a presença destes parâmetros conseguimos controlar o tamanho dessa matriz, e assim, a resolução da imagem final. Isto é feito realizando médias nas linhas e colunas da matriz, de forma a que o seu tamanho seja o pretendido. O processo de realização de médias, bem como os algoritmos desenvolvidos para esse efeito, serão introduzidos na seção 3.3.3 deste capítulo.

[1]Nota: O programa para criação de imagens B-Scan desenvolvido, permite especificar o tipo das amostras, que serão processadas, como *doubles* (8 bytes) ou *floats* (4 bytes). Esta propriedade é definida num ficheiro criado única e exclusivamente para esse efeito.

Com tudo isto dito, iniciamos agora uma explicação detalhada sobre as diferentes etapas presentes no processamento dos dados adquiridos. Assim, em primeiro lugar, introduzimos a filtragem dos dados, começando pela filtragem passa banda, que é a primeira filtragem realizada neste processo.

3.3.1 Filtragem Passa Banda

Utilizamos um filtro passa banda, IIR (*Infinite Impulse Response*), para remover o *offset* referido. Este filtro é de ordem 8, e a sua função de transferência é constituída por 9 coeficientes de numerador e 9 de denominador, que aqui são representados respetivamente por b_n e a_n. O filtro apresenta-se assim como um filtro recursivo - saída do filtro é novamente utilizada à entrada. O módulo da resposta em frequência deste filtro pode ser visualizada na figura 3.3 (a).

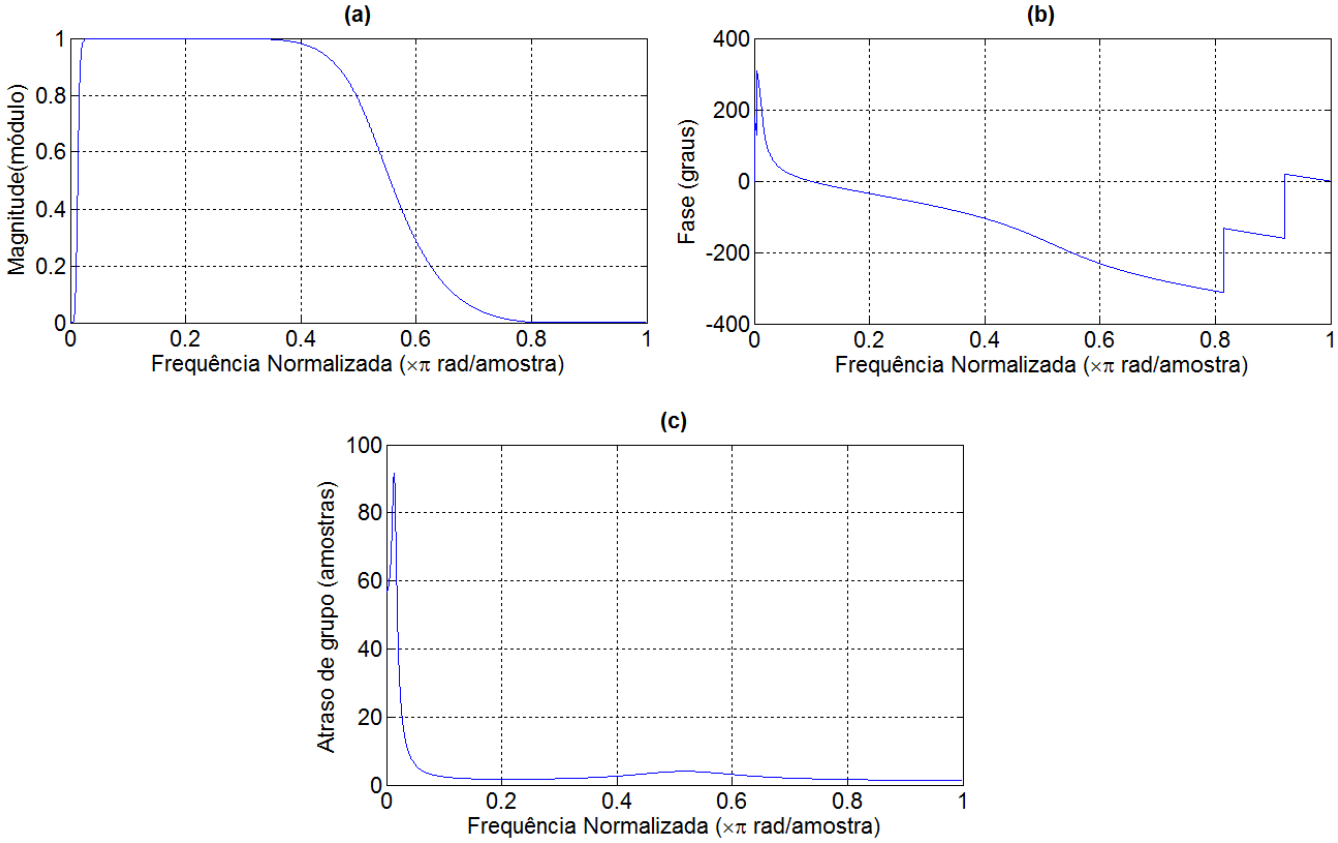


Figura 3.3: Representação do módulo (a) e da fase (b) da resposta em frequência do filtro passa banda. Em (c) está representado o atraso de grupo deste filtro.

Outra característica importante a salientar sobre este filtro, relaciona-se com o facto de este apresentar fase não linear o que poderá provocar um atraso de sensivelmente duas amostras (depende da frequência) no sinal resultante. Isto pode ser constatado ao analisarmos a fase

(figura 3.3 (b)) e o atraso de grupo (figura 3.3 (c)) deste filtro. Sabendo que o atraso de grupo representa o atraso provocado pelo filtro, isto é, a derivada da fase, pela sua análise verificamos que este não é linear e, consequentemente, o atraso de grupo não é constante (depende da frequência). Contudo, isto é pouco relevante para o nosso caso visto o atraso ser quase sempre inferior a 2 amostras, logo desprezamos esta situação.

Temos assim um sistema linear de processamento digital de sinal, capaz de realizar filtragens FIR (*Finite Impulse Response*) ou IIR. A figura 3.4 apresenta o diagrama de fluxo deste sistema.

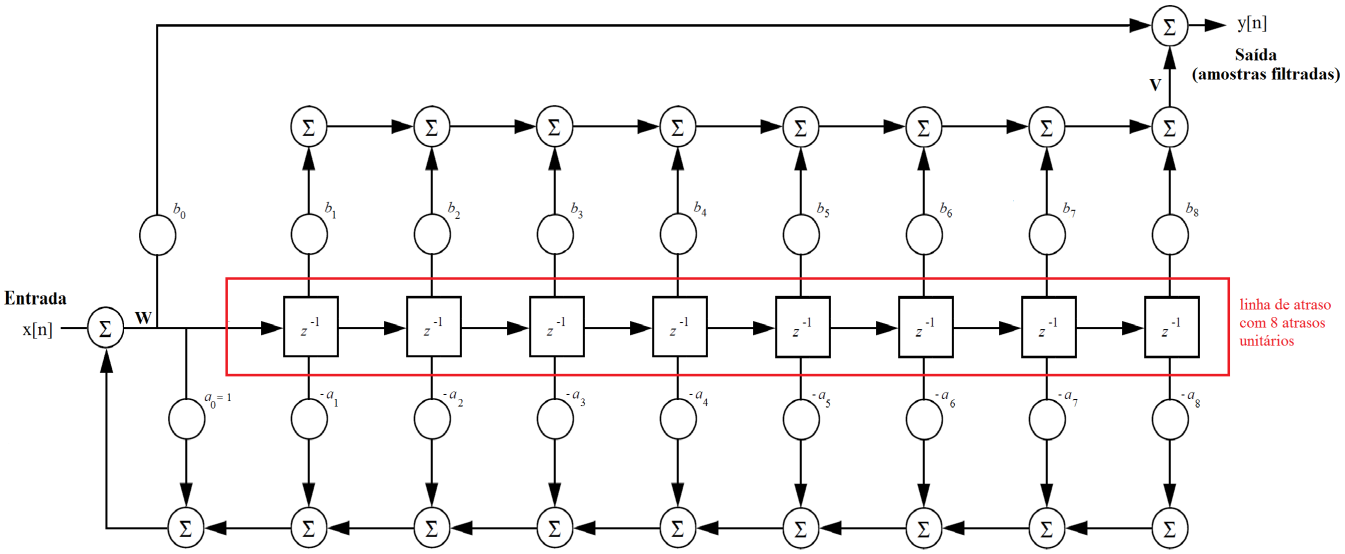


Figura 3.4: Diagrama de fluxo do sistema para filtragem IIR (permitem também filtragem FIR - $a_n = 0$). A sequência de amostras de entrada, x[n], é filtrada através da multiplicação entre os termos dessa sequência e os coeficientes do filtro (a_n e b_n). O resultados destas multiplicações é somado, produzindo assim a sequência y[n] na saída deste sistema. Neste diagrama, z^{-1} representa um atraso unitário.

Pela análise da figura 3.4, facilmente verificamos que:

$$w[n] = x[n] - a_1 \times x[n-1] - a_2 \times x[n-2] + (...) - a_8 \times x[n-8] \tag{3.1}$$

$$v[n] = x[n] + b_1 \times x[n-1] + b_2 \times x[n-2] + (...) + b_8 \times x[n-8] \tag{3.2}$$

$$y[n] = b_0 \times w[n] + v[n] \tag{3.3}$$

Ainda relativamente à figura 3.4, repare-se na existência de uma linha de atraso, isto é, uma sequência de atrasos unitários, em que cada unidade de atraso é um operador z^{-1} na transformada de Z. Ao desenvolver o algoritmo que implementa o funcionamento deste sistema, definiu-se um vetor representativo do estado do sistema, ou seja, a linha de atraso referida. Este vetor é

encarado como um *buffer* circular, e fornece as entradas atrasadas às operações de multiplicação. Assim, o estado do sistema é constantemente atualizado (recorre-se a uma *Look up Table* [2] para indexar os valores do vetor do estado, instaurando assim o princípio de funcionamento de um *buffer* circular).

3.3.2 Filtragem Hilbert

Para calcular a envolvente do sinal ultrassónico utilizou-se um filtro de Hilbert, FIR, com 99 coeficientes representando uma resposta a impulso de -49 a 49. É um filtro do tipo 3, cujos coeficientes são obtidos pelo método das janelas. Através dele pode-se realizar a chamada transformação de Hilbert, que é usada na representação de sinais passa-banda definidos com parte real e imaginária, de forma a gerar a componente imaginária do sinal dada a sua componente real (desviador de fase de 90°). Assim, utilizando o sinal resultante da filtragem passa banda (componente real) e o sinal resultante da filtragem com filtro Hilbert (componente imaginária) calcula-se a envolvente do sinal.

Analisemos agora o módulo e a fase da resposta em frequência deste filtro, apresentados nas figuras 3.5 (a) e (b), respetivamente. Rapidamente verificamos que a fase deste filtro é linear, logo temos uma distorção de fase constante, que se traduz num atraso constante para qualquer valor de frequência.

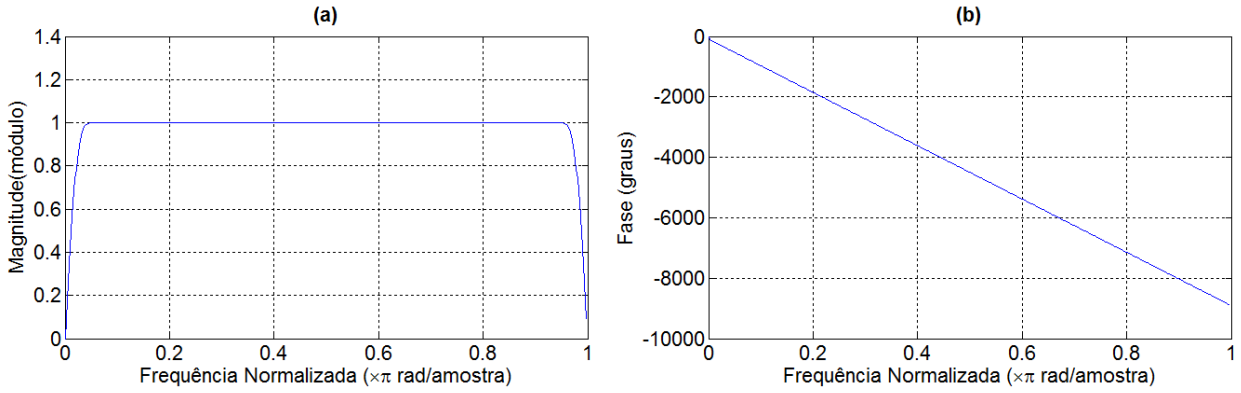


Figura 3.5: Representação do módulo (a) e da fase (b) da resposta em frequência do filtro de Hilbert.

Resumidamente, esta filtragem traduz-se numa convolução, e portanto devemos estar conscientes de que ao filtrarmos o sinal este fica atrasado em 49 amostras (metade da ordem do filtro), como podemos verificar pelo atraso de grupo deste filtro, representado na figura 3.6. É portanto

[2] *Look up Table* - vetor utilizado para realizar operações de indexação em outros vetores ou matrizes.

de extrema importância não nos esquecermos deste facto, sendo que ao efetuar qualquer cálculo computacional devemos tê-lo sempre em consideração.

