UNIVERSIDADE D
COIMBRA

Carlos Daniel Silva Pinto

DESENVOLVIMENTO DE UM SISTEMA DE CLASSIFICAÇÃO E VISUALIZAÇÃO DE CATARATAS EM HUMANOS

Dissertação no âmbito do Mestrado Integrado em Engenharia Electrotécnica e de Computadores orientado pelo Professor Doutor Fernando Manuel dos Santos Perdigão e coorientado pelo Professor Doutor Marco Alexandre Cravo Gomes apresentado à Faculdade de Ciências e Tecnologia da Universidade de Coimbra Departamento de Engenharia Electrotécnica e de Computadores

Setembro de 2019

Agradecimentos

Agradeço ao professor Doutor Fernando Manuel dos Santos Perdigão e ao professor Doutor Marco Alexandre Cravo Gomes por toda a orientação, ajuda e cooperação ao longo da dissertação. Foram essenciais para a resolução dos problemas que foram encontrados ao longo do trabalho. E muito obrigado pelo feedback fornecido durante a escrita deste documento.

Um agradecimento a todo o conjunto de pessoas envolvidas neste trabalho através da realização de críticas construtivas permitindo assim melhorar a qualidade do trabalho produzido.

Agradeço à minha família pelos valores incutidos, pela motivação e mentalidade necessária para atingir este objetivo e os próximos que virão.

Por fim, mas não menos importante agradeço a todos os meus amigos que me acompanharam e ajudaram nesta jornada.

Este trabalho foi financeiramente apoiado por FCT/MEC através de fundos nacionais e cofinanciado por POCI – COMPETE2020 e pelo acordo de parceria FEDER – PT2020 no âmbito dos projetos POCI-01-0145-FEDER-028758 (CATARACTUS) e UID/EEA/50008/2019 (Instituto de Telecomunicações).

Resumo

A catarata é uma doença ocular cujo tratamento é atualmente feito através de técnicas cirúrgicas como a facoemulsificação[1] e cirurgia assistida por lasers de femtosegundo. A facoemulsificação consiste fundamentalmente na utilização de ultrassons para fragmentar a catarata. O nível de energia dos ultrassons utilizado na operação é crucial para o sucesso da mesma. Devido principalmente à necessidade de calcular o nível adequado de energia dos ultrassons, é criado o projeto CATARACTUS (POCI-01-0145-FEDER-028758) financiado pela Fundação para a Ciência e Tecnologia (FCT). Este projeto tem como objetivo o desenvolvimento de um dispositivo médico oftalmológico destinado a auxiliar o médico na cirurgia da catarata disponibilizando, em tempo-real, informação sobre a dureza e localização da catarata bem como a estimação do nível adequado de energia dos ultrassons a aplicar.

Esta dissertação, está integrada no projeto CATARACTUS e centra-se no desenvolvimento do sistema de aquisição e tratamento de dados recolhidos através da sonda, e disponibilização ao médico de informação relevante, já processada, para auxílio à cirurgia.

Ao longo deste documento, são apresentadas e discutidas duas soluções que foram desenvolvidas para o sistema de aquisição. Na primeira solução temos a separação do módulo que gera os pulsos de excitação, do módulo de aquisição de dados, enquanto na segunda temos a agregação de ambos os módulos num único. A discussão das duas soluções irá debruçar-se sobre as vantagens e desvantagens de cada solução, os respetivos processos de integração no dispositivo médico e ainda o processo de transferência dos dados adquiridos para um computador hospedeiro. Por fim, é discutido o software de visualização de sinais com o qual o médico recolherá e visualizara os dados de doentes em ambiente clínico, e como é feita a integração de cada uma das duas soluções neste software.

Palavras-Chaves: Catarata, Ultrassons, *Transmission Control Protocol* (TCP), ESUS, CATARACTUS, xSCAN, Sistema de Aquisição de sinais

[1] Facoemulsificação é o nome atribuído à técnica mais atual para a Cirurgia de Catarata (Caixinha, 2016)

Abstract

Cataract is an ocular condition which nowadays is treated only by surgery. One of the most used techniques is phacoemulsification[1]. This technique uses an ultrasonic device to fragment the cataractous lens into small pieces. The phacoemulsification energy level used during the surgery is crucial to its success and to the patient well-being. Project CATARACTUS (POCI-01-0145-FEDER-028758), funded by Portuguese Foundation for Science and Technology, has born from the necessity to know the optimal phacoemulsification energy level. The main goal of this project is to develop a medical prototype, in order to help the surgeon during the surgery, capable of supplying information in real-time such as the cataract location and hardness and the optimal phacoemulsification energy level.

This thesis is part of the CATARACTUS project and the developed is focused on the data acquisition system which is responsible for acquiring the ultrasound signal coming from the ophthalmic probe. Throughout this work it will be presented and discussed two solutions that were developed for the data acquisition system. The first solution consists on the separation of the pulse generator module from the data acquisition module while the second solution combines both the pulse generator and data acquisition module in one module. The discussion addresses the advantages and disadvantages of each solution, how they will be integrated in the medical device and how the acquired data is transferred to a computer. It will also be introduced the visual interface software made for medical personal to acquire data from the patients in a clinical environment and how each solution is integrated in this software.

Keywords: Cataract, Ultrasounds, *Transmission Control Protocol* (TCP), ESUS, CATARACTUS, xSCAN, Signal acquisition system

[1] Phacoemulsification is the name given to the most current technique for Cataract Surgery (Caixinha, 2016)

Índice

Índice de figuras

Índice de tabelas

Lista de Acrónimos

AAO	American Academy of Ophthalmology
AOA	American Optometric Association
API	Application Programming Interface
CEH	Community Eye Health
CART	Classification and Regression Tree
DAQ	Data Acquisition
ECCE	Extracapsular Cataract Extraction
FCT	Fundação para a Ciência e Tecnologia
FLD	Fisher Linear Discriminant
GUI	Graphical User Interface
ICCE	Intracapsular Cataract Extraction
IDE	Integrated Development Environment
IMO	Instituto de Microcirurgia Ocular
JSON	JavaScript Object Notation
KNN	K Nearest-Neighbour
NICE	National Institute for Health and Care Excellence (United Kingdom)
PCA	Principal Component Analysis
SVM	Support Vector Machines
TCP	Transmission Control Protocol
WHO	World Health Organization

1. Introdução

1.1 Descrição do problema

A catarata é uma condição de saúde ocular que afeta parcial ou totalmente a capacidade visual (ver Figura 1.1). De acordo com o National Eye Institute e a World Health Organization (WHO), a catarata é uma doença em que as suas fases são distinguidas através da opacidade do cristalino do olho humano, sendo a principal causa para a cegueira globalmente (Caixinha, Jesus, Velte, Santos, & Santos, 2014) e a segunda maior causa de deficiência visual (WHO, 2010).

Segundo a WHO não existe ainda qualquer tratamento da catarata através de fármacos, sendo a cirurgia a única opção para a recuperação da visão (Caixinha, 2016; Olson, Mamalis, Werner, & Apple, 2003; The Royal College of Ophthalmologists (Scientific Department), 2010; WHO, 1997). Atualmente, a facoemulsificação é a técnica cirúrgica mais utilizada para a remoção da catarata (Olson et al., 2003). Esta técnica utiliza a energia dos ultrassons para fragmentar a catarata. O sucesso desta técnica depende muito da aplicação de um nível de energia de ultrassons adequado, sendo que uma incorreta estimativa desse nível de energia poderá resultar na disrupção da capsula posterior do cristalino e na perda de células endoteliais da córnea (Fine, Packer, & Hoffman, 2002). É assim de extrema importância para o sucesso da cirurgia a obtenção de uma estimativa correta e adequada do nível de energia a usar (Caixinha, 2016). Atualmente o valor da energia dos ultrassons utilizada é estimado com base na classificação da dureza da catarata que é feita através de sistemas de classificação subjetivos como o Lens Opacities Classification System (LOCS) Versão III e o The Oxford Clinical Cataract Classification and Grading System.

Segundo a Organização Mundial de Saúde o número de operações às cataratas tem vindo a aumentar (ver Figura 1.2), o que aponta para a necessidade do desenvolvimento de técnicas mais rigorosas de estimação da energia dos ultrassons utilizada na operação. É com base nesta necessidade que nasce o projeto CATARACTUS[2] (POCI-01-0145-FEDER-028758), financiado pela Fundação para a Ciência e Tecnologia (FCT) cuja finalidade é a construção de um dispositivo médico que auxilie o médico na correta

[2] https://www.it.pt/Projects/Index/4579

caracterização e localização da catarata e também na estimação do nível ótimo de energia dos ultrassons a usar durante a cirurgia.

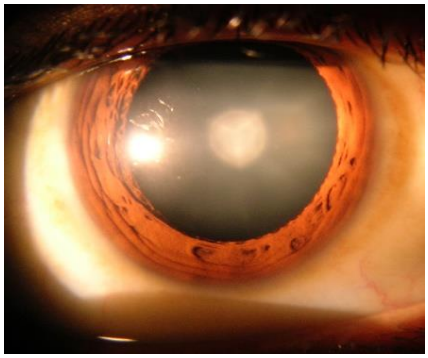

Figura 1.1 - Catarata num olho humano (Ahuja, 2005)[3]

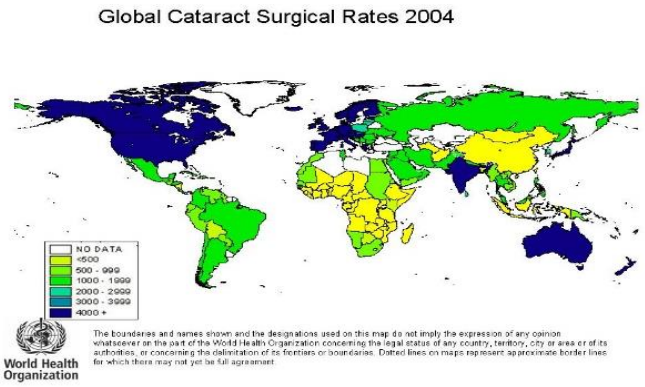

Figura 1.2 - Número de operações feitas à catarata a nível global em 2004 (WHO, 2004)

O projeto CATARACTUS vem na sequência do projeto FCT CATARACTA[4] (PTDC/DTP-PIC/0419/2012) cujo objetivo era a construção de um protótipo capaz de caracterizar e localizar cataratas em ratos Wistar para prova de conceito: deteção e classificação de cataratas através de ultrassons. O projeto CATARACTUS tem por objetivo a transladação do estudo para humanos. O objetivo principal do projeto consiste na evolução do protótipo existente e certificação no sentido da sua aplicação no diagnostico e classificação de cataratas em humanos. O trabalho a ser realizado para atingir este objetivo foi divido em 5 partes:

1) Configuração e desenvolvimento do sistema de aquisição e visualização de sinais em tempo real;
2) Desenvolvimento de algoritmos capazes de detetar as várias interfaces do olho e a presença de catarata;
3) Estimação da dureza da catarata através de métodos de *machine learning*;
4) Estimação da energia ótima de facoemulsificação a utilizar na cirurgia;
5) Desenvolvimento de um protótipo médico capaz de assistir o cirurgião durante a realização da cirurgia à catarata.

[3] Imagem de autoria Rakesh Ahuja extraída em https://pt.wikipedia.org/wiki/Catarata
[4] https://www.it.pt/Projects/Index/1866

1.2 Objetivos da dissertação

A presente dissertação está integrada no projeto CATARACTUS e tem como objetivo o desenvolvimento de um sistema de aquisição e visualização de sinais em tempo-real para ambiente clínico. Ao longo deste documento vão ser apresentadas as duas soluções desenvolvidas para a concretização do sistema. Ambas as soluções fazem uso de uma sonda oftalmológica de ultrassons customizada, um gerador de pulsos de excitação, um módulo de aquisição de dados e um computador para extrair os dados adquiridos e visualizá-los. Devido à necessidade de visualizar os sinais adquiridos em tempo-real, é necessário que os módulos de aquisição de dados de ambas as soluções tenham uma taxa de transferência de dados elevada para o computador hospedeiro. Outro requisito que um sistema a ser usado em ambiente clínico deve cumprir, é de ser de utilização fluida e fácil. Na primeira solução temos o módulo de gerador de pulsos de excitação (*Panametrics*), separado do módulo de aquisição de dados (placa PCIe-9842). Na segunda solução temos o módulo de geração de pulsos de excitação e de aquisição num único módulo customizado, o sistema xSCAN.