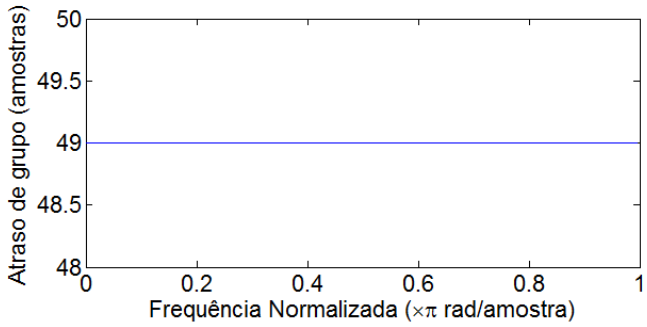


Figura 3.6: Representação do atraso de grupo do filtro de Hilbert.

Apresenta-se agora o diagrama de fluxo do sistema para filtragem FIR (figura 3.7).

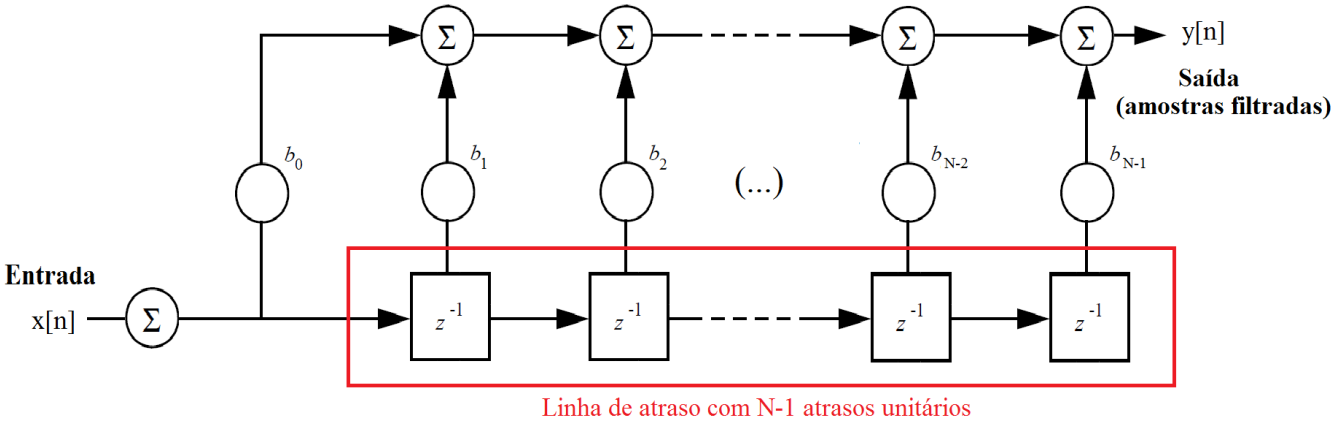


Figura 3.7: Diagrama de fluxo do sistema para filtragem FIR. A sequência de amostras de entrada, x[n], é filtrada através da multiplicação entre os termos dessa sequência e os coeficientes do filtro (b_n). O resultados destas multiplicações é somado, produzindo assim a sequência y[n] na saída deste sistema. Neste diagrama, z^{-1} representa um atraso unitário.

A saída do sistema apresentado na figura 3.7 é dada pela equação 3.4, onde N representa o comprimento do filtro de Hilbert utilizado.

$$\mathrm{y[n]} = b_0 \times x[n] + b_1 \times x[n-1] + b_2 \times x[n-2] + (...) + b_{N-1} \times x[n-(N-1)] \tag{3.4}$$

No desenvolvimento do algoritmo representativo desta filtragem, tivemos em conta outro aspeto relevante acerca deste filtro. Considerando que os 99 coeficientes deste filtro são representados por b_n, com $n = 0, ..., 98$, os coeficientes ímpares deste filtro são todos nulos e, por isso, o algoritmo é desenvolvido de forma a que se calculem apenas os resultados das operações que

derivam de multiplicações com coeficientes pares do filtro, reduzindo assim o número de operações matemáticas efetuadas e, consequentemente, o tempo gasto neste processamento, que é reduzido para metade. De notar que o primeiro coeficiente é representado por b_0, o segundo por b_1 e assim sucessivamente, sendo que b_0 denota um coeficiente par e b_1 um coeficiente ímpar. Assim, segundo esta perspetiva, a equação 3.4 é novamente definida, resultando a equação 3.5 que é aplicada no algoritmo responsável por realizar esta filtragem.

$$y[n] = b_0 \times x[n] + b_2 \times x[n-2] + b_4 \times x[n-4] + (...) + b_{N-1} \times x[n-(N-1)] \tag{3.5}$$

Tal como no caso da filtragem passa banda, na realização da filtragem de Hilbert recorre-se a um *buffer* circular para representar o estado do sistema, e a uma *Look up Table* para indexar os valores do vetor do estado.

3.3.3 Realização de Médias

A imagem B-Scan, criada por este programa, é uma imagem de pseudo-cores, isto é, cores representativas de determinada informação, neste caso das diferentes amplitudes dos sinais ultrassónicos recebidos. Como referido no início deste capítulo, existe uma matriz, resultante do processamento das amostras adquiridas, que representa a informação de cor desta imagem, sendo que o seu tamanho corresponde à resolução especificada pelo utilizador na entrada deste programa. Assim, procuramos desenvolver métodos rápidos e eficientes que, a partir do resultado da filtragem efetuada e posterior cálculo de envolvente, permitissem criar uma matriz, cujo tamanho é definido pelo utilizador.

O processo para a criação de uma matriz com tamanho específico é realizado fazendo médias das amostras da envolvente calculada. Contudo é preciso saber como é que estas médias são realizadas, e por isso apresentamos de seguida uma breve descrição deste processo.

Inicialmente, para obtermos uma matriz com a largura desejada, criamos uma classe que opera segundo o princípio de funcionamento de um *buffer* circular, a qual denominamos `CircularBuffer`. A ideia principal consiste em realizar médias do conteúdo do *buffer* desta classe, que é circularmente preenchido com vetores representativos da envolvente de cada A-Scan, e cujas amostras inicial e final estão associadas às profundidades especificadas pelo utilizador. Criamos assim um *buffer* que é encarado como uma matriz de tamanho fixo, cujas colunas são preenchidas circularmente. Com o *buffer* devidamente preenchido, são realizadas médias das linhas desta matriz, resultando um vetor coluna cujo número total de elementos correspondente à diferença

(em amostras) entre a profundidade final e inicial especificadas pelo utilizador. Posto isto, falta apenas realizar médias ao longo do vetor coluna criado, tendo em conta a resolução em altura pretendida. Para isso são realizadas novas médias sobre esse vetor coluna, criando finalmente uma matriz com um tamanho correspondente à resolução especificada pelo utilizador.

Para que se possa compreender melhor esta estrutura de dados implementada, e como ou quando são realizadas as referidas médias, apresentamos de seguida uma explicação mais detalhada sobre estes processos.

Buffer Circular

Como referido, a classe `CircularBuffer` permite a realização de médias recorrendo a um *buffer* circular. O código desenvolvido pretende-se rápido e eficiente na utilização deste conceito, e portanto esta classe foi projetada com vista a não ter grande significado no tempo de execução deste programa. De seguida apresentamos a tabela 3.1, que contém os membros e métodos utilizados por esta classe. Posteriormente, apresentamos também a figura 3.8 que ilustra a estrutura de dados relativa ao funcionamento do *buffer* circular.

Tabela 3.1: Membros e métodos da classe `CircularBuffer`.

Membros	
`pointers`	Ponteiro para ponteiro de um vetor coluna do *buffer* circular.
`data`	Ponteiro para vetor coluna do *buffer* circular.
`N`	Número de elementos do *buffer* circular (número de vetores coluna).
`size`	Tamanho de cada vetor coluna do *buffer* circular.
`i_atual`	Índice atual do *buffer* onde se insere o vetor coluna.
Métodos	
Construtor	Alocação de memória e inicialização de variáveis.
Destrutor	Libertação de memória e *reset* ao índice do *buffer* circular.
`insert`	Insere um vetor coluna no Buffer circular. Este vetor tem tamanho correspondente à diferença entre a profundidade inicial e final especificadas pelo utilizador, em amostras.
`meanX`	Faz médias das linhas dos N vetores do *buffer* circular (médias na direção horizontal).

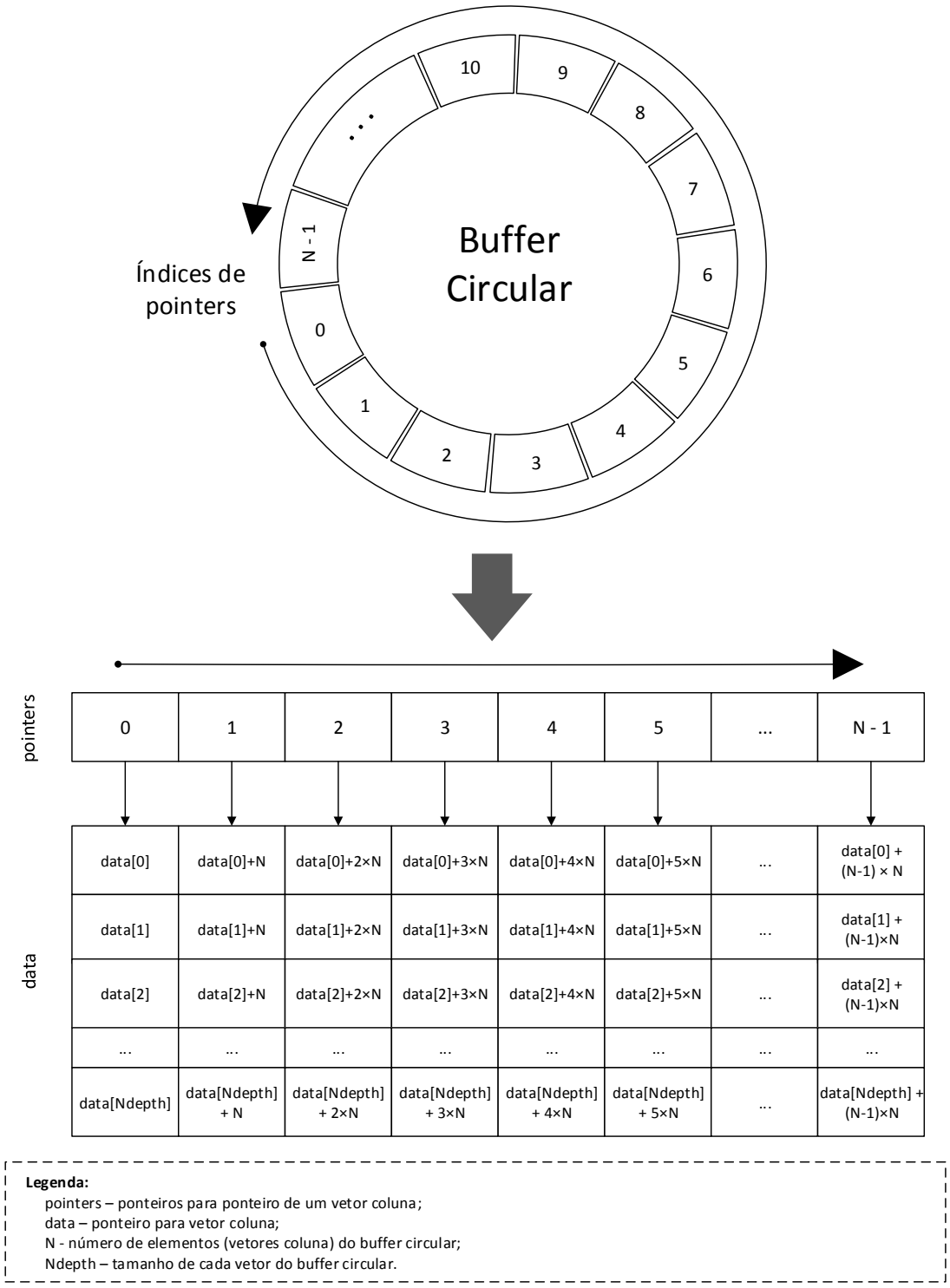


Figura 3.8: Estrutura de dados do *buffer* circular, que pertence à classe `CircularBuffer`, utilizado para a realização de médias na direção horizontal.

Pela análise da informação exposta na figura referida e na tabela 3.1, facilmente compreendemos o funcionamento do *buffer* circular e a estrutura de dados da classe desenvolvida. Após a extração da envolvente, inserem-se vetores no *buffer* e incrementa-se o seu índice de posição (`i_atual`) para cada vetor inserido. Assim que é atingido o número máximo de elementos do *buffer* (`N`) é efetuado um *reset* ao seu índice (`i_atual`= 0) e os novos vetores coluna a inserir são colocados sobre os anteriores, isto é, preenche-se o *buffer* de forma circular. A sequência dos `N` vetores inseridos no buffer pode ser encarada como uma matriz, em que a primeira coluna está associada ao índice 0 do *buffer* e a última ao índice `N`-1, e assim são realizadas médias das linhas dessa matriz (médias na direção horizontal da matriz), obtendo como resultado deste processo um vetor coluna final. Após estas médias serem realizadas, falta ainda realizar as médias ao longo dos elementos deste vetor obtido (médias na direção vertical). O método responsável pela realização destas últimas médias é exterior à classe `CircularBuffer`. Concluído este processo, o vetor resultante é então inserido numa coluna da matriz final.

O número de elementos do *buffer* circular e o momento em que as sucessivas médias são realizadas dependem da resolução especificada pelo utilizador. De seguida explicaremos como é que estas médias são realizadas.

Realização de Médias na Vertical e na Horizontal

Como já foi referido, após o processo de extração de envolvente sucede-se o preenchimento de um elemento do *buffer* circular, sendo este processo ciclicamente repetido para todos os A-Scans (contêm os ecos de um pulso de emissão). À medida que este processo decorre, são realizadas médias em determinados momentos. Para percebermos como e quando é que estas médias são realizadas definimos um conjunto de variáveis:

$\boldsymbol{N_{scans}}$ - Número total de A-Scans;
$\boldsymbol{N_b}$ - Número total de elementos do *buffer* circular;
$\boldsymbol{N_{depth}}$ - Número total de amostras de cada A-Scan;
$\boldsymbol{N_x}$ - A resolução em largura especificada pelo utilizador;
$\boldsymbol{N_y}$ - A resolução em altura especificada pelo utilizador.

Por sua vez, estas variáveis definem as seguintes dimensões (altura × largura):

Dimensão do *buffer* circular: $\boldsymbol{N_{depth}} \times \boldsymbol{N_b}$;
Dimensão da matriz de envolventes: $\boldsymbol{N_{depth}} \times \boldsymbol{N_{scans}}$;

Dimensão da imagem de pseudo-cores a criar: $\boldsymbol{N_y \times N_x}$;

As médias são realizadas com o objetivo de criar uma imagem, com a dimensão pretendida ($N_y \times N_x$), a partir da matriz de envolventes, cuja dimensão é dada por $N_{depth} \times N_{scans}$. Assim é necessário fazer médias nas direções horizontal e vertical dessa matriz.

Suponhamos um exemplo em que temos 30 A-Scans no total, ou seja, $N_{scans} = 30$, e que cada um destes A-Scans tem 1000 pontos ($N_{depth} = 1000$). Em contrapartida, queremos uma imagem de 50 × 4 pixels de resolução ($N_y = 50$ e $N_x = 4$). Posto isto, realizamos 4 médias ao longo dos 30 A-Scans, resultando uma matriz de 1000 × 4 amostras, e seguidamente realizamos 50 médias nas 1000 linhas desta matriz.

Para além disto, definimos o comprimento e o avanço da janela de médias, de forma a que existisse uma sobreposição nas médias realizadas, evitando assim um possível efeito de blocos na imagem final. Para o caso, consideramos realizar médias com 33% de sobreposição. Vejamos em primeiro lugar o caso das médias realizadas na horizontal.

Considerando então a sobreposição de 33%, o comprimento do *buffer* circular onde são realizadas as médias horizontais, corresponde ao comprimento da janela de médias na horizontal, e calcula-se como demonstrado na equação 3.6. Nesta equação, Δ representa o avanço da janela de médias ao longo dos vários A-Scans, ou seja, na direção horizontal da matriz de envolventes, e é definido na equação 3.7.

$$N_b = \lfloor 1.5 \times \Delta \rfloor \tag{3.6}$$

$$\Delta = \frac{N_{scans}}{N_x} \tag{3.7}$$

Na figura 3.9 é representado o avanço efetuado pela janela de médias na horizontal.

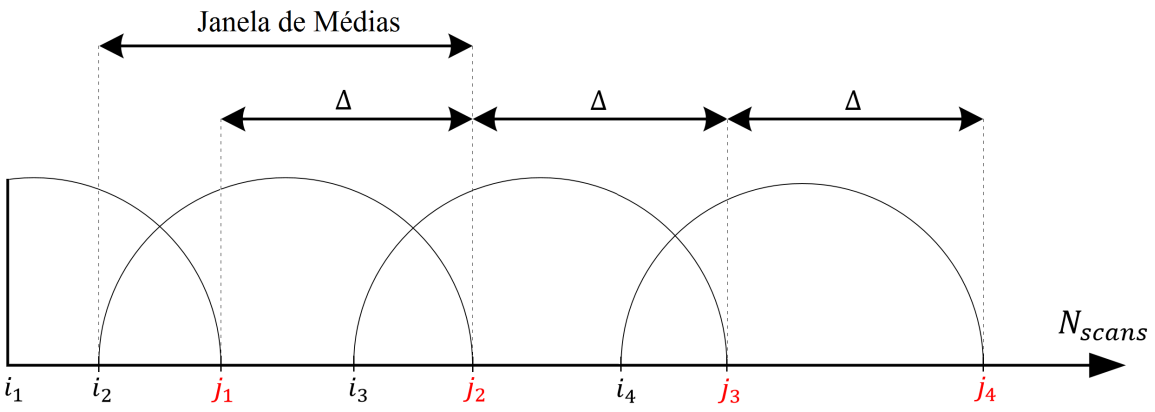


Figura 3.9: Gráfico representativo do processo de avanço da janela de médias na horizontal.

Analisando a figura 3.9, conseguimos determinar o momento em que as médias são realizadas. Sabendo que o comprimento de uma janela de médias é $N_b = 11$ e que esta janela faz sucessivos avanços de $\Delta = 7,5$ A-Scans, então os momentos em que as médias horizontais são realizadas, correspondem aos avanços das janelas. Posto isto, definimos então as seguintes variáveis:

$$i_k = \max(0, j_k - N_b + 1) \qquad k = 1, ..., Res_x \tag{3.8}$$

$$j_k = \lfloor k \times \Delta \rfloor - 1 \qquad k = 1, ..., Res_x \tag{3.9}$$

$$l_k = j_k \% N_b \qquad k = 1, ..., Res_x \tag{3.10}$$

Nas equações anteriores, i_k e j_k representam, respetivamente, os índices do primeiro e do último A-Scan em cada janela de médias, como é ilustrado na figura 3.9. Por sua vez os índices l_k dizem respeito ao fim da janela k no *buffer* circular, ou seja, aos índices do *buffer* circular que correspondem aos índices dos fins da janela de médias (j_k).

Para uma melhor compreensão deste processo, observe a figura 3.10, que demonstra a realização de médias na horizontal para o exemplo em questão ($N_{scans} = 30$, $N_x = 4$, $\Delta = 7,5$ e $N_b = 11$).

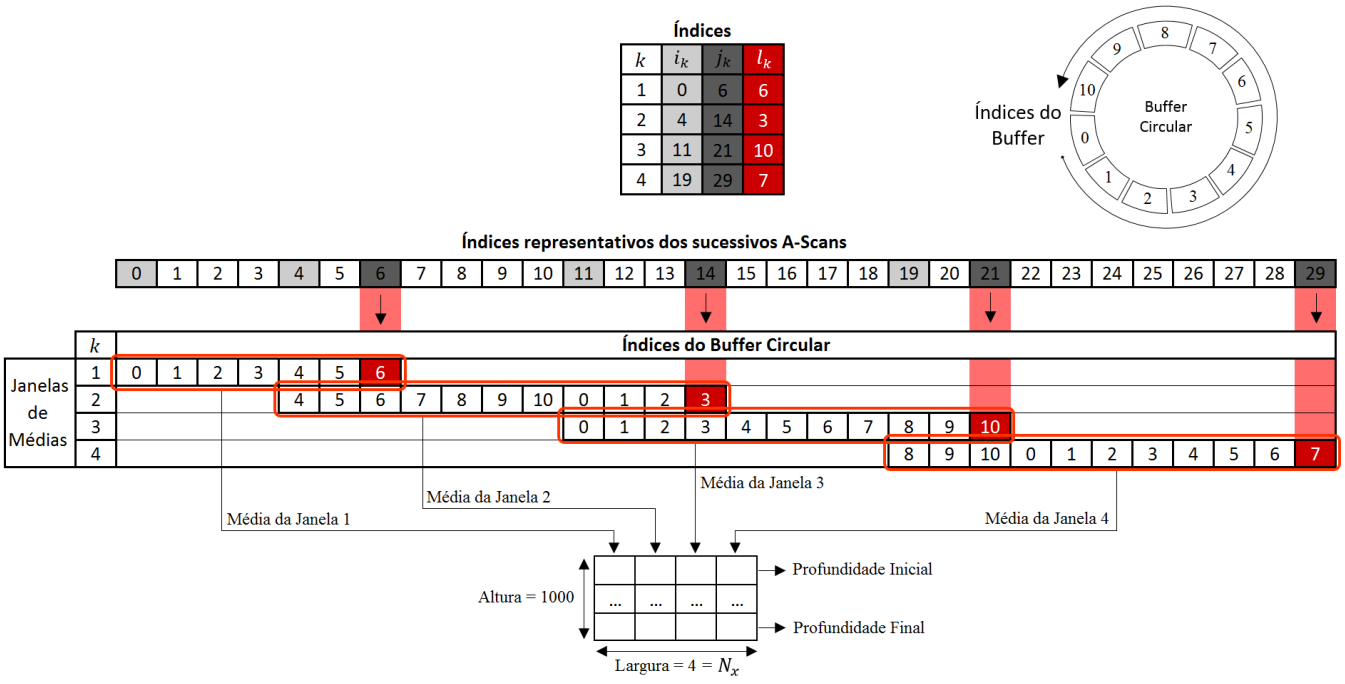


Figura 3.10: Exemplo ilustrativo da realização de médias na horizontal.

Finalizado o processo ilustrado na figura 3.10, em que realizamos 4 médias ao longo dos 30 A-Scans, resulta uma matriz de 1000 × 4 amostras. Assim, faltam apenas realizar 50 médias nas 1000 linhas desta matriz (na direção vertical). Como referido anteriormente, esta última

operação é exterior à classe `CircularBuffer`, mas a ideologia presente no processo de realização de médias verticais é igual em relação às médias horizontais que acabamos de descrever. Em concreto, como queremos uma imagem final com 50 pixels de altura ($N_y = 50$), é necessário realizar 50 médias sobre as amostras de uma coluna da matriz de envolventes, que tem $N_{depth} = 1000$ linhas. Então, seguindo a estratégia de realizar médias com uma sobreposição de 33%, o comprimento da janela de médias utilizada, que aqui designamos por N, e o avanço dessa janela (Δ), são definidos segundo as próximas equações.

$$N = \lfloor 1.5 \times \Delta \rfloor \tag{3.11}$$

$$\Delta = \frac{N_{depth}}{N_y} \tag{3.12}$$

Sabendo N e Δ, podemos então determinar os momentos em que as médias na direção vertical da matriz de envolventes são realizadas. Para isso recorremos às equações 3.8 (considerar $N_b = N$) e 3.9, anteriormente definidas. Desta forma, sabendo os inícios e os fins de cada janela de médias ao longo de uma coluna da matriz de envolventes, realizamos médias na direção vertical. No final obtemos uma matriz representativa da imagem de pseudo-cores a criar, cuja dimensão é de 50 $\times$ 4 pixels ($N_y = 50 \times N_x = 4$).

3.4 A Matriz Final da Imagem

Para construir uma matriz que contenha a informação de cor da imagem ecográfica pretendida, criamos uma classe denominada `Matrix` que reserva memória para essa matriz e incorpora métodos que permitem inserir e obter elementos da mesma. Consideramos que 256 cores (pseudo-cores) seriam suficientes para a representação da imagem ecográfica desejada, por isso, as amostras do tipo *double* resultantes do processamento efetuado são convertidas para o tipo de dados *unsigned char* ("uint8"). Assim, os valores desta matriz estarão compreendidos no intervalo de 0 a 255.

Para que a conversão dos dados seja adequada, aplica-se uma fórmula que faz a escalação dos dados para os 256 valores possíveis. Esta fórmula consiste essencialmente na aplicação do logaritmo natural aos dados (a amplitude da envolvente), criando assim uma escala logarítmica representativa das 256 cores que a imagem terá. Só depois de aplicada esta fórmula é que é feita a conversão do tipo de dados, de *double* para *unsigned char*. Assim, à medida que os ecos recebidos, provenientes do ficheiro adquirido, são processados, as colunas da matriz são preenchidas até que

no fim tenhamos uma matriz cujo número de linhas e colunas correspondem respetivamente à resolução em altura e largura especificadas pelo utilizador.

Com a matriz construída verificamos que existia um problema relativo aos valores das primeiras linhas e colunas dessa mesma matriz. Este problema estava associado ao processo de realização de médias. Na maior parte dos casos, a primeira janela de médias não é completamente preenchida, como acontece no exemplo da figura 3.9, por isso, visto que a janela de médias tem um comprimento fixo, a primeira média realizada contém elementos nulos e elementos de interesse, logo o seu resultado pode ser significativamente diferente em relação à segunda média realizada (a janela completamente preenchida, contendo apenas amostras de envolvente). Concretamente, este problema traduz-se numa diferença de cor significativa entre os primeiros pixels nas direções vertical e horizontal da imagem e os restantes pixels da imagem. Para o resolver, acrescentamos um método à classe `Matrix` que repete as primeiras linhas e colunas da matriz de forma a que este problema não seja percetível quando olhamos para a imagem final.

Finalmente, este conjunto de dados final constitui a informação necessária para representar a imagem ecográfica do tipo B-Scan desejada. Contudo, falta-nos determinar alguns dados que permitam ao utilizador da aplicação Android medir distâncias sobre a imagem que está a visualizar.

3.5 Determinação de Dados Biométricos

Iremos agora indicar quais os dados necessários para uma análise biométrica da imagem, ou seja, os dados que permitem ao utilizador da aplicação medir distâncias entre as diferentes estruturas do olho apresentado na imagem ecográfica. Assim, explicaremos como podem ser determinados os valores, em milímetros, da largura e altura de 1 pixel da imagem final, que sendo enviados para a aplicação, permitem que o utilizador realize medições sobre a imagem.

Em primeiro lugar determinemos como se pode calcular a altura de 1 pixel. Sabe-se que a distância percorrida por uma onda ultrassónica calcula-se pelo produto entre a velocidade de propagação dessa onda, v, e o tempo que ela demora a percorrer essa distância, t. Considerando que a onda tem uma viagem de ida e volta, utilizamos então a equação 2.2 (capítulo 2, secção 2.1) para determinar a profundidade atingida pela onda. Uma vez que já sabemos a frequência de amostragem (f_s) do sinal adquirido e a velocidade de propagação das ondas, a partir da equação

2.2 obtemos a equação 3.13 que permite calcular a altura relativa a 1 pixel em milímetros.

$$\text{Altura Pixel} = \frac{v}{f_s} \times \frac{1000}{2} \text{ [milímetros]} \tag{3.13}$$

Posto isto, a altura total da imagem em milímetros é facilmente determinada com o número total de elementos de uma coluna da matriz final.