1.3 Contribuições

- Sistema de aquisição de sinais em tempo real
    - Produzidos dois sistemas de aquisição de sinais em tempo real. Um com a placa PCIe-9842 que se encontra descrito ao longo do capítulo 3 e outro com o equipamento xSCAN que é abordado e discutido ao longo do capítulo 4.
- API para o dispositivo xSCAN (Capítulo 4.4)
- Programa para aceder aos dados adquiridos e visualizá-los pela Placa PCIe-9842 (Capítulo 3.3)
- Função para integração do dispositivo xSCAN no sistema de aquisição de sinais (Capítulo 4.5)
- Software de visualização gráfica em tempo real (Capítulo 5)
    - Para a visualização dos sinais foi desenvolvido um software de visualização, apresentado no capítulo 5, que agrupa uma interface de dados (capítulo 5.2),

um sistema de diretorias (capítulo 5.3.1) e uma base dados (capítulo 5.3.2) desenvolvida.

- Artigo ConfTele 2019 (Anexo A)
  - O estudo realizado para a construção da API e o seu desenvolvimento levou à produção de um artigo apresentado na ConfTele 2019 em Lisboa. Esse artigo pode ser consultado no Anexo A.

1.4 Organização da dissertação

Este documento está organizado em seis capítulos.

No primeiro capítulo é introduzido o contexto em que esta dissertação se enquadra e o seu objetivo.

No segundo capítulo é feito uma revisão da literatura onde abordamos a catarata: o que é a catarata, o método cirúrgico atualmente mais usado e complicações que possam surgir durante a cirurgia, os diversos classificadores subjetivos da dureza e localização da catarata. Posteriormente, é abordado os ultrassons e o porquê da sua utilização no projeto CATARACTUS. De seguida introduzimos diversos módulos de aquisição atualmente utilizados e as suas características. Por fim, discutimos o projeto CATARACT, o projeto antecessor do atual projeto CATARACTUS: Os seus objetivos e resultados.

No terceiro capítulo é introduzido uma das soluções para o sistema de aquisição e visualização de sinais em tempo-real, a solução com a placa PCIe-9842 da AD-LINK.

No quarto capítulo é apresentada a solução com o xSCAN onde também é discutido o desenvolvimento da API para este dispositivo.

No quinto capítulo é apresentado o software desenvolvido para visualização de sinais em tempo-real.

Por fim, no sexto capítulo apresentamos as conclusões deste trabalho e abordamos o trabalho que se irá realizar no futuro.

2. Estado da Arte

Neste capítulo é apresentado o resultado da pesquisa do estado da arte no âmbito do trabalho de dissertação. Numa primeira fase, é feita uma introdução à catarata onde se aborda a situação desta doença a nível mundial, os comportamentos de risco que aumentam a probabilidade de uma pessoa desenvolver cataratas, a classificação da catarata quanto ao seu tipo e os tratamentos disponíveis. De seguida, é feita uma introdução aos ultrassons onde são descritas as suas potencialidades na deteção e classificação de cataratas, de acordo com os objetivos do projeto, onde este trabalho se insere. Numa fase seguinte, abordamos o estudo dos sistemas de aquisição de dados existentes. Por último, são apresentados os resultados do projeto CATARATA de deteção de catarata e sua classificação in-vivo em ratos de Wistar, que serviram de ponto de partida para o projeto CATARATUS.

2.1 A Catarata

A catarata é uma condição ocular que leva à perda total ou parcial da visão. Esta desenvolve-se no cristalino do olho humano, representada na Figura 2.1, cuja constituição é maioritariamente água e proteínas (National Eye Institute, 2015a). Normalmente as proteínas estão alinhadas por forma a permitir a passagem da luz pelo cristalino (que é responsável pela focagem da luz na retina) sem sofrer qualquer tipo de interferência. A catarata surge quando estas proteínas se agrupam de tal modo que gera no cristalino uma zona de opacidade (Boyd, 2018; National Eye Institute, 2015a) que distorce ou mesmo impede a passagem da luz, efeito que pode ser observado na Figura 2.2. A opacidade pode ser baixa ou elevada consoante a progressão da catarata.

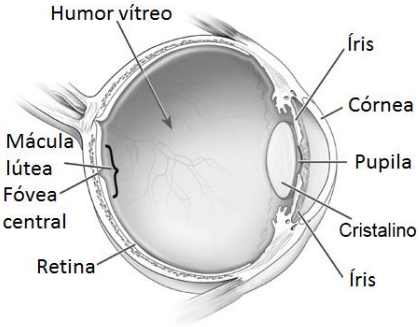

Figura 2.1 - Diagrama do olho humano (adaptada)[5]

Figura 2.2 - Visão sem Catarata (Esquerda) VS Visão com Catarata (Direita)[6]

[5] Imagem original de autoria National Eye Institute extraída em https://nei.nih.gov/health/eyediagram
[6] Imagem de autoria National Eye Institute extraída em https://nei.nih.gov/health/cataract/cataract_facts

Existem três tipos de catarata conforme representadas na Figura 2.3:

- A catarata subcapsular posterior;
- A catarata cortical;
- A catarata nuclear;

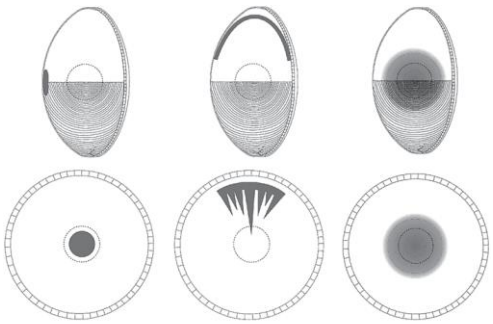

Figura 2.3 - Diferentes tipos de catarata: Catarata subcapsular posterior (esquerda), Catarata cortical (centro), Catarata nuclear (direita) (Beebe, Holekamp, & Shui, 2010)

Os sintomas normalmente associados à presença da catarata são os seguintes (National Eye Institute, 2015b; Royal National Institute of Blind People, 2019):

- Visão turva;
- Visão noturna limitada;
- Cores desvanecidas;
- Frequente alteração da prescrição de lentes/óculos;

De entre os fatores de risco conhecidos que podem levar à criação da catarata são de referir como mais relevantes os seguintes (WHO, 1997):

- Alcoolismo;
- Doenças sistémicas;
- Tabagismo;
- Envelhecimento;
- Diabetes.

A catarata afeta maioritariamente a faixas etárias mais elevadas, apesar de existirem casos registados nas diversas faixas etárias (National Eye Institute, 2015b; NICE, 2018; WHO, 1997). Na figura 2.4 pode ver-se a distribuição percentual de ocorrência de cataratas por faixa etária existente nos Estados Unidos da América em 2010. Na figura 2.5 é apresentada uma previsão efetuada pelo National Eye Institute nos Estados Unidos da América que mostra uma estimativa do número de pessoas com cataratas em 2030 e 2050. Como se pode observar, por comparação a 2010 prevê-se que em 2050 o número de pessoas com catarata irá duplicar.

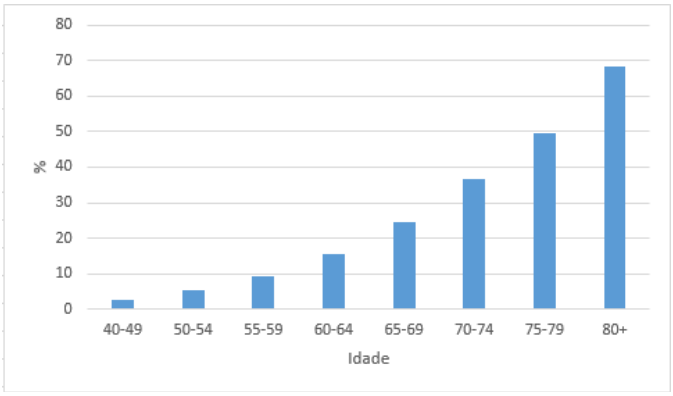

Figura 2.4 - Percentagem de cataratas existentes nos Estados Unidos da América em 2010 por faixa etária[7]

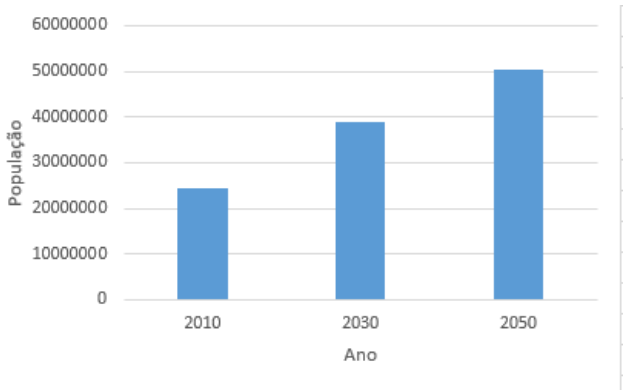

Figura 2.5 - Previsão do número de pessoas com cataratas existentes nos Estados Unidos da América em 2030 e 2050[7]

Como atualmente não existe nenhum fármaco para o tratamento de cataratas, a única opção é a intervenção cirúrgica. Ao longo dos anos a técnica cirúrgica para remoção das cataratas tem vindo a evoluir (Eric J. Linebarger, David R. Hardten, Gaurav K. Shah, & Richard L. Lindstrom, 1999). A facoemulsificação é a técnica cirúrgica usada

[7] Gráfico construído com dados extraídos a partir de https://nei.nih.gov/eyedata/cataract/tables

atualmente a nível global (Caixinha, 2016; Eric J. Linebarger et al., 1999; Huang et al., 2007).

Resumidamente, a cirurgia de facoemulsificação consiste em fazer uma pequena incisão na cápsula anterior do cristalino para que o seu conteúdo opacificado seja fragmentado através da energia de ultrassons e de seguida aspirado. Depois da realização destas etapas é inserido uma lente intraocular (Eric J. Linebarger et al., 1999). Apesar de ser uma técnica cirúrgica avançada, tem as suas desvantagens e complicações (Eric J. Linebarger et al., 1999). Uma das complicações que podem surgir é um uso de energia de ultrassons mais elevada do que o necessário o que pode levar à rotura da cápsula posterior do cristalino (Eric J. Linebarger et al., 1999). Para evitar esta complicação, é necessário que haja uma estimação do nível de energia de facoemulsificação a aplicar em função da dureza da catarata, não existindo atualmente qualquer técnica que permita de forma objetiva a determinação do valor de energia a aplicar. O propósito do projeto CATARACTUS consiste precisamente em identificar a dureza da catarata medindo a energia de ultrassons que é refletida na catarata, catalogá-la em termos de algumas classes de dureza e estimar o nível de energia ótimo a usar na cirurgia de facoemulsificação com base na estimação da dureza.

2.2 Sistemas de classificação de catarata

Atualmente existem vários sistemas de classificação de catarata:

- *Age-Related Eye Disease Study* (AREDS) *System* que é uma extensão do *Wisconsin System for Classifying Cataracts* (Kassoff et al., 2001);
- WHO Cataract grading System (WHO, 2012);
- Cooperative Cataract Epidemiology Study Group (Sasaki & Sakamoto, 1997);
- Wilmer (West, Rosenthal, Newland, & Taylor, 1988);
- Japanese Cooperative Cataract Epidemiology Study Group (Sasaki, Shibata, & Obazawa, 1990);
- The Oxford Clinical Cataract Classification and Grading System (Sparrow, Bron, Brown, Ayliffe, & Hill, 1986);

- Lens Opacities Classification System Versão I,II e III (L. T. Chylack, Leske, Sperduto, Khu, & McCarthy, 1988; Leo T Chylack et al., 1989, 1993);

Alguns dos sistemas referidos baseiam-se em comparação, com um conjunto de imagens de referência, das imagens do olho do paciente obtidas através de técnicas como biomicroscopia com lâmpada de fenda e fotografia de retroiluminação por lâmpada de fenda. Resumindo, o modo como estes sistemas classificam as cataratas faz com que a classificação seja subjetiva, ficando a classificação dependente da experiência do médico (Fan, Dyer, Hubbard, & Klein, 2003).

Dos sistemas referenciados anteriormente, o LOCS III é o sistema mais utilizado para classificação da catarata (Caixinha, 2016). É um sistema que contém seis imagens de lâmpada de fenda para classificação da cor e da opacidade da catarata nuclear, cinco imagens de retroiluminação para classificação da catarata cortical e cinco imagens de retroiluminação para classificação da catarata subcapsular posterior (Figura 2.6).