Sabendo que a sonda ultrassónica se desloca a v_d milímetros por segundo, que o PRF define o número de pulsos que são emitidos num segundo, que o número total de A-Scans que o utilizador pretende são N_{scans}, e que a largura em pixels da imagem final é Res_x, então a equação 3.14 permite-nos calcular a largura de 1 pixel.

$$\text{Largura Pixel} = \frac{v_d \times N_{scans}}{PRF \times Res_x} \text{ [milímetros]} \tag{3.14}$$

Sabendo a largura de 1 pixel facilmente calculamos a altura total da imagem (produto entre o número total de colunas da matriz e a largura de 1 pixel).

3.6 Envio da Imagem Ecográfica para o Cliente

O cliente (a aplicação Android) requesita os dados para a visualização de uma imagem B-Scan e, por isso, desencadeia o processo de análise dos dados ecográficos, conforme foi indicado nas secções anteriores. Os dados que constituem a imagem produzida são então enviados em formato binário para um *stream* de saída, e são lidos pela aplicação presente no servidor, que encarregar-se-á de os enviar para a o dispositivo Android. Em particular, estes dados são enviados segundo uma determinada estrutura: um cabeçalho seguido da imagem (matriz final). Em concreto, o cabeçalho contém os dados biométricos e outros dados relativos à aquisição dos sinais ultrassónicos, e os dados que se seguem constituem a matriz final representativa da imagem ecográfica.

No próximo capítulo iremos discutir como é feita a chamada do processo responsável pela execução deste programa e a transferência de dados efetuada entre o servidor e a aplicação.

Capítulo 4

Arquitetura Servidor Cliente

Neste capítulo pretende-se descrever o processo de comunicação entre o servidor e o cliente. O servidor simula o instrumento ecográfico e o cliente constitui o visualizador que contém a interface para interação com o utilizador (aplicação Android). A comunicação entre ambos é realizada através da Internet (TCP/IP), utilizando o protocolo HTTP numa aplicação Web ASP.NET.

4.1 ASP.NET

Como descrito em [21], ASP.NET é a solução atual da Microsoft para o desenvolvimento de páginas Web dinâmicas do lado do servidor, e é a sucessora da tecnologia ASP(*Active Server Pages*) clássica. Foi desenvolvida para permitir a construção de sites, aplicações e serviços para a Web. No presente caso, utilizando a linguagem C#, desenvolveu-se uma aplicação Web que realiza a comunicação entre servidor e cliente. Esta aplicação é executada no servidor Web IIS(*Internet Information Services*), criado pela Microsoft para o SO Windows.

Para o desenvolvimento da aplicação Web pretendida, recorreu-se a uma técnica introduzida pela ASP.NET. Utilizou-se um *Handler* HTTP, que se define como sendo "um processo que corre em resposta a uma requisição feita a uma aplicação Web ASP.NET", [16]. Basicamente, este *Handler* consiste num ficheiro de código (no presente caso escrito em C#) que escreve dados para a resposta HTTP do servidor.

4.2 A Aplicação Web

O princípio básico de comunicação entre cliente e servidor, consiste no envio de respostas do servidor face a requisições HTTP efetuadas pelo cliente. Estas mensagens de resposta são enviadas do servidor para o cliente utilizando o protocolo de transporte TCP, [13].

As requisições HTTP são efetuadas pela aplicação Android, utilizando o método GET deste protocolo. Como resposta a estas requisições o servidor envia os dados que permitem a visualização das imagens ecográficas tipo A-Scan e B-Scan. No caso de imagens B-Scan o servidor (instrumento) analisa o ficheiro ecográfico produzido e cria a imagem binária com a informação das pseudo-cores que envia para o cliente. No caso de imagens A-Scan, é enviado apenas o último ficheiro produzido (com nome invariável). Apresentamos na figura 4.1 uma descrição de todo este processo de comunicação.

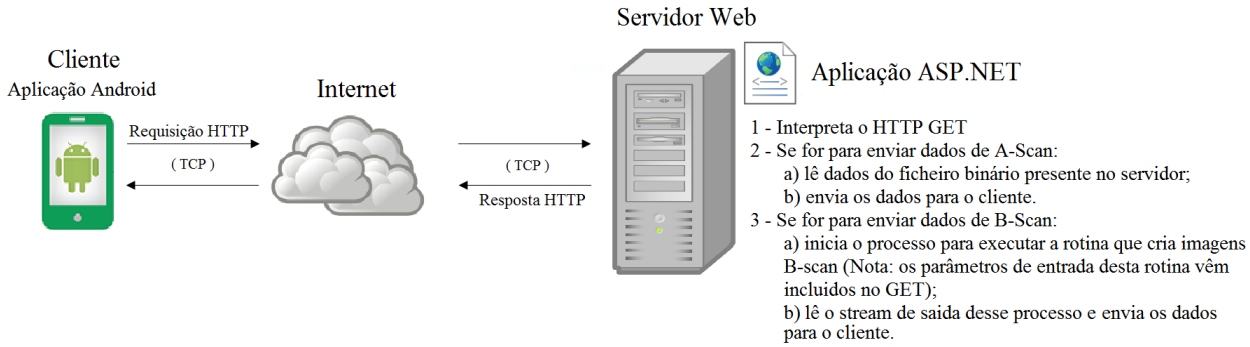


Figura 4.1: Comunicação entre o servidor Web e a aplicação Android.

Numa aplicação mais perto do real, a requisição do cliente deveria provocar uma nova aquisição A-Scan ou B-Scan. Posteriormente, só depois de os dados estarem processados é que haveria a respetiva resposta.

Capítulo 5

A Aplicação Android

Neste capítulo explicamos como foi desenvolvida uma aplicação para o SO Android, na linguagem de programação Java, que permite a visualização de imagens ecográficas do tipo A-Scan e B-Scan. Para além disto apresentaremos também os métodos e algoritmos desenvolvidos que permitem a realização de medições de distâncias nas imagens ecográficas visualizadas.

5.1 A Interface Gráfica

Quando se desenvolve uma aplicação Android, cria-se uma atividade (capitulo 2, secção 2.2.3), que por sua vez fornece um ecrã com o qual o utilizador pode interagir. Assim, após a criação desta atividade inicia-se a construção da interface gráfica. Como descrito em [6], a interface gráfica para o utilizador é construída usando uma hierarquia de objetos `View` e `ViewGroup`. Um objeto `View` poderá ser um botão ou um campo de texto, enquanto que um objeto `ViewGroup` é um "contentor"de `Views`. O objeto `ViewGroup` é invisível e define a organização dos objetos `View`, que pode ser em grelha, numa lista vertical, etc. Tudo isto é feito num ficheiro XML. Definimos então dois `ViewGroups`, um para a tela onde serão desenhadas as imagens ecográficas e outro para colocar todos os botões necessários. O primeiro contém um objeto `View` para colocar a tela de desenho, e o segundo incorpora seis `Views`, sendo que cada um destes contém um botão. Posto isto, ilustramos na figura 5.1 a interface gráfica desenvolvida.

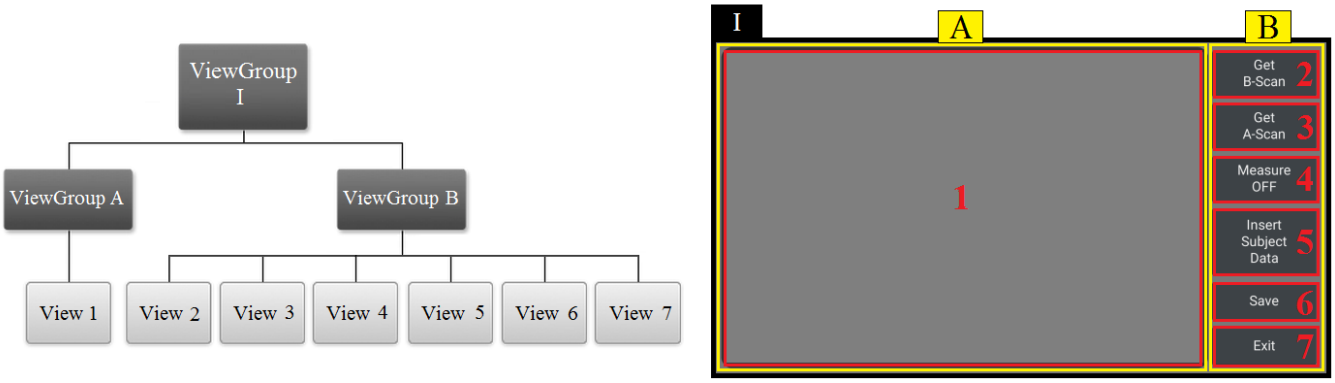


Figura 5.1: Interface gráfica da aplicação.

Tendo em conta a enumeração ilustrada na figura 5.1, explicaremos agora o propósito de cada um dos botões criados. O botão 2 permite visualização de imagens ecográficas do tipo B-Scan, por outro lado o botão 3 serve para a visualização imagens tipo A-Scan. O botão 4 serve para a realização de medidas sobre as imagens visualizadas, em concreto medidas de distâncias. O botão 5 serve para inserir alguma informação sobre o sujeito relativo à imagem que está a ser visualizada (só é permitida a sua utilização no modo B-Scan). O botão 6 permite salvar a imagem que está a ser visualizada para a galeria do dispositivo e finalmente o botão 7 serve para sair da aplicação.

Desenhada a interface gráfica, é necessário fornecer operabilidade a esta interface, programando as funcionalidades dos diferentes botões. Assim introduzimos de seguida as diferentes etapas que permitem a visualização de uma imagem ecográfica.

5.2 Transferência, Construção e Desenho das Imagens

5.2.1 Transferência dos Dados

Inicialmente criamos a classe `MainActivity` que define a atividade que é iniciada quanto o utilizador inicia a aplicação. Poderá dizer-se que ela corresponde ao método `Main()` desta aplicação. É nela que definimos as funções relativas às ações de cada um dos botões presentes na interface gráfica anteriormente exposta. Como foi ilustrado na figura 5.1, os botões 2 e 3 correspondem aos modos de visualização de imagens ecográfica do tipo B-Scan e A-Scan respetivamente. No seu ponto de origem o processo para a visualização destas imagens pressupõe a transferência dos dados que as representam. Assim para realizar essa transferência de dados criamos uma classe à

qual chamamos `ImageDownloaderTask`. Esta classe estende a classe `AsyncTask` que é executada numa *thread* que não a *thread* principal(UI *thread* - *User Interface thread*), permitindo assim a realização de uma computação assíncrona em relação à *thread* principal, garantindo uma redução do esforço realizado por esta última ([4], [1]).

O processo para transferência de dados correspondente à ação dos botões 2 e 3 (figura 5.1) é diferente, no que diz respeito ao pedido feito pela aplicação ao servidor e aos dados que são recebidos. Como enunciado no capítulo 4 , seção 4.2, a transferência de dados é realizada após uma requisição HTTP efetuada pela aplicação, que concretamente consiste no envio de um GET onde se define o tipo de imagem pretendida (A-Scan ou B-Scan) e caso esta seja B-Scan define-se ainda os parâmetros de entrada (profundidade inicial e final, percentagem de varrimento, largura e altura da imagem) para a execução do programa que fornece os dados desta imagem. É de salientar que para a definição dos parâmetros de entrada do modo B-Scan construímos uma janela secundária que permite ao utilizador introduzir os respetivos valores. Esta janela apresenta-se visível quando o botão 2 (imagem B-Scan) é pressionado. Posteriormente estes parâmetros são utilizados para a construção de uma *string* que constitui o pedido a enviar ao servidor. Para a aplicação receber a resposta associada a esse pedido, é criado um objeto `inputStream`, que cria um *stream* de entrada para a transferência dos dados (em formato binário), que ao serem lidos ficam armazenados em memória num vetor cujos elementos são do tipo *byte*. Este processo é realizado por um método da classe `ImageDownloaderTask`, que tem como parâmetro de entrada uma *string* com o URL relativo à requisição HTTP efetuada e que retorna um vetor do tipo *byte* com os dados recebidos.

Importa ainda referir que os dados transferidos dividem-se em 2 partes: os dados iniciais constituem o cabeçalho e a seguir aparece o conjunto de dados representativos da imagem. Para o caso das imagens B-Scan o cabeçalho tem um tamanho de 130 bytes. Nele estão informações relativas ao processo de aquisição dos dados, que serão expostas na imagem final apresentada, à largura e altura de um pixel (em milímetros) e ao número de linhas e colunas da matriz referida no capítulo 3, secção 3.4. Por outro lado, no caso das imagens A-Scan o cabeçalho tem um tamanho total de 256 bytes e contém informações como por exemplo, o número total de amostras adquiridas por milímetro, algumas características do sinal recebido, a localização de algumas estruturas do olho e ainda informação relativa à presença e respetiva classificação de cataratas.

5.2.2 Construção das Imagens

Transferidos os dados, são necessárias ainda algumas operações para que possamos desenhar as imagens que eles representam. Assim diferenciamos aqui o processo para a construção das imagens B-Scan e A-Scan.