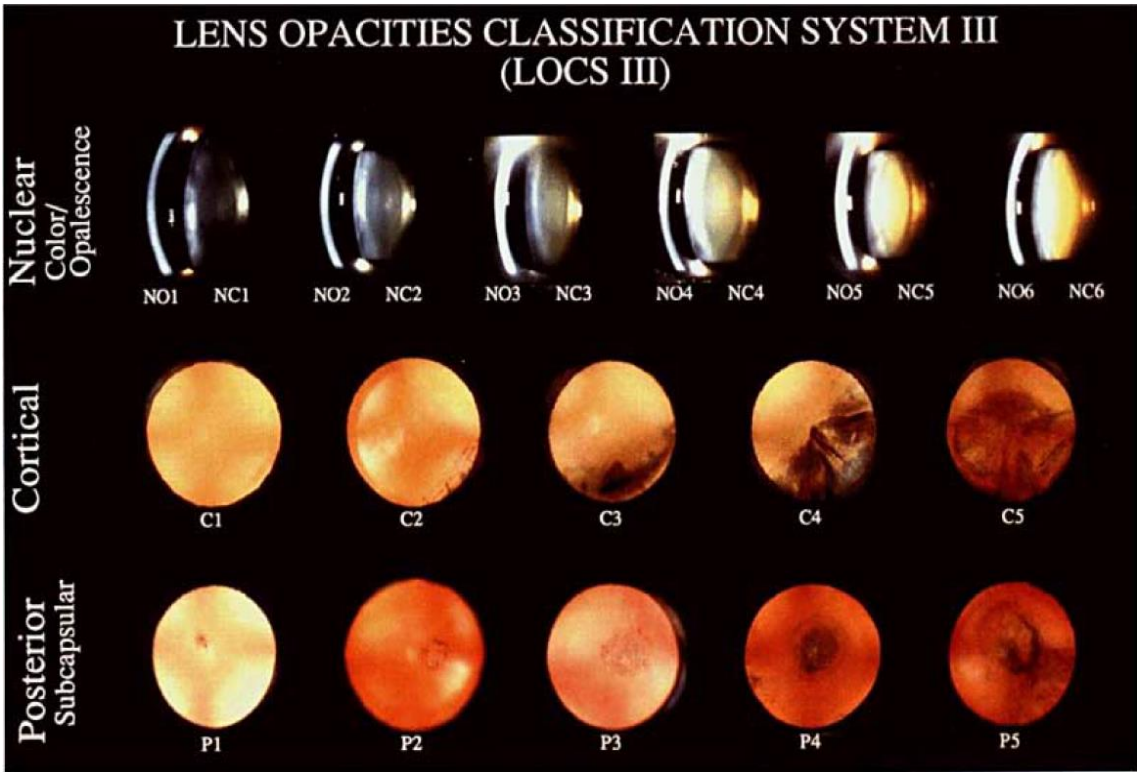

Figura 2.6 - As imagens de referência do sistema LOCS III Cor do Núcleo (NC) Opacidade do Núcleo (NO) Catarata Cortical (C) Catarata Subcapsular Posterior(P) (Leo T Chylack et al., 1993)

Algumas das vantagens para as versões anteriores do sistema LOCS é a adição de mais imagens para a classificação da cor e opacidade do núcleo para que haja uma melhor

representação das fases inicias da catarata nuclear, algo que também foi feito com o mesmo objetivo para a catarata subcapsular posterior.

Os sistemas de classificação usados atualmente, sendo subjetivos têm como principais desvantagens a variabilidade de classificação da mesma catarata por diferentes clínicos e a difícil avaliação da progressão da catarata ao longo do tempo (Fan et al., 2003). Para combater estas dificuldades tem-se vindo a apostar em sistemas de classificação automatizados (Caixinha, 2016; Fan et al., 2003; Harini & Bhanumathi, 2016; Li et al., 2010, 2008; Xu et al., 2013). Muitos destes sistemas usam como entradas imagens obtidas através de lâmpada de fenda e retroiluminação por lâmpada de fenda (Fan et al., 2003; Li et al., 2010, 2008; Xu et al., 2013). Apesar destes sistemas obterem bons resultados (Kolhe & K. Guru, 2015), a subjetividade da classificação ainda está presente devido aos parâmetros dos exames. A principal desvantagem destes sistemas é não permitirem a caracterização da dureza da catarata (Caixinha, 2016).

Atualmente existem outros sistemas de classificação de catarata que se baseiam em imagens do fundo do olho humano, como é o caso do *Fundus photography*, (Guo, Yang, Peng, Li, & Liang, 2015; Harini & Bhanumathi, 2016) cujo método de classificação se encontra esquematizado na Figura 2.7.

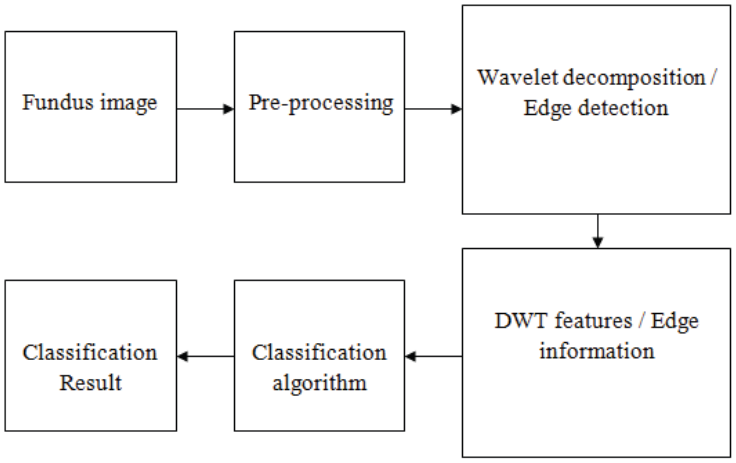

Figura 2.7 - Sistema de classificação automático de catarata utilizando imagens do fundo do olho, Fundus photography (Harini & Bhanumathi, 2016)

Em modo resumido, nestes sistemas as cataratas são classificadas de acordo com a visibilidade dos vasos sanguíneos que se encontram no olho. A desvantagem da utilização destes é a de não conseguirem caracterizar a dureza da catarata e de não detetarem cataratas em estado inicial devido a serem sistemas de classificação óticos, ou seja, numa fase inicial da catarata ainda não existe nenhuma opacidade visível, mas, no entanto, as proteínas existentes no cristalino já não se encontram corretamente alinhadas.

É com base nestas desvantagens que sistemas de deteção e classificação da dureza da catarata com base na utilização de ultrassons tem ganhado força. Estes sistemas fazem uso das reflexões que se originam quando os ultrassons transitam de um meio para o outro (meios com diferentes índices de refração). Este fenómeno pode usar-se para deteção precoce das cataratas. Quando uma catarata está numa fase inicial, ainda não existe opacidade visível no cristalino, mas forma-se uma região, não visível, com diferentes propriedades das do meio do cristalino. Devido a esta região ter um índice de refração diferente do que o meio do cristalino, quando os ultrassons incidem nesta região, são criadas reflexões com amplitude dependente da dureza da catarata. A tecnologia dos ultrassons permite, assim, a deteção da catarata e classificação da sua dureza de forma mais objetiva e precoce.

2.3 Os Ultrassons

Os ultrassons é uma onda mecânica cuja frequência é superior à frequência máxima audível por um ser humano. Como é uma onda mecânica, precisam de um meio de propagação. À medida que a onda se propaga, esta transfere energia para o meio, ou por outras palavras, a onda é atenuada. Meios diferentes provocam diferentes níveis de atenuação (Caixinha, 2016; Laugier & Haïat, 2011).

Os ultrassons são gerados por um transdutor de efeito piezoelétrico. O efeito piezoelétrico consiste na interação eletromecânica em cristais onde uma força aplicada nos seus extremos dá origem a uma carga elétrica ou onde um campo elétrico aplicado gera uma deformação mecânica. O transdutor trabalha em dois modos: transmissão e receção. Em modo de transmissão converte pulsos elétricos em ondas sonoras (ultrassons). Em modo de receção converte ondas sonoras em sinal elétrico. Uma característica dos ultrassons é que quanto maior for a sua frequência, melhor é a sua resolução espacial na deteção de alterações das características do meio onde se propagam

sendo, no entanto, pior o poder de penetração nesse meio, i.e. verifica-se um aumento da atenuação.

A intensidade de uma onda acústica ultrassons em função da profundidade, *x*, está relacionada com a intensidade em *x*=0, i.e., à saída do transdutor, e com o coeficiente de atenuação, α, de acordo com a seguinte equação:

$$I(x) = I(0)e^{-\alpha x} \quad (1)$$

Devido ao facto de a utilização de ultrassons ser uma técnica não-invasiva e considerada segura, tem vindo a conquistar o seu lugar na medicina, na medida em que permite caracterizar a estrutura e morfologia do tecido (modo B-Scan) (Paunksnis, Kurapkiene, Maciulis, Kopustinskas, & Paunksniene, 2007).

A utilização dos ultrassons na componente ocular da medicina resulta do facto do olho humano ser uma estrutura composta por fluidos que facilitam a propagação dos ultrassons e que permite facilmente visualizar a estrutura ocular (Chaudhari, Thakkar, & Gandhi, 2013). Na oftalmologia tem-se usado dois modos para visualizar a informação obtida através dos sinais de eco: A-Scan e B-Scan (Shlensky & Ringeisen, 2015).

Com A-Scan, *Amplitude Scan*, temos um gráfico 1D de acordo com (1) cujo eixo das ordenadas é a amplitude do sinal de eco recebido e o eixo das abcissas o tempo de propagação de ida e volta (i.e., emissão → reflexão → receção do eco). A amplitude do sinal depende de vários fatores tais como a propriedade do tecido nas várias interfaces existentes na estrutura (Ex: Córnea e cristalino), o ângulo de incidência do ultrassons e a densidade do meio de propagação (Caixinha, 2016; Shlensky & Ringeisen, 2015).

Com B-Scan, *Brightness scan*, temos uma imagem em 2D. Esta é construída através de diferentes A-Scans obtidos por deslocação sobre a superfície da córnea. Através do B-Scan é possível, por exemplo, extrair informação sobre a estrutura e lesões oculares, entre outras (Caixinha, 2016; Shlensky & Ringeisen, 2015).

Ao contrário dos métodos óticos de deteção de cataratas, os ultrassons são capazes de detetar a catarata num estado inicial (estado em que ainda não existe opacidade visível no cristalino, mas em que já existe alteração das propriedades acústicas do meio). Os ultrassons detetam a catarata devido a existir uma região com diferentes propriedades acústicas (catarata) do que o meio do cristalino. Quando o ultrassom incide nessa região, devido a estas terem propriedades acústicas diferentes, vão ser criadas reflexões com

amplitudes dependentes da dureza da mesma. Se estas reflexões forem capturadas, possibilitam a deteção e estimação da dureza da catarata.

Vários estudos ex-vivo feitos com cristalinos de porco, mergulhados numa solução que permite formar cataratas artificialmente, revelaram que existe uma correlação entre a velocidade de propagação dos ultrassons no cristalino e a rigidez da catarata formada no mesmo. O mesmo foi observado para o coeficiente de atenuação (Caixinha et al., 2014; El-Brawany, 2009; Huang et al., 2007; Jesus, Caixinha, Santos, & Santos, 2012; Tabandeh et al., 2000).

No projeto CATARACTUS, no âmbito do qual foi realizada esta dissertação, é utilizada uma sonda de ultrassons customizada fabricada pela Imasonic[8] cujas principais características estão mencionadas na Tabela 2.1. As características desta sonda permitem obter uma boa resolução espacial na zona de interesse, entre a cápsula anterior do posterior do cristalino. Ao mesmo tempo permite obter uma profundidade de penetração suficiente para que ocorram reflexões nas diversas interfaces do olho, incluindo o fim do mesmo.

Tabela 2.1 - Características da sonda de ultrassons (Imasonic, 2014)

Características da Sonda

Tipo	Sonda oftalmológica
Frequência central	20 MHz
Largura de banda	[4.0 MHz a 30.0 MHz]
Distância focal	8.9 mm
Largura do pulso	61.4 ns a -6 dB
Tamanho do elemento ativo	3.2 mm

[8] http://www.imasonic.com/Company/Identity.php

2.4 Sistema de aquisições de dados

Os sistemas de aquisição de dados (DAQ) são dispositivos cujo objetivo principal é adquirir informação do ambiente. Essa informação costuma ser originada a partir de um transdutor/sensor (Abdallah & Elkeelany, 2009). No caso do projeto CATARACTUS, esse sensor é uma sonda oftalmológica de ultrassons que emite pulsos acústicos e capta as reflexões.

Atualmente os dispositivos DAQ estão equipados com as seguintes capacidades/componentes (Keithley, 2001):

- Conversor analógico-digital cujo objetivo é a conversão do sinal analógico emitido pelo sensor, por outras palavras, realização da amostragem de um sinal analógico.
- Bus de comunicação com o computador. Este componente atua como interface entre o DAQ e o computador. É desta maneira que o computador tem acesso aos dados adquiridos pelo DAQ.
- Condicionamento do sinal. Muitas vezes o sinal analógico que chega ao DAQ traz ruido/interferências devido, por exemplo, à temperatura ambiente, campos elétricos externos e vibrações do sensor. Se o sinal analógico com ruido fosse amostrado, o resultado seria incorreto/impreciso. Sendo assim, antes do sinal analógico com ruido chegar ao conversor analógico-digital é necessário fazer o seu condicionamento. Isto pode consistir em filtragem do sinal, amplificação do sinal/atenuação do sinal.
- Conversor digital-analógico para gerar sinais, por exemplo para controlar atuadores como motor elétrico.
- Contador de eventos, como por exemplo contar quantos *triggers* se realizaram num dado equipamento.
- Múltiplos canais quer para receção de sinais analógicos, quer para emissão dos mesmos ou apenas para contar a ocorrência de eventos.
- Realização de aquisição de dados em múltiplos canais paralelamente.

- Níveis de tensão de entrada programáveis. Por exemplo, para o mesmo dispositivo com os seguintes possíveis níveis de tensão de entrada: ± 1V, ± 0.5V e ± 10V, podemos ter um sinal analógico que varia entre ± 1V e podemos utilizar um nível de tensão de entrada de ± 1 V. Isto garante que vamos ter uma melhor representação do sinal analógico porque caso utilizássemos o nível de tensão de entrada de ± 10V teríamos apenas a usar parte da resolução (número de bits) do dispositivo para representar o sinal analógico e não teríamos uma representação tão fiel dos valores verdadeiros do sinal analógico.

Na figura 2.8 é possível observar um diagrama de blocos de um dispositivo DAQ com múltiplos canais de entrada.

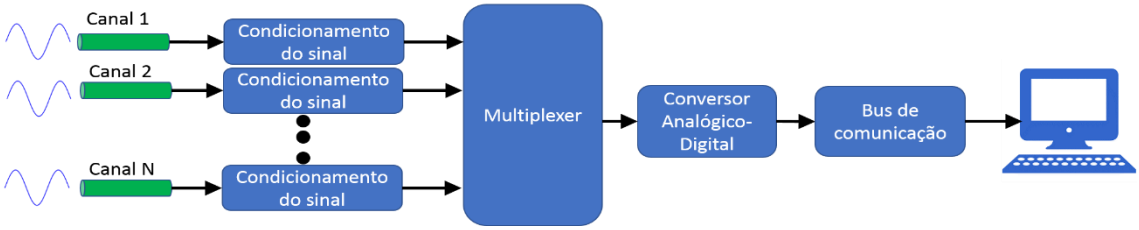

Figura 2.8 – Diagrama de blocos de um dispositivo DAQ de múltiplos canais de entrada com objetivo de fazer aquisição de sinais

Para a escolha de um dispositivo DAQ deve-se ter em conta vários parâmetros e características:

- O objetivo do sistema onde o dispositivo vai ser integrado. Se o sistema tem como objetivo fazer aquisição de dados, emitir sinais analógicos para, por exemplo, controlar um atuador ou ainda apenas para contar a ocorrência de eventos.

- Número de canais de entrada/saída. Tem de ter um número de canais suficiente para dar resposta às necessidades do sistema. Caso o objetivo do sistema seja a aquisição de dados, é preciso verificar se os canais de entrada têm largura de banda suficiente para capturar na totalidade o sinal analógico. Outro aspeto importante relacionado com os canais de entrada, é o seu nível de tensão admissível. Caso o nível de tensão de entrada do canal seja inferior ao nível de tensão do sinal analógico, este aparecerá saturado quando for visualizado no computador.

- Bus de comunicação entre o dispositivo e o computador também tem a sua importância. Atualmente existe PCIe, Ethernet, USB, Wi-Fi e muitos mais. Cada um com a sua velocidade de transferência de dados. A escolha do bus de comunicação está dependente da velocidade a que o sistema precisa de ter acesso aos dados e à quantidade de dados a enviar para o computador.

- Resolução dos dados recolhidos ou número de bits usado para representar a amostra recolhida. Quanto maior a resolução maior o número de bits disponíveis para representar a amostra.

- Frequência de amostragem. A sua escolha depende da frequência do sinal analógico de entrada. Devemos ter em conta o teorema de *Nyquist* que diz que para a reconstrução fiel de um sinal analógico é preciso fazer a amostragem a uma frequência duas vezes superior à frequência do sinal analógico.

- Erro de medida. A capacidade do dispositivo DAQ representar fielmente os verdadeiros valores do sinal analógico.

Atualmente existem dispositivos DAQ com 12 bits de resolução, com uma frequência de amostragem superior a 2 MHz, com diversos tipos de bus de comunicação com o computador e diversos canais. Na seguinte tabela pode-se consultar alguns dos dispositivos DAQ atuais juntamente com algumas das suas características (AD-LINK, 2014, 2016; AlazarTech, 2019a, 2019b).

Tabela 2.2 - Exemplos de dispositivos de aquisição de dados e algumas das suas características

Placa	PCIe-9852	PCIe-9834	PXIe-9848	ATS9350	ATS9870
Fabricante	AD-LINK	AD-LINK	AD-LINK	AlazarTech	AlazarTech
Frequência de amostragem	200 MS/s	80 MS/s	100 MS/s	500 MS/s	1 GS/s
Resolução (bits)	14	12	14	12	8
Número de canais	2	4	8	2	2
Throughput	2.0 GB/s	1.0 GB/s	1.0 GB/s	1.6 GB/s	1.6 GB/s

Memória *on-board*	1 GB	1 GB	512 MB	3.7 GB	4 GB
Largura de banda (-3 dB)	Até 90 MHz	Até 40 MHz	Até 100 MHz	Até 250 MHz	Até 450 MHz
Níveis de tensão de entrada disponíveis	± 0.2V ± 2V ± 10V	± 0.5V ± 1V ± 5V ± 10V	± 0.2V ± 2V	De ± 40 mV a ± 4V	De ± 200 mV a ± 4V

A maioria dos dispositivos DAQ trazem o seu próprio software e drivers para se poder visualizar os dados adquiridos num computador. Mas muitas vezes, o utilizador quer incorporar o dispositivo DAQ no seu sistema customizado onde o utilizador desenvolveu o seu próprio software de visualização dos dados adquiridos. Para incorporar o dispositivo neste sistema é necessário que este tenha uma *Application Programming Interface* (API) desenvolvida de modo a que o utilizador possa ter acesso aos dados adquiridos e podê-los visualizar no seu próprio software.

A questão da disponibilização dos dados adquiridos pelo dispositivo em tempo-real é de elevada importância, principalmente no contexto desta dissertação que é o do desenvolvimento de um sistema de aquisição e visualização de sinais em tempo-real. Para esse efeito, os sistemas de aquisição de dados têm à sua disposição vários tipos de bus de comunicação com um *throughput* elevado tais como o PCI-Express (PCIe) que pode ir desde 250 MB/s para a 1ª geração a 1 GB/s para a 3ª geração ou Ethernet com placas de rede capazes de suportar velocidades inerentes a aplicações *real-time*. Estes valores de *throughput* elevados, são auxiliados pela funcionalidade existente nos computadores atuais chamada *Direct Memory Access* (DMA) que, em modo muito resumido, permite escrever diretamente na memória dados sem estes terem de passar pelo CPU, o que, caso acontecesse, tornava o processo mais lento.

Para o projeto CATARACTUS é necessário um dispositivo DAQ com pelo menos um canal de entrada analógico para ser ligado à sonda oftalmológica. Como é uma sonda com largura de banda de 4.0 MHz a 30.0 MHz (Tabela 2.1) é preciso um dispositivo DAQ com frequência de amostragem superior a duas vezes 30.0 MHz, ou seja, um dispositivo

DAQ com frequência de amostragem de 60 MHz para que seja possível reconstruir o sinal analógico fielmente (Teorema de *Nyquist*). Em termos de bus de comunicação é preciso um dispositivo DAQ que seja capaz de transferir dados adquiridos em tempo-real. O *throughput* mínimo que o dispositivo DAQ deve ter é dado pela equação (2) onde é suposto uma velocidade de propagação dos ultrassons constante (v = 1500 ms $^{-1}$) e o comprimento do olho igual a 24.4 mm (d = 24.4 mm) (Gross, Blechinger, & Achtner, 2008).

$$A = B \times \left(\left\lceil \frac{2 \times d}{v \times 10^3} \right\rceil \times 10^6 \times C \times D + E\right) \quad (2)$$

A: *Throughput* máximo em MB/s

B: Valor do parâmetro PRF em Hz

C: Frequência de amostragem em MHz

D: Número de bytes usado para representar uma amostra

E: Tamanho do cabeçalho em bytes

v: Velocidade de propagação dos ultrassons no meio em ms^{-1}

d: Profundidade da aquisição em mm

Adicionalmente, o dispositivo DAQ a escolher para este projeto deverá ter um canal de sincronismo para que a aquisição dos pulsos esteja síncrona com a sua emissão, ou seja, o dispositivo DAQ tem de suportar uma fonte de *trigger* externa que virá do gerador de pulsos.

2.5 Projeto CATARATA

No âmbito do projeto CATARATA (PTDC/DTP-PIC/0419/2012) foi realizado um estudo para validar a seguinte hipótese: Possibilidade de classificar e detetar automaticamente as cataratas em humanos com recurso a ultrassons. Para validar esta hipótese foi feito um estudo inicial em animais (Projeto CATARATA). Com base nos resultados positivos deste estudo, este foi transladados para humanos, projeto CATARACTUS (POCI-01-0145-FEDER-028758) que atualmente está em curso. Adicionalmente, também com base nos resultados positivos do projeto CATARATA, foi desenvolvido um protótipo de dispositivo médico denominado ESUS.

2.5.1 Protótipo de dispositivo médico

Com base nos resultados obtidos nos estudo *Ex-Vivo* e *In-Vivo* foi desenvolvido um protótipo de dispositivo médico cujo nome é *Eye Scan Ultrasound System* ou ESUS que se encontra patenteado (PPP108836). Este dispositivo usa tecnologia de ultrassons para detetar e classificar automaticamente cataratas em tempo-real.

O diagrama de blocos do dispositivo encontra-se representado na Figura 2.9, sendo este composto por:

- Sonda de ultrassons (Sonda de 20 MHz, *Imasonic*, SAS, França);
- Gerador/recetor de ultrassons (módulo *Compact Pulser*, *Ultratek*, USA);
- Placa de aquisição de sinais com conversor AC/DC embebido (modelo FCM 150, 4DSP LLCTM, USA);
- Sistema de processamento de sinal (CPU/FPGA/DSP) (com base numa *ZeadBoard*);
- Servidor para armazenamento de dados (Computador);
- Dispositivo *wireless* para visualização dos dados.

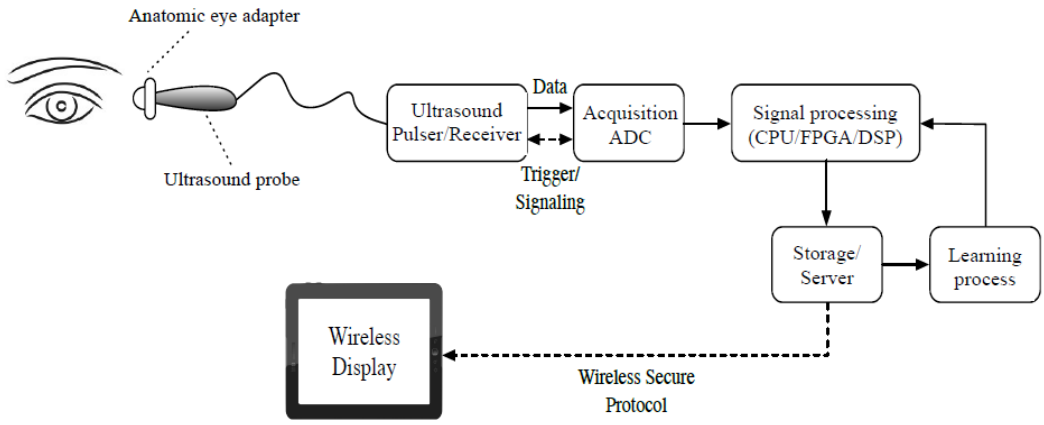

Figura 2.9 - Constituição do protótipo ESUS em blocos (Caixinha, 2016)

O sistema funciona, resumidamente, da seguinte forma: o módulo *Compact Pulser*, aquando em modo de transmissão excita a sonda oftalmológica. De seguida o mesmo módulo transita para modo de receção, onde são recebidos e amplificados os ecos

(originados pelas diferentes estruturas oculares) que são capturados pela sonda. Numa fase posterior, é feita a aquisição do sinal através do módulo *FCM 150*, ou seja, o sinal analógico é convertido em sinal digital através de um conversor analógico-digital (ADC). Com a conversão do sinal analógico para digital é realizado o processamento do mesmo que consiste, em modo resumido, em condicionamento do sinal, redução do ruído, deteção das estruturas oculares e extração das *features*. Com as *features* extraídas é feita a classificação. Tanto o resultado da classificação, bem como o A-Scan e B-Scan do sinal adquirido são mostrados ao clínico através de um dispositivo de visualização. Caso o clínico concorde com o resultado da classificação, este poderá adicionar o vetor de *features* ao *data-set* de treino para que o desempenho do classificador seja aumentado.