Imagens B-Scan

Inicialmente referimos que a imagem construída teria 256 cores (capítulo 3, seção 3.4). Contudo, após alguns testes verificamos que se construíssemos uma imagem com 128 cores a diferença seria impercetível, pelo menos para o olho humano. Assim decidimos representar a imagem com 128 cores. Para tal é necessário converter os dados para uma representação inteira de cor, isto é, para o tipo *int*. Para isso criamos um vetor de inteiros que representa a imagem, e um mapa (conhecido como JET) de 128 cores (pixels com 8 bits para cada canal de cor: R,G,B) que contém as cores para a imagem. Em concreto este mapa de cores funciona como uma *Look Up Table*, que permite atribuir cores aos elementos do vetor de inteiros. Como inicialmente os dados se apresentam em formato binário e têm um valor inteiro compreendido entre 0 e 255, é necessário fazer uma escalação destes dados para os 128 valores de cor possíveis. Para isso determinamos o valor máximo (V_{max}) e mínimo (V_{min}) dos dados recebidos, e de seguida mapeamos esses valores para um intervalo de 0 a 128, segundo a equação 5.1. D_i representa os dados recebidos, com $i = 0, ..., N-1$ e $N = largura \times altura$ da imagem.

$$\text{Índice da LUT} = \frac{127 \times D_i - 127 \times V_{min}}{V_{max} - V_{min}} \tag{5.1}$$

Esta equação é aplicada a cada elemento do vetor dos dados transferidos, e o seu resultado fornece o índice da *Look Up Table* que contém a cor a ser atribuída ao elemento do vetor de inteiros. Finalizado este processo, o resultado é um vetor com a informação de cor da imagem (inteiros de 64 bits), cujo número total de elementos corresponde ao produto entre a resolução em altura e largura inserida pelo utilizador. Por sua vez, este vetor de inteiros é utilizado para construir um objeto do tipo `Bitmap`, que basicamente é um formato de imagem que contém a descrição da cor de cada pixel. É de notar que na criação deste objeto `Bitmap` é necessário definir a resolução de bits para cada canal de cor e por isso, utilizando o modelo de cores RGB, de entre as configurações possíveis, escolhemos aquela que ocupasse menos espaço em memória: 5 bits para o vermelho, 5 bits para o azul e 6 bits para o verde, [5].

Imagens A-Scan

O processo de construção de uma imagem A-Scan é totalmente diferente do anterior. Uma vez que esta imagem é uma representação das diferentes amplitudes recebidas ao longo do tempo/profundidade, a sua representação é feita através de um gráfico de linha. Os dados têm 16 bits por amostra e, por isso, essas amostras podem assumir valores inteiros entre -32768 e $32,767$, inclusive. Começamos então por determinar o máximo e o mínimo valor de amplitude das amostras recebidas, para posteriormente podermos definir o limite de desenho deste gráfico.

Posteriormente, construímos um vetor que permite desenhar retas entre pontos, com o objetivo de fazer o desenho do gráfico. Para melhor perceber como esse vetor é construído considere que temos 4 pontos cujas coordenadas são: (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4). Assim, definimos o vetor $v = [x_1, y_1, x_2, y_2, x_2, y_2, x_3, y_3, x_3, y_3, x_4, y_4]$ que contém as coordenadas iniciais e finais de cada reta que será desenhada entre os sucessivos pontos.

5.2.3 Desenho das Imagens

Como referido em [15] existem diferentes possibilidades para desenhar imagens ou objetos numa aplicação. No nosso caso, para desenhar as imagens ecográficas criamos a classe `DrawingView` que estende a classe `View`. Esta última classe é responsável pelo desenho de eventos no retângulo associado ao objeto `View` 1, presente na figura 5.1 anteriormente apresentada. Para que isso seja possível esta classe incorpora o método `onDraw()`, que define o que queremos desenhar. Finalmente o desenho em si é tratada pela classe `Canvas` (tela para pintura). O método `onDraw()` é chamado quando o `View` precisa de fazer um desenho. A esse método é então passado um objeto `Canvas` que contém o `Bitmap` que será colocado nesse `View`. Por sua vez o `Canvas` fornece os métodos necessários para fazer as operações básicas de desenho, e esse desenho é efetuado no seu `Bitmap` interno. Relativamente ao processo de desenho em si são precisos 4 componentes: um `Bitmap` onde o desenho será feito (contém a descrição dos pixels; por defeito é fornecido pelo sistema, mas podemos ser nós a criá-lo), um `Canvas` que vai escrever no `Bitmap`, uma primitiva de desenho (por exemplo um `Bitmap`, texto, linha, formas geométricas, etc.), e um objeto `Paint` que descreve as cores e o estilo da pintura.

Em concreto ao efetuarmos o desenho das imagens tipo A-Scan e B-Scan a diferença está na primitiva de desenho utilizada. Para desenharmos a imagem B-Scan utilizamos o `Bitmap` criado anteriormente que é passado para o método `drawBitmap()` do `Canvas`. Por outro lado,

no caso da imagem A-Scan utilizamos um método do `Canvas`, denominado `drawLines()`, que recebe o vetor que contém os pontos do início e fim das retas que serão desenhadas. Finalizado este processo é apresentada uma imagem no ecrã do dispositivo. Contudo, no intuito de fornecer ao utilizador uma análise mais detalhada sobre a imagem, permitimos que ele a possa deslocar, aumentar ou diminuir. É neste contexto que introduzimos de seguida os conceitos relacionados com transformações geométricas.

5.3 Transformações Geométricas

Para perceber como são feitas transformações geométricas sobre as imagens começamos por apresentar na figura 5.2 o sistema de coordenadas que é utilizado pela classe `Canvas` para o desenho de imagens na tela respetiva.

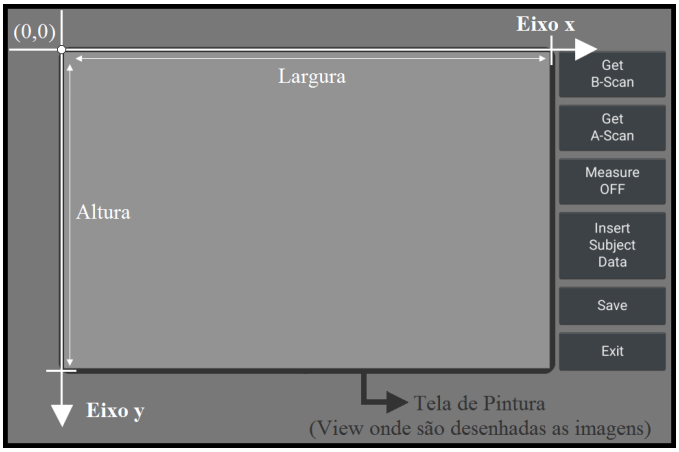


Figura 5.2: Representação do sistema de coordenadas utilizado pela classe `Canvas`.

Pela análise da figura verificamos que qualquer desenho é iniciado a partir do canto superior esquerdo da tela, estendendo-se ao longo da largura e altura da mesma. É de salientar ainda que as coordenadas representam os pixels do ecrã.

Para realizar uma transformação geométrica sobre uma imagem tendo como base este sistema de coordenadas, recorremos à classe `Matrix`. Basicamente esta classe permite a utilização de uma matriz de transformação afim, M, que transforma as coordenadas (x,y) do `Bitmap` no qual ela é aplicada (equação 5.2).

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = M \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \text{com} \quad M = \begin{bmatrix} m_x & i_x & d_x \\ i_y & m_y & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

Na matriz M apresentada, m está associado ao dimensionamento, d à deslocação, e i à inclinação da imagem. Assim, aplicando esta matriz às coordenadas de uma imagem, esta pode ser dimensionada, deslocada, inclinada ou rodada, [9].

No caso do desenho da imagem B-Scan, para que a imagem sofra uma transformação é passado um objeto da classe `Matrix` ao método `drawBitmap()`. Por sua vez este método encarrega-se de aplicar essa matriz a todos os pixels da imagem B-Scan (coordenadas do `Bitmap`). Para o desenho das imagens A-Scan, como não existe um `Bitmap` relativo ao gráfico a ser apresentado, a matriz de transformação da imagem é aplicada ao `Canvas`, que também contém o seu próprio `Bitmap` e por conseguinte uma matriz de transformação a ele associada. A primeira transformação realizada sobre as imagens B-Scan é feita no sentido de estas ficarem compreendidas entre os limites da tela e de manterem uma determinada relação aspeto. Para as imagens A-Scan também é feito um dimensionamento dos dados para que estes fiquem compreendidos na tela de pintura. Contudo, o dimensionamento associado ao eixo y é feito para que o utilizador possa obter uma melhor representação do gráfico, e portanto inicialmente a imagem excede os limites superior e inferior da tela. No apêndice A descreve-se uma explicação sobre a transformação inicial a que estas imagens estão sujeitas.

5.3.1 Dimensionamento e Deslocação da Imagem

A classe `View`, para além de ser responsável pelo desenho da tela como dito anteriormente, também é responsável pela manipulação de eventos que possam ocorrer nessa tela. Em concreto estes eventos dizem respeito às ações de toque do utilizador. Estes eventos são "escutados"pelo método `onTouchEvent()` da classe `View`, utilizando um *switch* que interpreta os diferentes eventos recebidos. Em [8] e [7] encontra-se uma descrição acerca da utilização deste método e das suas diferentes aplicações. Essencialmente, este método interpreta as seguintes ações para cada evento "escutado"(aqui um ponteiro está associado a um dedo, uma caneta, etc.):

- **ACTION_DOWN** - O utilizador colocou um ponteiro sobre o ecrã;
- **ACTION_POINTER_DOWN** - O utilizador colocou outro ponteiro no ecrã;
- **ACTION_MOVE** - O utilizador move um dos ponteiros;
- **ACTION_UP** - O utilizador removeu um ponteiro do ecrã que não o primeiro;
- **ACTION_POINTER_UP** - O utilizador removeu o primeiro ponteiro colocado no ecrã.

Cada ponteiro contém um identificador a ele associado, de forma que seja possível distinguir vários dedos colocados sobre o ecrã e contém ainda as respetivas coordenadas, segundo o sistema de eixos apresentado anteriormente na figura 5.2. Posto isto a ideia é conseguir efetuar determinados gestos sobre o ecrã que possibilitem deslocar e dimensionar a imagem, como se ilustra na figura 5.3.

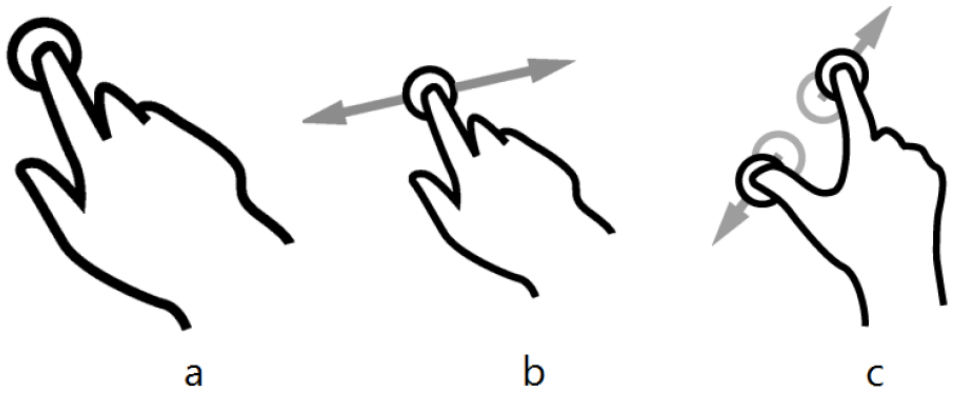


Figura 5.3: Representação de três gestos de toque: (a) - Início de um gesto; (b) - Gesto associado ao deslocamento da imagem; (c) - Gesto associado ao dimensionamento da imagem. Editado de [11].

Posto isto descrevemos agora como procedemos para transformar a imagem com base nestes gestos, utilizando uma matriz de transformação (através de um objeto da classe `Matrix`). Para facilitar a explicação deste processo denotaremos a partir de agora o gesto para dimensionamento por ZOOM e o gesto para deslocação por DRAG. Começamos por declarar duas matrizes na classe `DrawingView` (uma para o valor atual e outra para o valor anterior à transformação). Estas matrizes serão usadas no método `onTouchEvent()` para transformar a imagem e são calculadas dentro de um *switch*[1] quando um gesto for detetado. Após a computação deste *switch* chamamos o método `invalidate()` que força `View` a realizar o desenho da nova imagem. Precisamos também de uma variável de estado que permita distinguir o tipo de gesto que está a ser efetuado (DRAG, ZOOM, ou nenhum dos dois). Começando pelo gesto DRAG, este gesto inicia-se quando o utilizador coloca um dedo sobre o ecrã (ACTION_DOWN) e termina quando o utilizador retira o dedo (ACTION_UP). Quando este gesto se inicia guardamos o valor atual da matriz de transformação e a posição inicial do ponteiro. Sempre que o utilizador mover o dedo utilizamos o valor da matriz original que foi guardada e chamamos o método `postTranslate()` para adicionar um vetor de translação, isto é, a diferença entre a posição inicial e final do ponteiro. Em 1 apresentamos o algoritmo desenvolvido para este gesto, implementado no método `onTouchEvent`.

[1]Estrutura seletiva utilizada em programação. Verifica uma variável e age de acordo com os seus casos.

Algoritmo 1 Algoritmo para o gesto DRAG

Switch(tipo de evento detetado):

1: caso ACTION_DOWN:

⇒ matriz anterior = matriz atual

⇒ posição inicial = posição do ponteiro (as suas coordenadas)

⇒ modo = DRAG

2: caso ACTION_MOVE:

⇒ se modo for DRAG:

→ matriz atual = matriz anterior

→ aplicar `postTranslate(`posição atual - posição inicial`)` à matriz atual

3: caso ACTION_POINTER_UP:

⇒ modo = nenhum

4: caso ACTION_UP:

⇒ não fazer nada.

Chamada do método `invalidate()`.