No âmbito desta dissertação são propostas duas evoluções do presente sistema ESUS, tendo em vista a utilização do mesmo na deteção de cataratas em humanos. As mesmas são apresentadas nos capítulos seguintes.

3. Sistema de Aquisição de Sinais com placa AD-LINK

Neste capítulo é descrito o sistema de aquisição de sinais desenvolvido para ambientes clínicos utilizando como placa de aquisição a PCIe-9842, desenvolvida pela AD-LINK. Nesta solução o módulo de excitação da sonda encontra-se separado do módulo de aquisição de dados.

Inicialmente é feita uma introdução a cada componente utilizado no sistema de aquisição de sinais onde é dado um foco especial à placa PCIe-9842. De seguida, introduzimos o método desenvolvido para aceder aos dados adquiridos pela placa e as suas limitações.

3.1 Sistema de aquisição e seus componentes

O sistema desenvolvido para aquisição de sinais em ambiente clínico, utilizando a placa PCIe-9842 para aquisição de dados, está ilustrado na figura 3.1

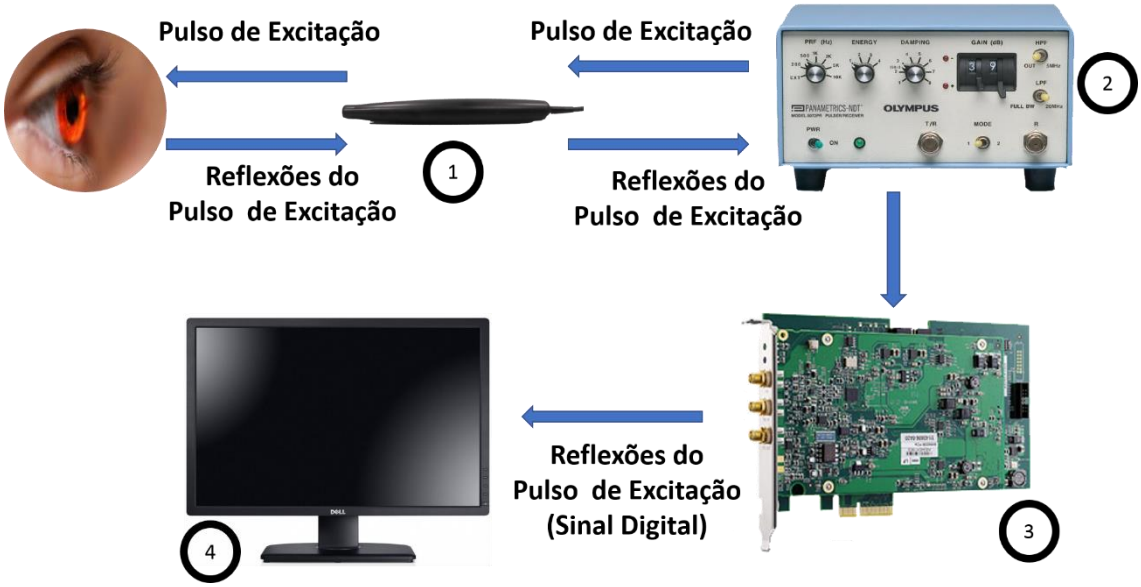

Figura 3.1 - Elementos que constituem o sistema de aquisição de sinais com a placa PCIe-9842

O sistema ilustrado é constituído por uma sonda oftalmológica customizada (identificada na figura 3.1 com número 1), com frequência central de 20 MHz desenvolvida pela *Imasonic* cujas características estão listadas na tabela 2.1 no capítulo 2.3. Esta sonda funciona em modo pulso-eco, ou seja, a mesma sonda transmite pulsos de ultrassons e a seguir recebe as reflexões. Temos a fonte de excitação/receção modelo

5072PR da *Panametrics*[9] (identificado na figura 3.1 com o número 2) que é responsável pela geração de pulsos de excitação que são enviados pela sonda de ultrassons e também pela receção e amplificação das reflexões capturadas pela sonda. O elemento seguinte do sistema de aquisição de sinais é a placa PCIe-9842 da *AD-LINK* (identificada com o número 3) que tem como objetivo realizar aquisição de dados, neste caso, fazer a aquisição das reflexões que foram capturadas pela sonda. Por fim, temos um computador (identificado com o número 4), onde é possível visualizar os sinais adquiridos em tempo-real através de um software especialmente desenvolvido para esse fim, tendo em consideração os requisitos de utilização em ambiente clínico.

3.2 Placa PCIe-9842

A placa PCIe-9842 foi desenvolvida pela AD-LINK. É uma placa de aquisição de dados de alta frequência de amostragem (200 MHz) com 14 bits de resolução por amostra. As suas características estão listadas na tabela 3.1.

Tabela 3.1 - Características da placa PCIe-9842

Característica	**Valores**
Frequência de amostragem	Até 200 MHz
Resolução (bits)	14
Número de canais	1
Largura de banda da entrada (-3 dB)	100 MHz
Intervalo de valores para sinal de entrada	±1.0 V
Trigger externo	Sim
Compatibilidade do *trigger*	3.3 V
Largura mínima do pulso de *trigger*	20 ns
Tipo de *trigger*	*Post-Trigger* e *Delay-Trigger*
Condição de *trigger*	*Raising* ou *falling edge*
Bus de comunicação	PCIe *First Gen*
Throughput	1 GB/s

Na implementação realizada, o sinal de entrada da placa PICe-9842 e o seu trigger provêm do módulo *Panametrics*. A utilização do *trigger* proveniente do *Panametrics* permite que o início da aquisição esteja síncrono com a geração dos pulsos de excitação (Figura 3.2), evitando realizar um processo de sincronismo dos sinais após aquisição (Figura 3.3).

[9] https://www.olympus-ims.com/en/.downloads/download/?file=285213012&fl=en_US

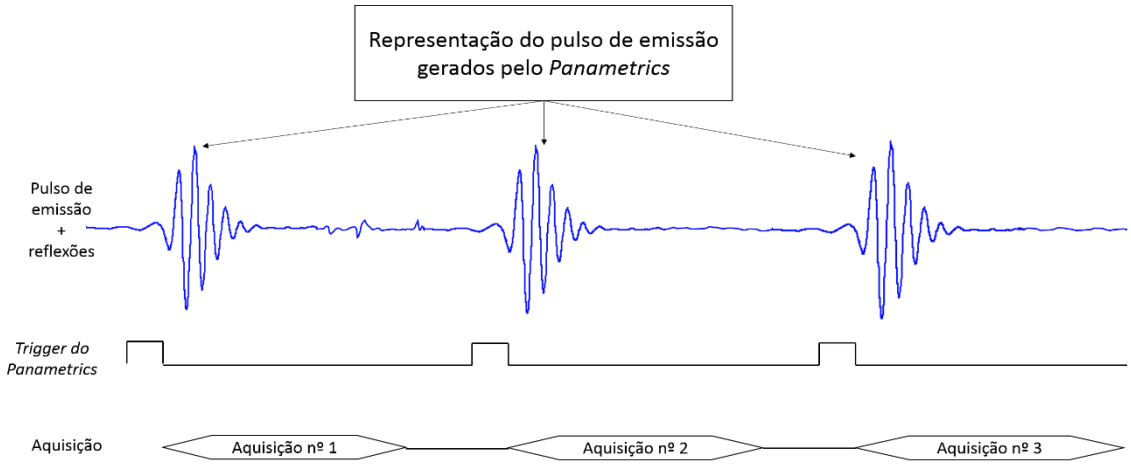

Figura 3.2 - Esquema de aquisição quando se usa o *trigger* vindo do dispositivo *Panametrics*

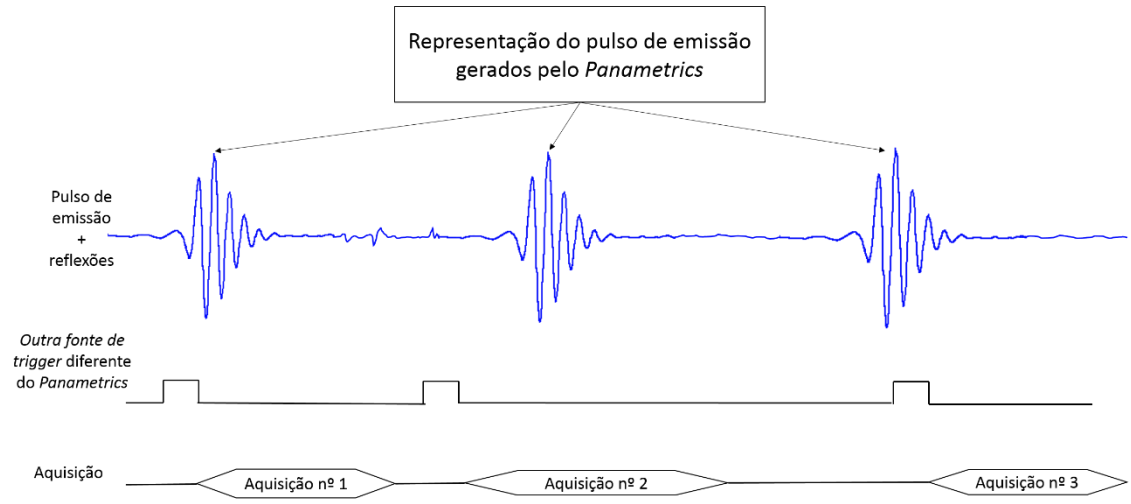

Figura 3.3 - Esquema de aquisição quando se usa uma outra fonte de trigger diferente do Panametrics

A utilização desta placa no sistema de aquisição de sinais, permitiu observar um conjunto de vantagens e desvantagens. As vantagens consistem na sua elevada frequência de amostragem, na possibilidade de usar um *trigger* externo para realizar a aquisição de dados, num elevado *throughput* através da utilização PCIe e DMA e numa resolução de 14 bits. As desvantagens de utilização da placa são as de que esta necessita de um computador com entrada PCIe disponível e de um gerador/recetor de pulsos autónomo.

3.3 Comunicação entre a placa e o computador

Para aceder aos dados adquiridos pela placa para posterior processamento (condicionamento e análise) e visualização pelo utilizador, foi desenvolvido um programa em C++ usando o ambiente de desenvolvimento integrado (IDE) da *Microsoft Visual Studio* cujo nome é *startAcquisitionLive*.

Este programa foi desenvolvido com o auxílio de uma API do software *DAQPilot* que é uma API para arquiteturas x86 de 32 bits. Isto ainda limita mais a possibilidade de usar o sistema de aquisição de sinais noutras plataformas.

Na tabela 3.2 é possível consultar os argumentos de entrada do programa, cada um com valores por omissão caso não sejam especificados:

Tabela 3.2 - Argumentos de entrada do módulo desenvolvido com o auxílio da API do DAQPilot

***Flag* do argumento de entrada**	**Informação do argumento de entrada**	**Valores válidos**	**Valor por omissão**
-h	*Help* do programa		
-v	Escolha do *virtual device*		
-b	Escolha da placa PCIe-9842		
-f	Escolha da frequência de amostragem em Hz	0 a 200×10^6	100000 para o *virtual device* 200×10^6 para a placa PCIe-9842.
-n	Número de amostras adquiridas, *N*, por cada aquisição de sinais	Inteiro sem sinal de 4 bytes	10000 para o *virtual device;* 200×10^6 para a placa PCIe-9842.
-t	Tipo de *trigger*	0 => *Trigger* Interno*;* 1 => *Trigger* software 2 => *Trigger* externo	0 => Para o *virtual device* e placa PCIe-9842
-tm	Modo de *trigger*	0 => *Post trigger* 1 => *Delay trigger*	0 => Para o *virtual device* e placa PCIe-9842
-dl	Quantas amostras de atraso (*trigger delay*)	Inteiro sem sinal de 4 bytes	0 => Para o *virtual device* e placa PCIe-9842
-lf	*Log file* criado pela API do DAQPilot	Vetor de caracteres (*char array*)	Vetor de caracteres vazio para o *virtual device* e placa PCIe-9842

-df	Formato dos dados	0 => *Scaled data*	2 => Para o *virtual device* e placa PCIe-9842
		1 => *Scaled Data in 2D array*	
		2 => *Binary codes*	
		3 => *Binary codes in 2D arrays*	
-p	Nome do ficheiro binário de dados, *binaryFile*.	Vetor de caracteres (*char array*)	"default.cat" => Para o *virtual device* e placa PCIe-9842
-bf	Tamanho do *buffer* do ficheiro *binaryFile* em nº de aquisições de *N* amostras, *M*.	Inteiro de 4 bytes	20 => Para o *virtual device* e placa PCIe-9842

O ficheiro, *binaryFile*, tem um comportamento de *buffer* circular, onde vão existir continuamente as *M* aquisições mais recentes. Este valor de *M* é especificado pelo argumento de entrada com *flag* -bf.

O funcionamento deste programa é descrito nos seguintes passos:

1. Inicialização e configuração de um objeto "*task*" tal como é definida pela API DAQPilot.

    - Uma "*task*" corresponde a uma aquisição de sinal e tem parâmetros de configuração tais como a frequência de amostragem, o número de amostras a adquirir por pulso de *trigger*, o tipo de *trigger* e o modo de *trigger*;

    - Cada lançamento da "*task*" gera uma aquisição de *N* amostras. No presente caso é usado o valor de *N*=6000, o que a 200 MHz de frequência de aquisição corresponde a 30μs ou cerca de 2.25 cm para uma velocidade de propagação do som de 1500 ms^{-1}. Este tempo de aquisição, 30μs, é suficiente para capturar as zonas de interesse do olho, desde o inicio do olho até à capsula posterior do cristalino (Gross et al., 2008). Por omissão, *M*=20 aquisições, o que, para um *Pulse Repetition Frequency* (PRF) de 1 kHz existem no ficheiro *binaryFile* 20 ms de sinal continuamente atualizados.

2. Abertura/Criação do ficheiro *binaryFile*.

3. Execução da "*task*" /realização da aquisição de *N* amostras

    - Em cada aquisição são recolhidas *N* amostras. Este valor é imposto pelo argumento de entrada com *flag* -n. Como cada amostra é representada por um short (2 bytes), então vamos ter uma aquisição de $2 \times N$ bytes, sendo *N* o número de amostras adquiridas.

4. Escrita de um cabeçalho por cada aquisição realizada.

    - Em cada aquisição realizada é escrito um cabeçalho com o seguinte conteúdo:

        - Um *timestamp* relativo ao início do programa com formato inteiro de 32 bits. Permite saber qual a aquisição mais recente;

        - O número de amostras adquiridas no formato inteiro de 32 bits. Este campo contém o valor de ***N***;

        - A posição relativa da aquisição no ficheiro pipeline no formato inteiro de 32 bits. Este campo contém um número que está no intervalo de 1 a *M* (valor do argumento de entrada com *flag* -bf);

        - O número de aquisições que podem ser guardadas no ficheiro *binaryFile* no formato inteiro de 32 bits, *M*.

5. Escrita das amostras recolhidas no ficheiro *binaryFile* em formato *short.*

    - Depois de realizada a escrita do *buffer* de memória para o ficheiro, é feito *flush* para o ficheiro para que as amostras fiquem disponíveis imediatamente para leitura por parte de outro programa;

    - Caso a posição relativa da aquisição que acabou de ser escrita no ficheiro seja igual a *M*, então o ponteiro de escrita do ficheiro é posto no início do mesmo e a variável que tem a informação da posição relativa da aquisição é reiniciada a 1. Assim, temos implementado um *buffer* circular no ficheiro binário de dados.

6. Verificação de paragem do processo de aquisição circular. Para parar o lançamento de nova "*task"* /realização de novas aquisições, o utilizador deve clicar no botão central do rato. Caso contrário, volta a executar-se o 4º passo.

Na figura 3.4 é possível observar o fluxograma do programa explicado anteriormente.

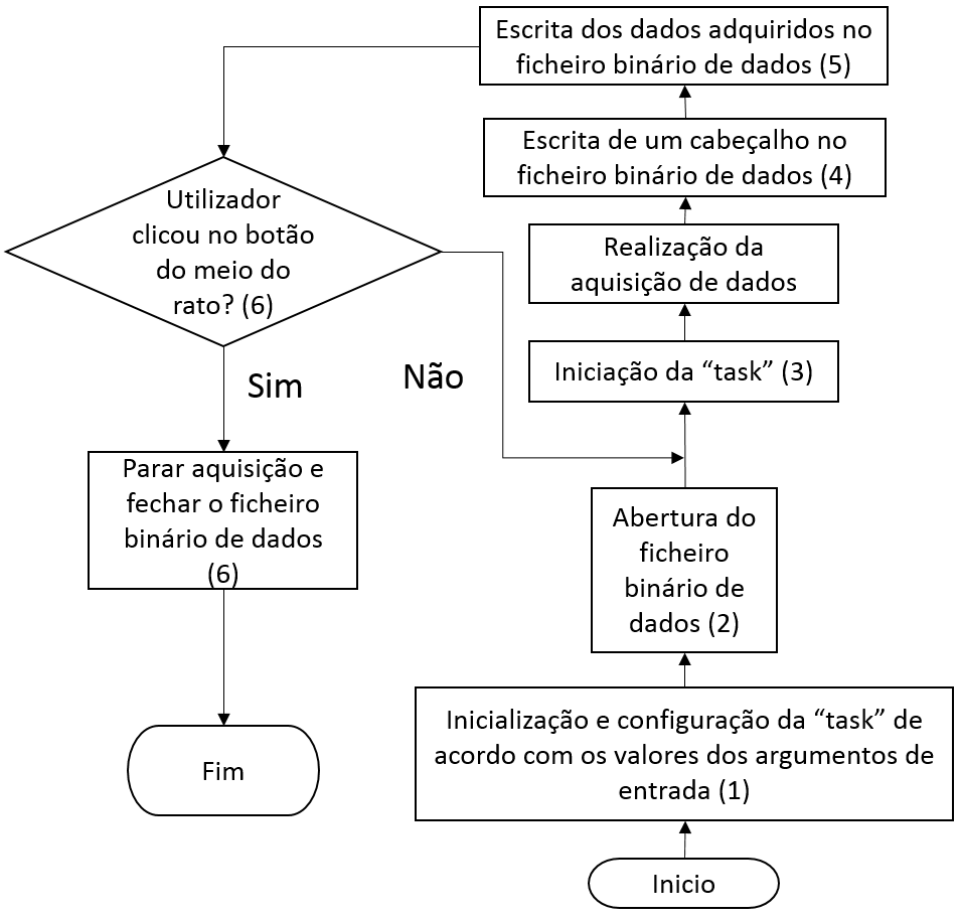

Figura 3.4 - Fluxograma do programa desenvolvido para extração dos dados adquiridos pela placa PCIe-9842

Para que um software de visualização de sinais tenha acesso às aquisições realizadas basta abrir o ficheiro *binaryFile* e iniciar o processo de leitura de acordo como o ficheiro está a ser escrito (consultar passo 5º e 6º do método descrito anteriormente).

A escrita e leitura dos dados no ficheiro *binaryFile* é feita de modo independente, isto é, em *threads* de execução diferentes. Ou seja, o programa aqui definido escreve os dados no ficheiro a um ritmo fixo, por omissão a 50 vezes por segundo (de 20 em 20ms), supondo um PRF de 1 kHz e $M = 20$, e outro programa concorrente pode ler os dados do ficheiro e atualizar um display do sinal adquirido.

Em ambiente Matlab foi também criado um programa a fim de mostrar em tempo real o sinal de ultrassons adquirido. O fluxograma deste programa pode ser visualizado na figura 3.5. Para atualizar o gráfico do sinal basta ler o ficheiro binário de dados criado pelo programa de aquisição descrito anteriormente (Figura 3.4) e atualizar o gráfico com, por exemplo, a 1ª aquisição presente no ficheiro. Sabendo que estes dados são atualizados

a um ritmo de 50 vezes por segundo (para *PRF* = 1 kHz e *M* = 20), basta limitar a taxa de atualização dos gráficos a este ritmo. Isso pode ser feito especificando o comando Matlab "drawnow limitrate". que limita o "refresh rate" dos gráficos a 20 tramas por segundo. Neste caso poderá ser especificado o valor de *M*=50, o que faz com que o ficheiro fique com 50 aquisições que se atualizam ao ritmo de 20 por segundo.

Para parar o processo de aquisição basta usar o botão central do rato, ficando o gráfico estático uma vez que os dados não são mais atualizados.

Este processo de usar um ficheiro de dados para guardar os dados em tempo real tem a vantagem de se tornar independente do programa de apresentação gráfica dos dados.

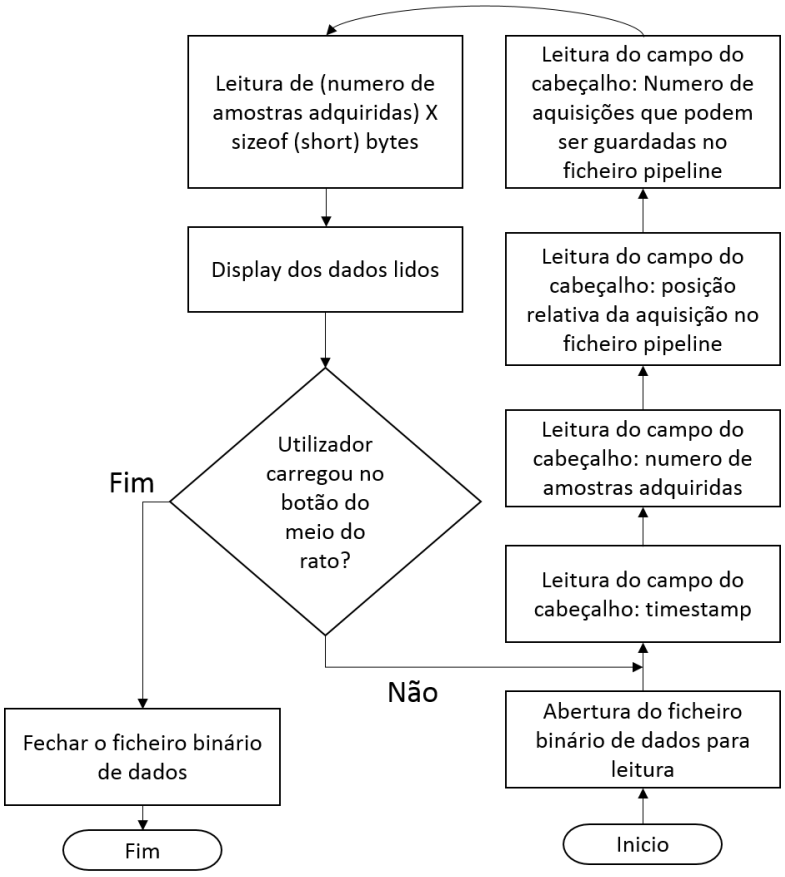

Figura 3.5 - Exemplo de um fluxograma de um script em Matlab de leitura e visualização dos dados guardados no ficheiro binário de dados

4. Sistema de Aquisição de Sinais com xSCAN

Neste capítulo é descrito a evolução do sistema ESUS implementado com base no dispositivo xSCAN desenvolvido pela Tribosonics[10] que integra o módulo de excitação da sonda e o módulo de aquisição de dados num único equipamento.

Inicialmente será feita uma introdução ao sistema de aquisição de sinais e a cada um dos seus componentes. Será igualmente feita uma comparação deste sistema com o sistema de aquisição de sinais com a placa PCIe-9842 da AD-LINK. Numa fase posterior, é feito uma introdução mais profunda ao xSCAN onde são apresentadas as suas características, capacidades, vantagens e desvantagens da sua utilização. Apresentamos o mecanismo/protocolo de comunicação do xSCAN com um computador que inclui a troca de comandos e dados entre os dois elementos. Posteriormente é feito uma introdução à API desenvolvida para a troca de comandos e dados entre o xSCAN e um computador. Serão discutidas as características dessa API, o seu desenvolvimento e as suas limitações. Posteriormente, é introduzido e discutido o programa desenvolvido para incorporar o xSCAN no sistema de aquisição de sinais de maneira a que o software de visualização tenha acesso aos sinais adquiridos pelo xSCAN. Por fim, introduzimos a estrutura do ficheiro onde os dados adquiridos são guardados para que mais tarde o utilizador tenha acesso aos mesmos.

4.1 Sistema de aquisição de sinais e os seus componentes

O sistema desenvolvido para aquisição de sinais em ambiente clínico utilizando o xSCAN como gerador de sinais e placa de aquisição está ilustrado na figura 4.1.

Este sistema é constituído por uma sonda oftalmológica customizada (identificada na figura 4.1 com o número 1). Esta é a mesma sonda descrita no capítulo 3.1. O elemento seguinte do sistema é o xSCAN da Tribosonics (identificado na figura 4.1 com o número 2). Este elemento gera os sinais elétricos de excitação que são enviados para a sonda e também tem a capacidade de receber e amplificar as reflexões que a sonda captura. Adicionalmente, faz a aquisição (conversão analógico-digital) das reflexões amplificadas.

[10] https://www.tribosonics.com

Por último, temos um computador, (identificado na figura 4.1 com o número 3), onde é feita a visualização em tempo real dos dados adquiridos pelo xSCAN.