O gesto de ZOOM diferencia-se do anterior no sentido em que é necessário um dedo extra para efetuar o gesto. Assim, ao colocar outro dedo (ACTION_POINTER_DOWN) sobre o ecrã, calculamos e guardamos o valor da distância entre os dois dedos, e ainda o valor do ponto médio entre eles. Para calcular a distância entre os dois pontos basta apenas aplicar o teorema de Pitágoras, e para o cálculo do ponto médio basta somar as abcissas (x) e as ordenadas (y) dos dois ponteiros e dividir o resultado de cada soma por 2. A partir daqui, sempre que for detetado movimento (ACTION_MOVE) no modo ZOOM, voltamos a calcular a distância entre os dedos, calculando de seguida a matriz atual (aplica-se um dimensionamento em torno do ponto médio determinado). Este dimensionamento é calculado com base na razão entre a nova e a antiga distância entre dedos. Se a nova distância for maior que a primeira então o resultado será maior do que 1 fazendo com que a imagem aumente, caso contrário o resultado é inferior a 1 e a imagem diminui. Assim apresentamos de seguida o algoritmo 2 que representa o gesto ZOOM.

Algoritmo 2 Algoritmo para o gesto ZOOM

Switch(tipo de evento detetado):

1: caso ACTION_DOWN:

⇒ (fica como definido no algoritmo 1)

2: caso ACTION_POINTER_DOWN:

⇒ calcular a distância entre os dois ponteiros (distância antiga)

⇒ matriz anterior = matriz atual

⇒ calcular o ponto médio entre os dois ponteiros

⇒ modo = ZOOM

3: caso ACTION_MOVE:

⇒ se modo for DRAG:

→ (fica como definido no algoritmo 1)

⇒ se modo for ZOOM:

→ calcular nova distância entre dedos (distância atual)

→ matriz atual = matriz anterior

→ calcular dimensionamento (=distância atual/distância antiga)

→ aplicar `postScale(`dimensionamento x, dimensionamento y, ponto medio x, ponto medio y`)` à matriz atual

4: caso ACTION_POINTER_UP:

⇒ modo = nenhum

5: caso ACTION_UP:

⇒ não fazer nada.

Chamada do método `invalidate()`.

Importa salientar que para definir a matriz atual utilizou-se o método `postScale()` (da classe `Matrix`), que recebe 4 parâmetros: os valores de dimensionamento para cada um dos eixos e as coordenadas do ponto médio. Este método aplica um dimensionamento à imagem com centro no ponto médio determinado.

A título de exemplo, apresentamos de seguida as transformações geométricas associadas ao método `postTranslate(Tx,Ty)`. Suponhamos que a matriz inicial é a matriz M, e que a matriz T contém a informação da translação pretendida. Ao aplicarmos o método `postTranslate(Tx,Ty)`

à matriz M (`M.postTranslate(Tx,Ty)`) obtemos a matriz $M^{'}$:

$$M^{'} = M \cdot T = \begin{bmatrix} m_x & 0 & d_x \\ 0 & m_y & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} m_x & 0 & d_x + m_x \times T_x \\ 0 & m_y & d_y + m_y \times T_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

Podemos ainda aqui enunciar a geometria analítica que se esconde por trás do processo de transformação da imagem. Suponha que temos um ponto $P(x, y)$ no momento anterior à transformação da imagem, e o mesmo ponto $P^{'}(x^{'}, y^{'})$ após a transformação. A seguinte equação descreve esta transformação.

$$P^{'} = M^{'} \cdot P \Leftrightarrow \begin{bmatrix} x^{'} \\ y^{'} \\ 1 \end{bmatrix} = \begin{bmatrix} m_x & 0 & d_x + m_x \times T_x \\ 0 & m_y & d_y + m_y \times T_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \times m_x + d_x + m_x \times T_x \\ y \times m_y + d_y + m_y \times T_y \\ 1 \end{bmatrix} \tag{5.4}$$

Para um melhor entendimento acerca deste tópico, poderá encontrar no apêndice B as equações paramétricas que definem as transformações realizadas sobre as imagens B-Scan.

5.4 Medir Distâncias sobre a Imagem

Para além das funcionalidades apresentadas anteriormente permitimos ainda que o utilizador possa medir distâncias sobre as imagens. Para que tal seja possível temos de saber a razão entre o número de pixels (amostras no caso das imagens A-Scan) e a unidade de medida aqui utilizada (milímetros). Como referimos na seção 5.2.1 deste capítulo, os dados relativos a cada uma destas imagens transportam essa informação no seu cabeçalho. Assim introduzimos um botão que permite realizar estas medições (figura 5.1, `View` 4).

Isto é realizado utilizando os mesmos princípios de transformação geométrica que descrevemos anteriormente. Basicamente, para o caso das imagens B-Scan, inserimos um gesto adicional que se traduz na inserção de um marcador na imagem após uma longa ação de toque sem mover o dedo. Este gesto é interpretado dentro do *switch* que deteta os diferentes gestos recebidos. Quando estiverem dois marcadores sobre o ecrã calcula-se o número de pixels entre eles (da mesma forma que é calculada para o gesto ZOOM) e converte-se este número de pixels para a distância em milímetros correspondente. Existe um retângulo fictício que define a área associado ao marcador, para que quando o utilizador colocar o dedo sobre essa área poder mudar a localização deste na imagem visualizada. Quando ocorrem gestos de ZOOM ou DRAG, tanto o marcador como o retângulo a ele associado sofrem as mesmas transformações que a imagem, podendo estes ser

alterados e a distância entre ambos calculada para qualquer que seja o dimensionamento ou a deslocação aplicada à imagem.

No caso das imagens A-Scan, como só temos um eixo representativo de distâncias (eixo x), resolvemos utilizar duas barras verticais, sendo que o número de pixels entre essas barras representa a distância calculada. Como no caso dos marcadores das imagens B-Scan, estas barras possuem retângulos a elas associados que permitem detetar a colocação de um dedo sobre elas e por conseguinte a sua posição pode ser alterando. Desta forma o utilizador pode medir distâncias ao longo do sinal apresentado.

5.5 Considerações sobre as Imagens A-Scan

Para finalizar importa ainda referir alguns aspetos sobre as imagens A-Scan visualizadas. Quando o utilizador visualiza uma imagem deste tipo é possível a identificação da região do cristalino do olho (os dados são de olhos de rato). Para além disto nesta imagem é ainda possível a deteção de cataratas e respetiva classificação. Estas características são obtidas a partir dos dados recebidos para a visualização da imagem (determinada no âmbito de outra dissertação, [12]), não sendo necessário qualquer tratamento de dados adicional. Para visualizar estas características introduzimos dois botões sobre a tela de pintura. Um permite a visualização das estruturas do olho, e a deteção e classificação de cataratas. O outro botão permite a visualização de um conjunto de características relativas ao sinal visualizado.

Capítulo 6

Resultado Final

O resultado final de tudo o que foi apresentado até aqui consiste na aplicação Android que permite a visualização de imagens ecográficas tipo A-Scan e B-Scan. Como tal vamos agora descrever como o utilizador poderá usufruir das diferentes funcionalidades que esta aplicação possui.

Assim observe em primeiro lugar a figura 6.1 que explica o processo para a obtenção de uma imagem B-Scan. Inicialmente o utilizador deve pressionar o botão **Get B-Scan**, sendo que de seguida aparecerá uma janela com 5 parâmetros. Estes valores inicialmente aparecem predefinidos, mas o utilizador poderá alterá-los bastando apenas carregar sobre um deles. Em concreto estes parâmetros representam as entradas do programa que foi apresentado no capítulo 3.

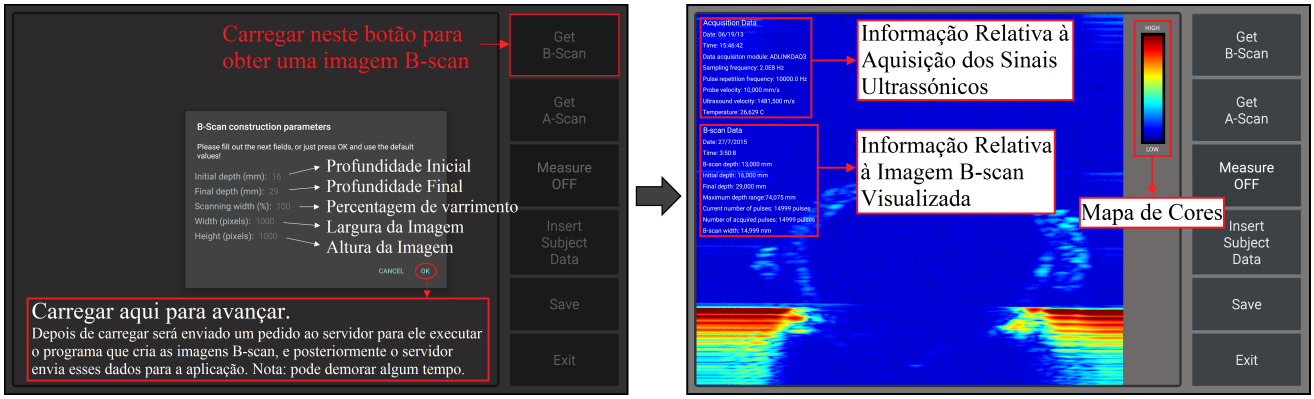


Figura 6.1: Demonstração para obtenção de uma imagem B-Scan.

Uma vez obtida a imagem o utilizador poderá realizar medidas sobre a mesma. Para tal basta clicar sobre o botão **Measure OFF** e proceder de acordo com o que está explicado na figura 6.2.

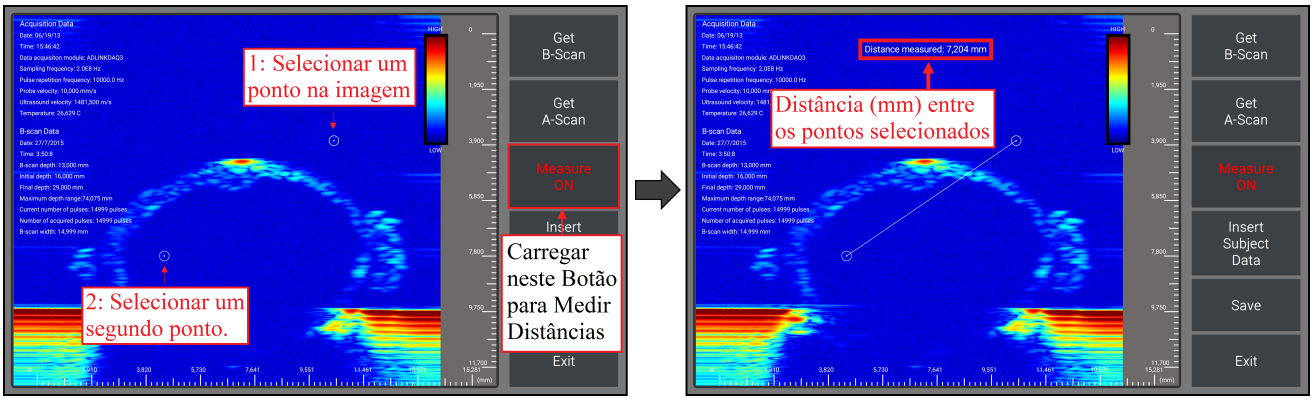


Figura 6.2: Demonstração da medição de distâncias em imagens B-Scan.

Assim como no processo anterior, para a obtenção de uma imagem A-Scan o utilizador deve pressionar o botão **Get A-Scan**. Quando a imagem for apresentada o utilizador poderá então usufruir de diversas funcionalidades, como é descrito nas figuras 6.3 e 6.4.

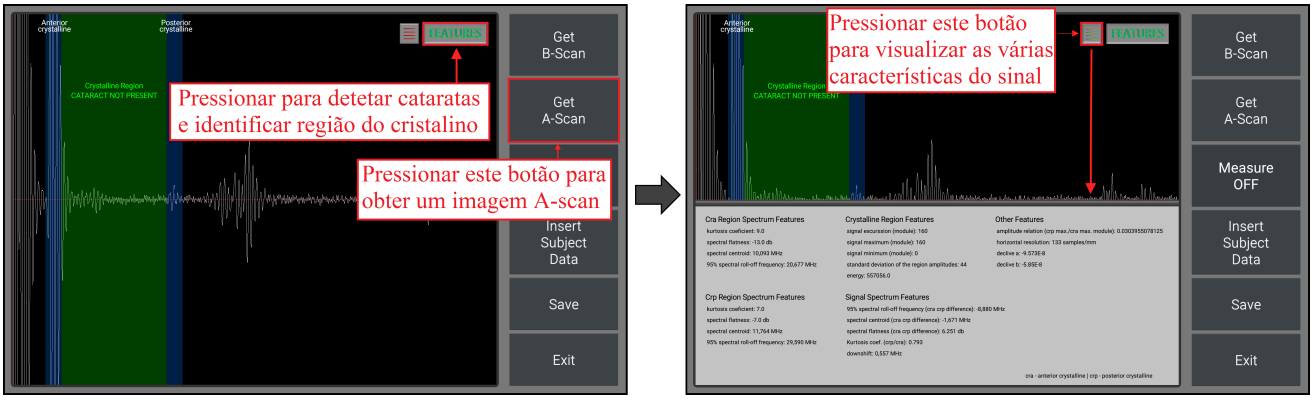


Figura 6.3: Demonstração para obtenção de uma imagem A-Scan

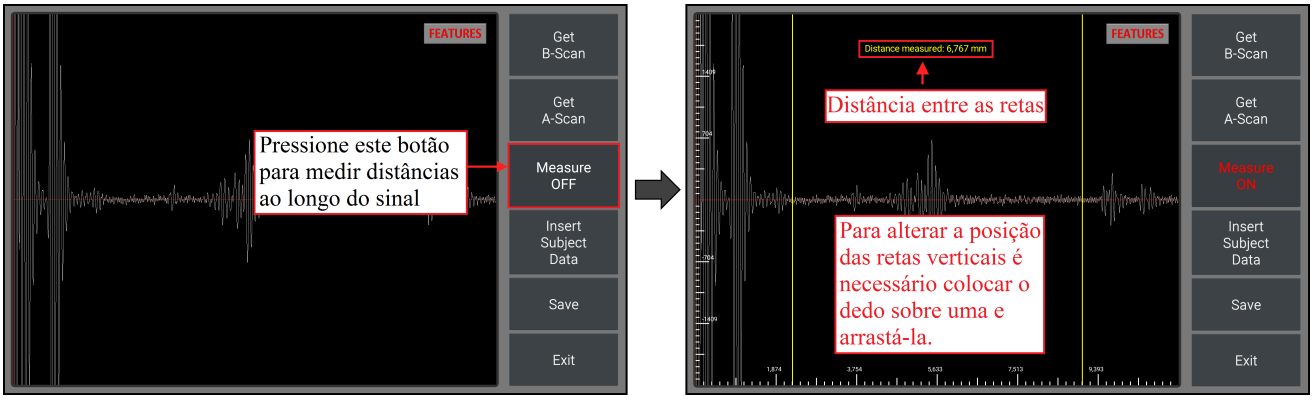


Figura 6.4: Demonstração da medição de distâncias em imagens A-Scan.