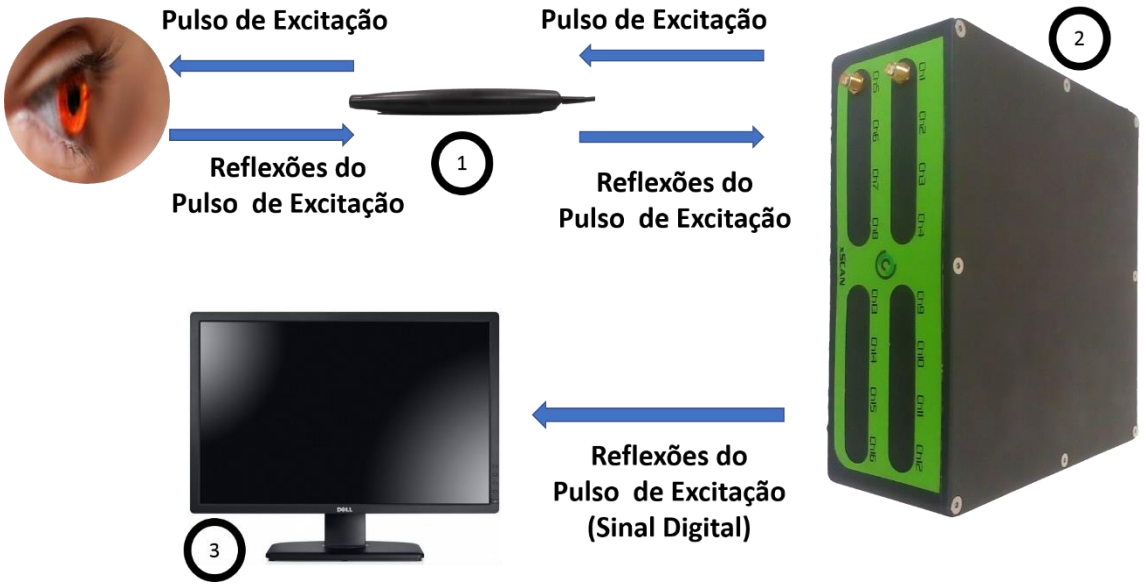

Figura 4.1 - Elementos que constituem o sistema de aquisição de sinais com o xSCAN

4.2 xScan

O xSCAN é um dispositivo cujo diagrama de blocos pode ser consultado na figura 4.2. É capaz de gerar pulsos de excitação com diferentes configurações e receber sinais elétricos. Adicionalmente, também consegue fazer a aquisição dos sinais elétricos recebidos. As duas primeiras funcionalidades deste dispositivo mencionadas equivalem ao trabalho realizado pelo dispositivo *Panametrics* enquanto a terceira e última funcionalidade equivale ao trabalho realizado pela placa PCIe-9842, que são usados no sistema de aquisição de sinais discutido no capítulo 3. Sendo assim, xSCAN é um dispositivo capaz de realizar três trabalhos diferentes enquanto que no sistema de aquisição de sinais com placa PCIe-984 eram precisos dois dispositivos independentes para os realizar.

As características do xSCAN podem ser consultadas na tabela 4.1. Os parâmetros de configuração do xSCAN e os respetivos valores válidos encontram-se descritos na tabela 4.2.

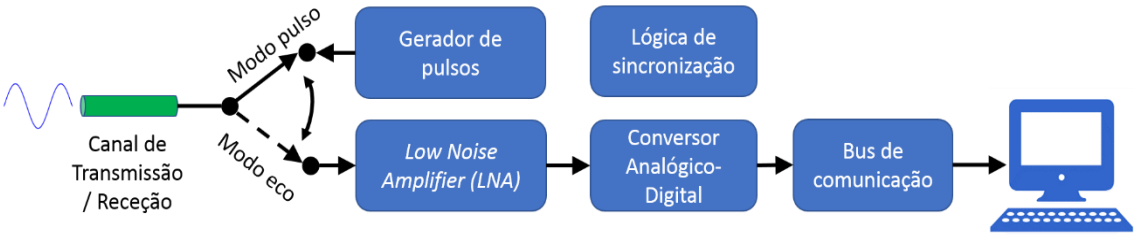

Figura 4.2 - Diagrama de blocos do dispositivo xSCAN

Tabela 4.1 - Características do xSCAN

Característica	**Valor**
Frequência de Amostragem	100 MHz
Número de canais	2
Bus de comunicação	*Ethernet*
Trigger Interno	
Resolução (bits)	14
Modos dos canais	*Pulse-echo* e *Pitch-catch*
Throughput	Dependente do computador 1000Base-T: Até 20 MB/s; 100Base-TX: Até 11 MB/s;
Largura de banda da entrada	500 kHz a 20 MHz (-3 db)

Tabela 4.2 - Parâmetros do xSCAN que podem ser configurados e os respetivos valores validos

Parâmetro	**Valores válidos**
PRF (Pulse Repetition Rate)	1 Hz a 100 kHz
Pre Amplifier Filter	*Off;* *Thru;* 0.25 MHz a 1 MHz; 0.75 MHz a 2 MHz; 1.75 MHz a 4 MHz; 2.5 MHz a 5 MHz; 4.5 MHz a 9 MHz; 6 MHz a 12 MHz; 10 MHz a 20 MHz;
Post Amplifier Filter	*Off;* *Thru;* 0.25 MHz a 1 MHz; 0.75 MHz a 2 MHz; 1.75 MHz a 4 MHz; 2.5 MHz a 5 MHz; 4.5 MHz a 9 MHz; 6 MHz a 12 MHz; 10 MHz a 20 MHz;
TX-Channel (canais do xSCAN)	*Off;* 1; 5;
RX-Channel (canais do xSCAN)	*Off;* 1; 5;

Pulse Type	Pulso retangular positiva; Pulso retangular negativa; Pulso bipolar positiva-negativa; Pulso bipolar negativa- positiva; Sequência de pulsos retangulares positivos; Sequência de pulsos retangulares negativos; Sequência de pulsos bipolares positivo-negativo; Sequência de pulsos bipolares negativo-positivo;
Pulse Width	Em múltiplos de 10 ns. Desde 0 a 1000 ns
Pulse Clamp	Em múltiplos de 10 ns. Desde 0 a 10000 ns
Pulse Train Iterations	De 0 a 100. Nota: Apenas disponível para os seguintes valores do parâmetro tipo de pulso: • Sequência de pulsos quadrados positivos; • Sequência de pulsos quadrados negativos; • Sequência de pulsos quadrados bipolares positivo-negativo; • Sequência de pulsos quadrados bipolares negativo-positivo;
Low Noise Amplifier	Ligado ou desligado
Amplifier Gain	De 0 a 80 dB
Acquisition Delay (após pulso de emissão)	De 0 a 100 µs
Acquisition Range (tempo de aquisição)	De 1 a 5000 µs
Step (número de aquisições a serem realizadas em paralelo)	De 1 a 16

Sendo cada *Step* uma aquisição a ser realizada, o equivalente a uma *task* definida no capítulo 3.3, podendo existir até 16 aquisições a serem realizadas ao mesmo tempo.

Parâmetros como o PRF, *Pre Amplifier Filter* e *Post Amplifier Filter* são comuns para todos os *Steps*. Os restantes parâmetros podem ser configurados de maneira diferente para cada um dos *Steps* ativos (aquisição a ser realizada). Como o PRF é um parâmetro com valor único para todos os *Steps*, este é divido de forma igual para cada *Step* ativo. Ou seja, cada *Step* ativo recebe uma porção de pulsos de excitação por segundo igual. Sempre que o xSCAN é ligado à energia, este arranca com valores por omissão para cada parâmetro e não com o último valor configurado.

O xSCAN utiliza comunicação *Ethernet* para enviar os dados adquiridos para um computador. Assim, o débito máximo *throughput* é dependente da placa de rede do computador. Sendo assim, para uma aplicação em tempo-real é preciso ter uma placa de rede capaz de suportar o débito necessário, que tem a ver com os parâmetros *PRF* e *Range*, isto é, o número de pulsos de emissão por segundo e a quantidade de dados adquiridos

por cada pulso de emissão. Foi verificado durante os testes que se o *throughput* excede-se os 20 MB/s, o xSCAN interromperia as comunicações com o computador devido a limitação do próprio xSCAN. Apesar desta limitação, o valor máximo de *throughput* é superior ao valor de *throughput* necessário para visualizar os sinais adquiridos em tempo-real. Este valor é calculado de acordo com a equação (4) supondo uma velocidade de propagação dos ultrassons no meio constante, igual a 1500 ms^{-1}. Para o nosso sistema de aquisição de sinais temos PRF = 1000 Hz e d = 22,5 mm (Gross et al., 2008), visto que queremos visualizar no sinal o inicio do olho até pelo menos à capsula posterior do cristalino. Sendo assim, obtemos um *throughput* máximo arredondado igual a 7 MB/s o que é bastante inferior ao limite máximo suportado pelo xSCAN.

$$A = B \times \left(\left\lceil \frac{2 \times d}{v \times 10^3} \right\rceil \times 10^6 \times C \times 2 + D\right) \quad (4)$$

A: *Throughput* máximo em MB/s

B: Valor do parâmetro PRF em Hz

C: Frequência de amostragem em MHz

D: Tamanho do cabeçalho em bytes

v: Velocidade de propagação dos ultrassons no meio em ms^{-1}

d: Profundidade da aquisição em mm

Adicionalmente, a utilização da *Ethernet* como meio de troca de dados, permite ao xSCAN ser utilizado em qualquer plataforma que contenha uma placa de rede, ao contrário da placa PCIe-9842 que precisa de uma entrada PCI Express para poder haver troca de dados com o computador.

Durante a utilização do xSCAN foram verificadas/identificadas as seguintes vantagens e desvantagens na sua utilização no contexto deste trabalho que são apresentadas na tabela 4.3.

Tabela 4.3 - Vantagens e desvantagens da utilização do xSCAN

Vantagens	**Desvantagens**
Capacidade de gerar pulsos de excitação, captar sinais elétricos e fazer aquisição dos mesmos	Metade da frequência de amostragem da placa PCIe-9842
Possível de ser utilizado em diversas plataformas devido à utilização de *Ethernet* como meio de comunicação	Menor *throughput* quando comparado com a placa PCIe-9842
Múltiplos parâmetros de configuração, quer para aquisição de sinais como para a geração de pulsos de excitação	Necessário configurar placa de rede do computador para utilizar o xSCAN

Possibilidade de realizar múltiplas aquisições em paralelo. Cada uma com uma configuração única	Não guarda a última configuração dos parâmetros realizada quando é desligado da energia.

4.3 Comunicação entre o computador e o xSCAN

O xSCAN utiliza comunicação *Ethernet* como meio para enviar os dados adquiridos para o computador. Uma vez que este sistema xSCAN é novo e se encontra em modo de desenvolvimento pela Tribosonics, o modo de comunicação não estava documentado no início deste trabalho e teve de ser inferido usando um programa de análise de protocolos (Wireshark). Descreve-se seguidamente as suas principais características.

Como protocolo de camada de transporte utiliza o *Transmission Control Protocol* (TCP). Adicionalmente, xSCAN tem 2 portos de comunicação abertos, o porto 2000 que é denominado o porto de configuração e o porto 3000 que é denominado o porto de aquisição.

O porto de configuração é usado para trocar comandos, SET e GET. Os comandos SET são utilizados quando se quer configurar um parâmetro com um novo valor, como por exemplo configurar o parâmetro PRF com um valor de 100 Hz. Já os comandos GET são utilizados para se conhecer o valor atual de um dado parâmetro, por exemplo, saber o valor atual do parâmetro *Pulse Width*. Para cada tipo de comando enviado, o xSCAN responde com o valor atual desse parâmetro ou com uma mensagem de erro. A arquitetura de comunicação entre o porto de configuração do xSCAN e o computador é uma arquitetura cliente-servidor onde o cliente é o computador e o servidor é o xSCAN. Isto significa que o servidor, xSCAN, só envia mensagens para o cliente, computador, quando este lhe faz um pedido.

O porto de aquisição é usado para o xSCAN enviar os dados adquiridos, A-Scans, para o computador. A criação de uma ligação entre este porto e o computador é usado pelo xSCAN para iniciar a aquisição dos sinais. O fluxo de comunicação desta ligação é feito unicamente no sentido xSCAN → Computador, não tendo em conta as mensagens inerentes ao protocolo TCP.

Nas figuras 4.3 e 4.4 temos um diagrama que exemplifica uma sessão de comunicação entre o computador e o porto de configuração do xSCAN e uma sessão de comunicação entre o computador e o porto de aquisição do xSCAN respetivamente.