Finalmente, os botões **Save** e **Exit** apresentados nas figuras anteriores, permitem respetivamente, salvar a imagem para a galeria do dispositivo e sair da aplicação.

Capítulo 7

Conclusão

Nesta dissertação descreve-se como foi projetado um sistema que permite a visualização de imagens ecográficas do tipo A-Scan e B-Scan. O instrumento ecográfico é simulado por um servidor que tem acesso a dados ecográficos, os processa e envia para o visualizador (*smartphone* ou *tablet*). Por sua vez, desenvolveu-se uma aplicação Android para o visualizador, que permite a visualização das imagens ecográficas, realização de medições, marcação da região do cristalino, deteção e classificação de cataratas. Conseguimos assim separar o sistema de visualização da aquisição e processamento dos dados, aumentando a mobilidade do seu utilizador, um médico ou técnico ecográfico. Dito isto, pode-se afirmar que os objetivos foram cumpridos.

O facto da transferência de dados ser feita sem a necessidade de uma ligação física entre o visualizador e o sistema de aquisição apresenta-se como uma inovação na área dos instrumentos oftalmológicos.

Para trabalho futuro propõe-se que sejam acrescentadas novas funcionalidades à aplicação e que as existentes possam ser melhoradas. Essencialmente, esta proposta centra-se na implementação de um sistema que permita a visualização dos dados em tempo real, utilizando efetivamente um instrumento para a aquisição ecográfica, em alternativa a um servidor, que aqui é utilizado para simular esse instrumento.

Bibliografia

[1] Android-developers.blogspot.pt: *Multithreading For Performance — Android Developers Blog*. [Online]`http://android-developers.blogspot.pt/2010/07/multithreading-for-performance.html`, 2015.

[2] Developer.android.com: *App Manifest — Android Developers*. [Online]`http://developer.android.com/guide/topics/manifest/manifest-intro.html`, 2015.

[3] Developer.android.com: *Application Fundamentals — Android Developers*. [Online]`http://developer.android.com/guide/components/fundamentals.html`, 2015.

[4] Developer.android.com: *AsyncTask — Android Developers*. [Online]`http://developer.android.com/reference/android/os/AsyncTask.html`, 2015.

[5] Developer.android.com: *Bitmap — Android Developers*. [Online]`http://developer.android.com/reference/android/graphics/Bitmap.html`, 2015.

[6] Developer.android.com: *Building a Simple User Interface — Android Developers*. [Online]`http://developer.android.com/training/basics/firstapp/building-ui.html`, 2015.

[7] Developer.android.com: *Dragging and Scaling — Android Developers*. [Online]`https://developer.android.com/training/gestures/scale.html`, 2015.

[8] Developer.android.com: *Handling Multi-Touch Gestures — Android Developers*. [Online]`https://developer.android.com/training/gestures/multi.html`, 2015.

[9] Developer.android.com: *Matrix — Android Developers*. [Online]`http://developer.android.com/reference/android/graphics/Matrix.html`, 2015.

[10] Developer.android.com: *Resources Overview — Android Developers*. [Online]`http://developer.android.com/guide/topics/resources/overview.html`, 2015.

[11] Ed Burnette: *Hello, Android*. Pragmatic Bookshelf, 2010.

[12] J. Ferreira: *Sistema de aquisição ecográfica para oftalmologia usando uma arquitectura SoC Zynq-7000*. Tese de Mestrado, Universidade de Coimbra, 2015.

[13] J. Kurose, K. Ross: *Computer networking*. Pearson/Addison Wesley, 2008.

[14] micromedical: *E-Z Tip Immersion (Box of 100)*. [Online]`http://micromedinc.com/our-devices/e-z-tip-immersion-box-of-100/`, 2015.

[15] Mir Nauman Tahir: *Learning Android Canvas*. Packt Publishing Ltd., 2013.

[16] Msdn.microsoft.com: *HTTP Handlers and HTTP Modules Overview*. [Online]`https://msdn.microsoft.com/en-us/library/bb398986.aspx`, 2015.

[17] Optometron.pt: *US-500 Biometro/ Paquímetro - Optometron*. [Online]`http://www.optometron.pt/US-500.html`, 2015.

[18] Pt.Wikipedia.org: *Oftalmologia*. [Online] `https://pt.wikipedia.org/wiki/Oftalmologia`, 2015.

[19] Technology, ADLINK: *PCIe/PXIe-9842 Datasheet*. [Online] `http://www.adlinktech.com/PD/marketing/Datasheet/PXIe-9842/PXIe-9842_Datasheet_en_1.pdf`, 2013.

[20] W. Hendee e E. Ritenour: *Medical imaging physics*. Wiley-Liss, New York, 2002.

[21] Wikipedia: *ASP.NET*. [Online]`https://en.wikipedia.org/wiki/ASP.NET`, 2015.

Apêndice A

Transformações Geométricas Iniciais

Apresentação Inicial de uma imagem B-Scan no *Canvas*

Uma imagem B de dimensão $[B_x \times B_y](Pixels)$ vai ser apresentada no ecrã através de um *Canvas* C de dimensão $[C_x \times C_y](Pixels)$. A dimensão de B em milímetros é $[B_{mx} \times B_{xy}]$.

A imagem é transformada de $B \rightarrow C$ segundo a expressão $C = M \cdot B$, onde M é uma matriz de transformação afim.

Os referenciais utilizados são:

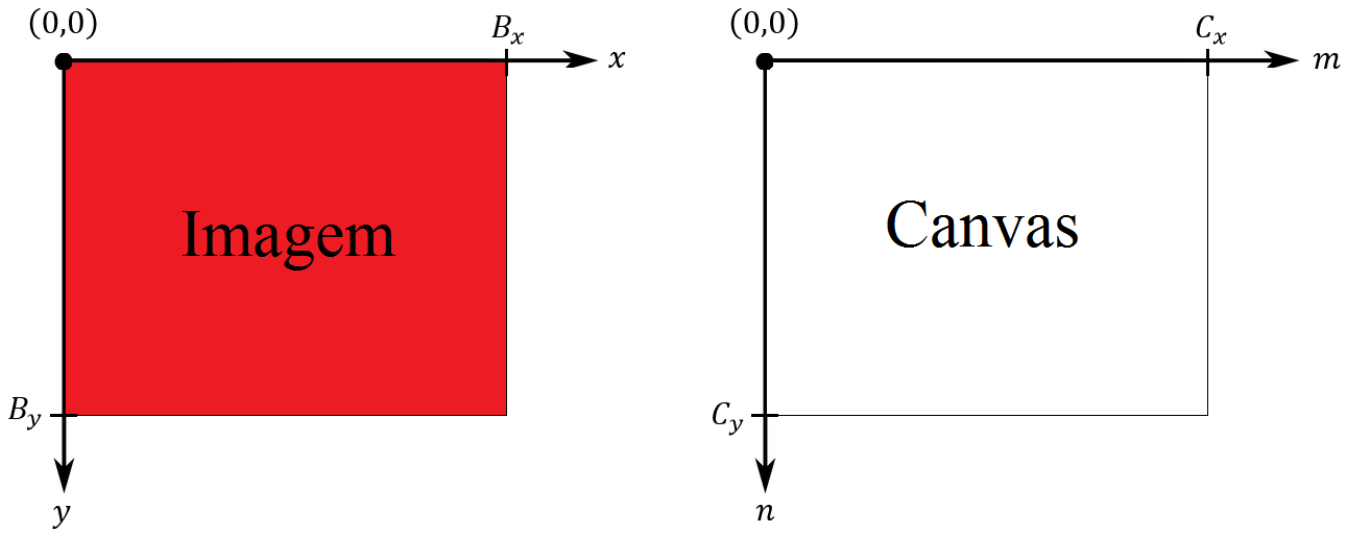


Figura A.1: Referênciais da imagem e do *Canvas*.

Devemos assumir que:

- Poderá haver distorção de escala, isto é, poderá ser aplicado um fator de dimensionamento diferente às duas dimensões;

- A imagem pode ser maior ou menor do que o *Canvas*;

- Temos de manter a relação aspeto da imagem em relação aos milímetros em altura e largura;

- A transformação afim é definida com uma matriz de transformação 3×3 de tal forma que:

$$\begin{bmatrix} m \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & d_x \\ 0 & s_y & d_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{cases} m = s_x \times x + d_x \\ n = s_y \times y + d_y \\ 1 = 1 \end{cases} \tag{A.1}$$

Assim para apresentarmos a imagem no *Canvas* basta determinar s_x, s_y, d_x e d_y. Poderão ocorrer dois casos:

⇒ **Primeiro Caso**

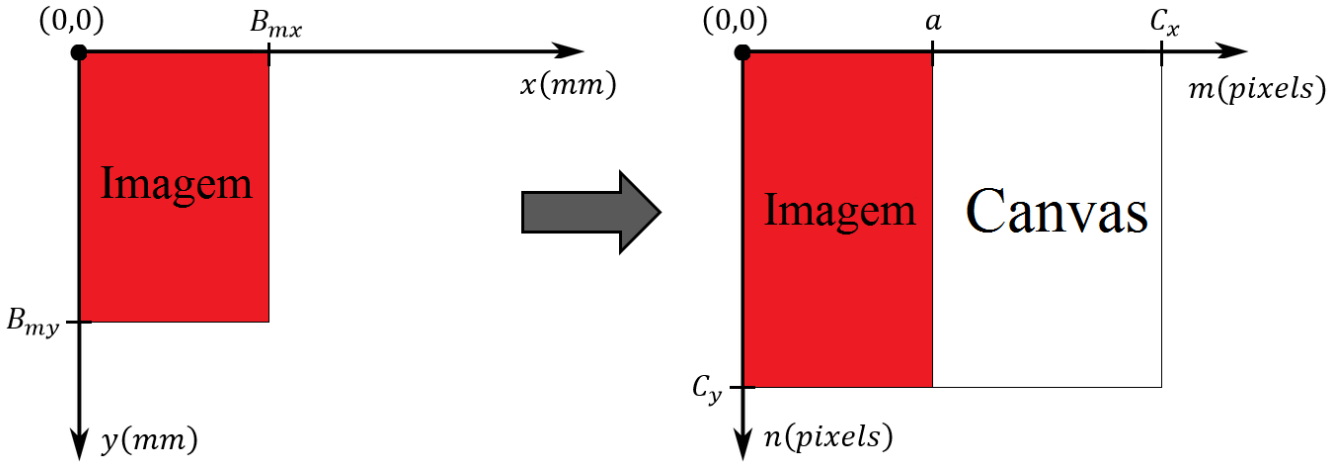


Figura A.2: A imagem é dimensionada de forma a preencher os limites do *Canvas* em altura.

Como $\frac{C_y}{B_{my}} < \frac{C_x}{B_{mx}}$ então:

$$s_x = \frac{B_{mx} \times \frac{C_y}{B_{my}}}{B_x} \Rightarrow a = B_{mx} \times \frac{C_y}{B_{my}} \tag{A.2}$$

$$s_y = \frac{B_{my} \times \frac{C_y}{B_{my}}}{B_y} \Rightarrow B_y \times s_y = C_y \tag{A.3}$$

$\Rightarrow$ **Segundo Caso**

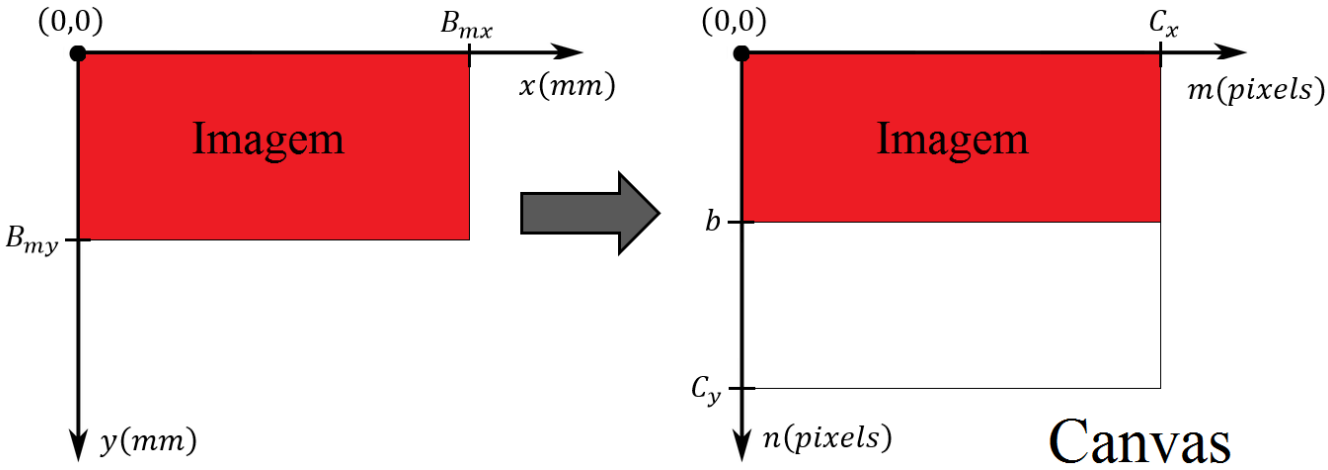


Figura A.3: A imagem é dimensionada de forma a preencher os limites do *Canvas* em largura.

Como $\frac{C_y}{B_{my}} > \frac{C_x}{B_{mx}}$ então:

$$s_x = \frac{B_{mx} \times \frac{C_x}{B_{mx}}}{B_x} \Rightarrow B_x \times s_x = C_x \tag{A.4}$$

$$s_y = \frac{B_{my} \times \frac{C_x}{B_{mx}}}{B_y} \Rightarrow b = B_{my} \times \frac{C_x}{B_{mx}} \tag{A.5}$$

Como seria de esperar $d_x = 0$ e $d_y = 0$, em ambos os casos.

Apresentação Inicial de uma imagem A-Scan no *Canvas*

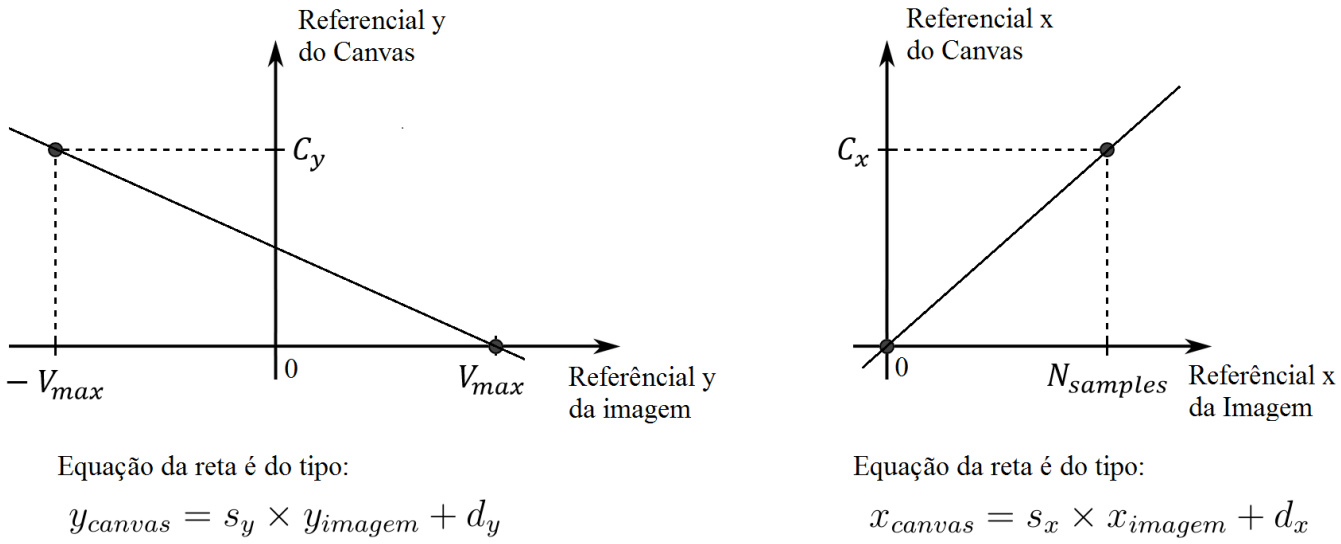
Considere que o vetor representativo do sinal recebido tem $N_{samples}$ elementos e V_{max} representa a amplitude máxima do sinal em módulo.

Esta imagem deve ficar compreendida nos limites do *Canvas*, que tem dimensão $[C_x \times C_y]$.

A transformação afim é definida com uma matriz de transformação 3×3 de tal forma que:

$$\begin{bmatrix} x_{canvas} \\ y_{canvas} \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & d_x \\ 0 & s_y & d_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_{imagem} \\ y_{imagem} \\ 1 \end{bmatrix} = \begin{cases} x_{canvas} = s_x \times x_{imagem} + d_x \\ y_{canvas} = s_y \times y_{imagem} + d_y \\ 1 = 1 \end{cases} \tag{A.6}$$

Assim devemos proceder de acordo com o seguinte:



Equação da reta é do tipo:

$$y_{canvas} = s_y \times y_{imagem} + d_y$$

Equação da reta é do tipo:

$$x_{canvas} = s_x \times x_{imagem} + d_x$$

Figura A.4: Escalamento dos referenciais da imagem para os referenciais do *Canvas*.

Da figura A.4 obtemos:

$$s_y = -\frac{C_y}{2V_{max}} \qquad d_y = \frac{C_y}{2} \tag{A.7}$$

$$s_x = -\frac{C_x}{N_{samples}} \qquad d_x = 0 \tag{A.8}$$

Apêndice B

Transformações Geométricas Utilizando Gestos

Deslocar uma imagem B-Scan no *Canvas*

Considere uma imagem B-Scan B com dimensão $[y_B \times x_B]$ e um *Canvas* C com dimensão $[y_C \times x_C]$.

Podemos deslocar a imagem na horizontal, na vertical, ou em ambas as direções.

A imagem é transformada de $B \rightarrow C$ segundo a expressão $C = M \cdot B$, onde M é uma matriz de transformação afim.

Os referenciais utilizados são:

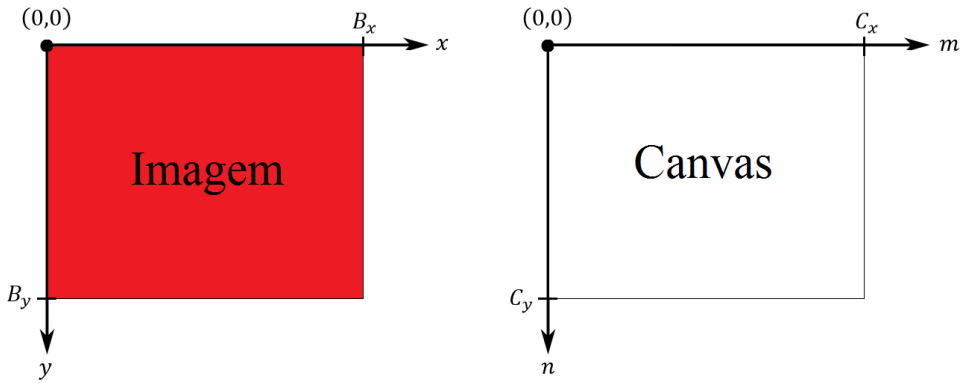


Figura B.1: Referenciais da imagem e do *Canvas*.

A transformação afim é definida com uma matriz de transformação 3×3 de tal forma que:

$$\begin{bmatrix} m \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & d_x \\ 0 & s_y & d_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{cases} m = s_x \times x + d_x \\ n = s_y \times y + d_y \\ 1 = 1 \end{cases} \tag{B.1}$$

A figura B.2, apresentada a seguir, ilustra este processo de deslocar a imagem ao longo do *Canvas*.

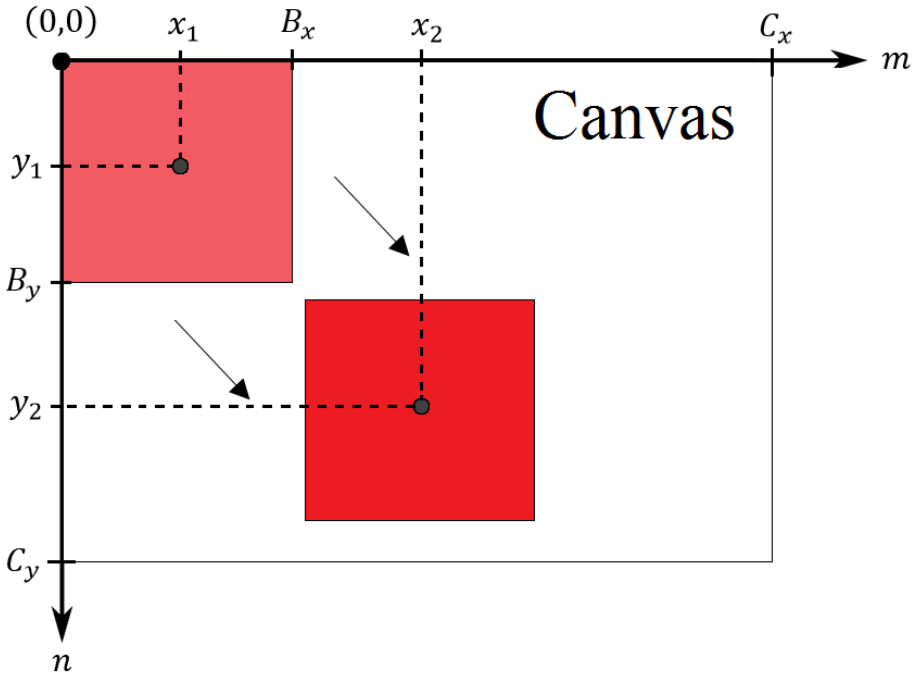


Figura B.2: Deslocação de uma imagem no *Canvas*.

Com base na figura B.2 temos:

$$s_x = 0 \qquad d_x = x_2 - x_1 \tag{B.2}$$

$$s_y = 0 \qquad d_y = y_2 - y_1 \tag{B.3}$$

Dimensionar uma imagem B-Scan no *Canvas*

Considere uma imagem B-Scan B com dimensão $[B_x \times B_y]$ e um *Canvas* C com dimensão $[C_x \times C_y]$.

Pretende-se dimensionar a imagem (fazer zoom) de fator α com centro no centro do *Canvas* $(\frac{C_x}{2}, \frac{C_y}{2})$.

A imagem é transformada de $B \rightarrow C$ segundo a expressão $C = M \cdot B$, onde M é uma matriz de transformação afim.

A transformação afim é definida com uma matriz de transformação 3×3 de tal forma que:

$$\begin{bmatrix} m \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha s & 0 & d_x \\ 0 & \alpha s & d_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{cases} m = \alpha s \times x + d_x \\ n = \alpha s \times y + d_y \\ 1 = 1 \end{cases} \tag{B.4}$$

A figura B.3, apresentada a seguir, ilustra este processo de dimensionar a imagem no *Canvas*, com centro no centro no *Canvas*.

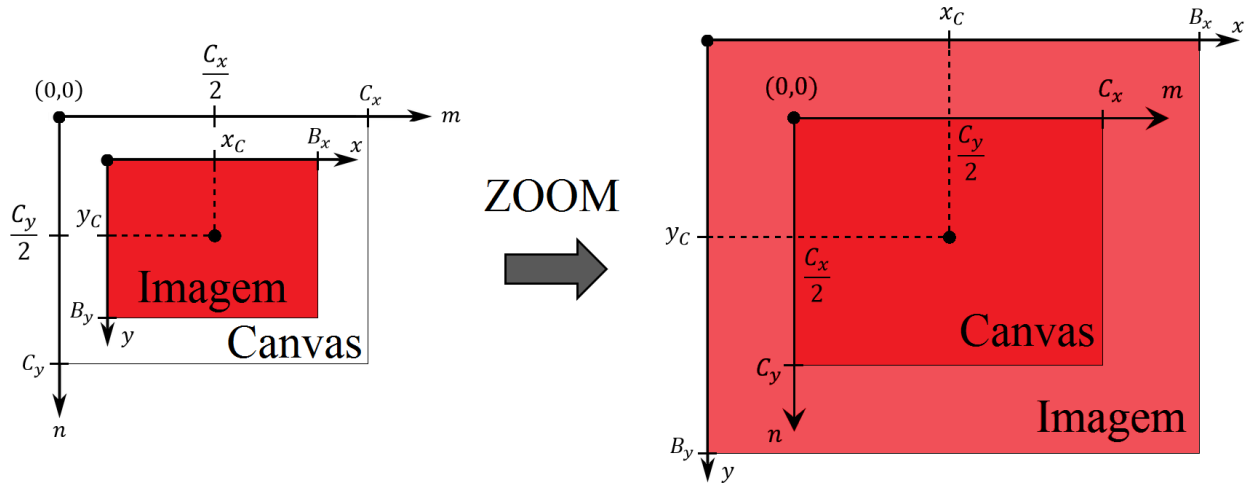


Figura B.3: Dimensionamento de uma imagem no *Canvas*.

1º Passo: Calcular o centro do *Canvas* na Imagem.

$$\frac{C_x}{2} = \alpha s \times x_C + d_x \Rightarrow x_C = \frac{1}{\alpha s}(\frac{C_x}{2} - d_x) \tag{B.5}$$

$$\frac{C_y}{2} = \alpha s \times y_C + d_y \Rightarrow y_C = \frac{1}{\alpha s}(\frac{C_y}{2} - d_y) \tag{B.6}$$

2º Passo: Calcular o deslocamento da imagem.

$$d_x = \frac{C_x}{2} - \alpha s \times x_C \tag{B.7}$$

$$d_y = \frac{C_y}{2} - \alpha s \times y_C \tag{B.8}$$

3º Passo: Aplicar a transformação $C = M \cdot B$.