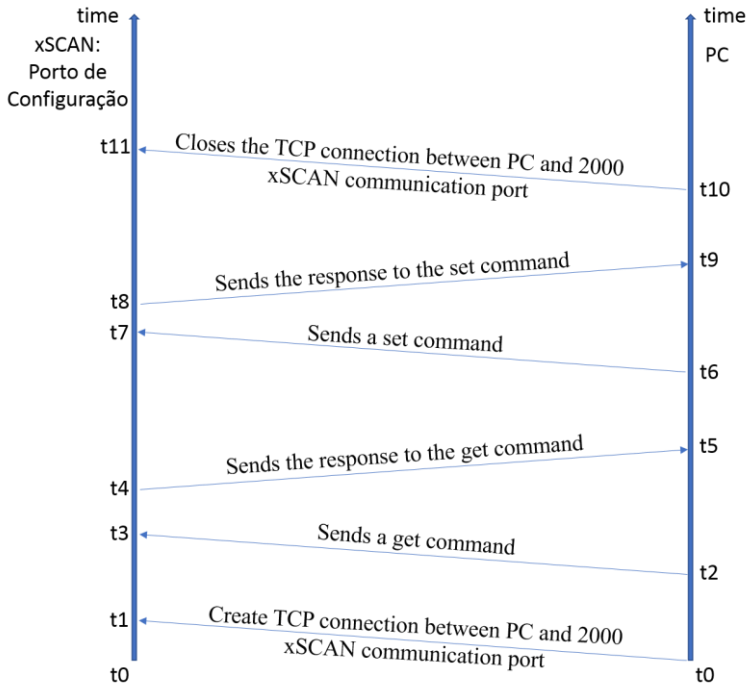

Figura 4.3 - Diagrama de comunicação entre o computador e o porto de configuração do xSCAN

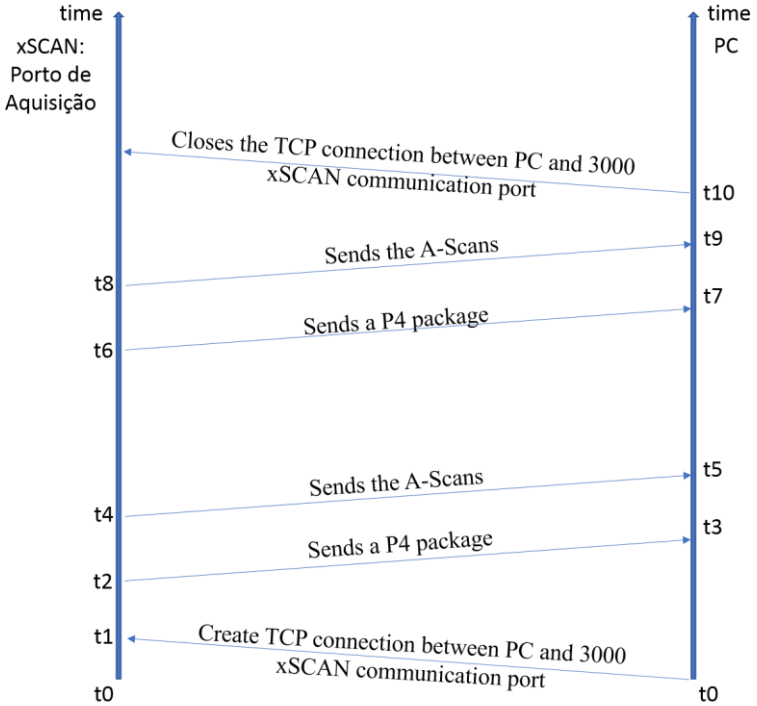

Figura 4.4 - Diagrama de comunicação entre o computador e o porto de aquisição do xSCAN

Na figura 4.3 temos os seguintes acontecimentos/eventos:

1. Em **t0**, temos a criação de uma ligação TCP entre o computador e o porto de configuração do xSCAN;
2. Em **t2**, o computador envia um comando GET para o xSCAN. O conteúdo deste comando está em formato *JavaScript Object Notation* (JSON);
    - Exemplo do conteúdo do comando GET para obter o valor atual do parâmetro PRF:
        - {“CMD”: “Get PRF”}*;
        - O símbolo ‘*’ serve para indicar ao computador e ao xSCAN o fim do conteúdo da resposta ou comando enviado respetivamente.
3. Em **t4**, o xSCAN responde ao comando GET enviado.
    - 2 possíveis respostas ao comando GET:
        - Uma mensagem de erro a dizer que o conteúdo do comando GET estava incorreto;
        - Valor atual do parâmetro pedido.
    - Conteúdo da resposta vem em formato JSON.
4. Em **t6**, o computador envia um comando SET;
    - Conteúdo deste comando e a sua resposta estão em formato JSON.
    - Exemplo de um comando SET para configurar o parâmetro PRF com um valor de 1000 Hz:
        - {“CMD”: ”Set PRF”,”Data”:1000}*.
5. Em **t8**, o xSCAN envia a resposta ao comando SET.
    - 2 respostas possíveis do xSCAN ao envio de um comando SET:
        - Uma mensagem de erro a dizer que o conteúdo do comando SET estava incorreto;
        - Valor com que o parâmetro ficou depois da configuração.

- Poderá ser ou não ser o valor que foi enviado no comando SET consoante este esteja no intervalo/conjunto de valores admissíveis para esse parâmetro.

6. Em **t10**, o computador fecha a ligação existente com o porto de configuração do xSCAN.

Na figura 4.4, em t0 temos a criação de uma ligação TCP entre o computador e o porto de aquisição do xSCAN. Como mencionado anteriormente, assim que o processo de criação da ligação TCP com o porto de aquisição é concluído, instante t1, é iniciado pelo xSCAN o processo de aquisição de dados. Em t2, um pacote é enviado pelo xSCAN denominado aqui por P4. Esse pacote contém uma *payload* de 4 bytes cujo conteúdo indica o número de bytes de informação útil, neste caso A-Scans, que irão ser enviados a seguir a este pacote. Se dividirmos este número pelo tamanho de um A-Scans, incluindo um cabeçalho de 12 bytes, em bytes obtemos o número de A-Scans enviado pelo xSCAN. Esta relação é descrita matematicamente pela seguinte equação (5).

$$A = \frac{B}{C \times D \times 2 + E} \tag{5}$$

A: Número de A-Scans
B: Número bytes indicado no pacote P4
C: Range(μs)
D: Frequência Amostragem (MHz)
E: Tamanho em bytes do cabeçalho

Como se pode observar pelo denominador, o tamanho de um A-Scan em bytes depende do valor do parâmetro *Range* em μs, da frequência de amostragem em MHz (a frequência de amostragem do xSCAN é constante: 100 MHz). A multiplicação por 2 advém de cada amostra ser representada por um short (2 bytes). Em t4, o xSCAN envia os A-Scans adquiridos. Como mencionado, a cada A-Scan está associado um cabeçalho de 12 bytes. A constituição do cabeçalho pode ser consultada na tabela 4.4.

Tabela 4.4 - Composição do cabeçalho do A-Scan

Informação do cabeçalho	**Bits (0-95)**
Valor do parâmetro *TX-Channel* (0,1 ou 5)	0-4
Valor do parâmetro *RX-Channel* (0,1 ou 5)	5-9
Número do *Step* (1-16)	10-13
Flag (bit fixo a 1)	15
Preâmbulo de informação ordem: • *Range* (bit 16) • *Delay* (bit 17) • Dados adquiridos (bit 18) • Bits 19-31 (bits fixos a zero)	16-31 Nota: Os 3 primeiros bits dizem respeito ao parâmetro *Range*, parâmetro *Delay* e aos dados adquiridos. Os restantes bits estão a zero
Range (valor está em unidades de 10 ns, ou seja, em número de amostras)	32-47 (15 bits mais significativos do valor do parâmetro *Range* em unidades de 10 ns) 48-63 (15 bits menos significativos do valor do parâmetro *Range* em unidades de 10 ns)
Delay (valor está em unidades de 10 ns, ou seja, em número de amostras)	64-79 (15 bits mais significativos do valor do parâmetro *Delay* em unidades de 10 ns) 80-95 (15 bits menos significativos do valor do parâmetro *Delay* em unidades de 10 ns)

Passado em média 100 ms, t6, o xSCAN volta a enviar um pacote P4. Caso o valor do parâmetro PRF ou do parâmetro *Range* tenha sido alterado entre t4 e t6, então o valor contido no pacote P4 em t6 será diferente do valor contido no pacote P4 em t2. Em t10, o computador encerra a ligação TCP com o porto de aquisição do xSCAN.

4.4 API desenvolvida para o xSCAN

O xSCAN vem equipado com software proprietário para uso deste equipamento como se tratasse de um instrumento (osciloscópio). Contudo, para usar o xSCAN como dispositivo médico é necessário a existência de uma API para configurar e usar este dispositivo de acordo com as necessidades. Dado que o xSCAN não traz uma API que permita a sua integração no nosso sistema de aquisição de sinais, foi necessário desenvolver uma API especifica para esse efeito. Esta API foi desenvolvida com base no processo de comunicação Ethernet descrito no capítulo anterior.

A API está definida como uma classe e foi desenvolvida na linguagem de programação C no IDE da *Microsoft Visual Studio*. Está disponível para arquiteturas x86 e x64.

A API foi construída para responder a cinco necessidades:

1. Configurar e conhecer os valores atuais dos diversos parâmetros de configuração do xSCAN (Tabela 4.2);
2. Receber os A-Scans em tempo-real e dar acesso aos mesmos ao programador;
3. Processar os dados adquiridos;
4. Interromper o processo de excitação da sonda entre 2 aquisições de sinais;
5. Permitir usar os dois canais de transmissão/receção disponíveis, para por exemplo trocar a sonda a ser utilizada na aquisição de sinais.

A classe desenvolvida define 71 métodos dos quais 27 métodos são públicos e os restantes privados. Existem métodos que permitem configurar ou conhecer o valor atual de cada parâmetro do xSCAN. Também foram desenvolvidos métodos que permitem carregar uma certa configuração do xSCAN a partir de um ficheiro ou guardar a atual configuração do xSCAN num ficheiro para que possa ser carregada mais tarde. Adicionalmente foram desenvolvidos métodos de processamento de sinal como a realização de médias e filtragem de A-Scans.

Entre os vários métodos públicos desenvolvidos existem dois que se destacam:

- **`startGettingAScans_in_seconds`**;
- **`startGettingAScans_in_NumberOfASCANS`**;

Ambos os métodos têm o mesmo objetivo que é extrair os dados adquiridos do xSCAN. Um permite especificar o número total de A-Scans que devem de ser guardados numa aquisição, enquanto que o outro permite especificar o número de segundos de dados numa aquisição. Neste último caso, é calculado quantos A-Scans são adquiridos nesse intervalo de tempo. Apesar destes dois métodos serem especificados em unidades diferentes, o processo de aquisição para os dois é igual.

Para guardar os A-Scans, foi implementado um *buffer* circular. A alocação de memória para *buffer* é realizada de acordo com o número de A-Scans a guardar. Este número é definido por um dos dois métodos mencionados no paragrafo anterior. Cada elemento do buffer circular é um A-SCAN com *N* amostras (dependente do valor do parâmetro *Range* e da frequência de amostragem). Para facilitar a interpretação, o *buffer*, pode ser visto como uma matriz cujo número de colunas é igual ao número de A-Scans que o *buffer* tem de guardar e o número de linhas é igual ao número de amostras de um A-Scan.

O processo de aquisição consiste no lançamento de uma *thread* que irá executar quatro etapas de forma sequencial. A primeira etapa consiste na configuração do canal a usar para transmissão dos pulsos gerados (parâmetro *TX-Channel*) e ligação do gerador de pulsos de excitação. Isto previne que a sonda esteja a ser excitada por longos períodos desnecessariamente e o sobreaquecimento do equipamento. A segunda etapa consiste na criação de uma ligação TCP com o porto de aquisição do xSCAN. A terceira etapa é um procedimento cíclico dividido em duas fases. A primeira fase consiste em encontrar um pacote P4 e de seguida extrair o número de bytes indicado pelo conteúdo do pacote. A segunda fase é guardar os A-Scans extraídos em cada coluna no *buffer*. Por fim, a quarta etapa consiste em libertar os recursos alocados durante a execução da *thread*, quando termina o ciclo de aquisição. Desliga também o gerador de pulsos e o canal de transmissão e por último o fecho da ligação com o porto de aquisição do xSCAN. Na figura 4.5 é possível visualizar o fluxograma da *thread* de aquisição.

Esta API desenvolvida tem limitações:

- Não é possível realizar múltiplas aquisições em paralelo, ou seja, o valor do parâmetro *Step* é 1.
- Não é possível configurar os parâmetros enquanto decorre uma aquisição;
- O máximo *throughput* permitido é de 10 MB/s. Isto previne que a ligação TCP entre o computador e o porto de aquisição do xSCAN seja interrompida de forma precoce pelo xSCAN.

O ficheiro *.h* da API pode ser consultado no Anexo B. Este contém a definição da classe e dos métodos, privados e públicos, com os respetivos objetivos. Adicionalmente, contém a definição de estruturas e *macros*.

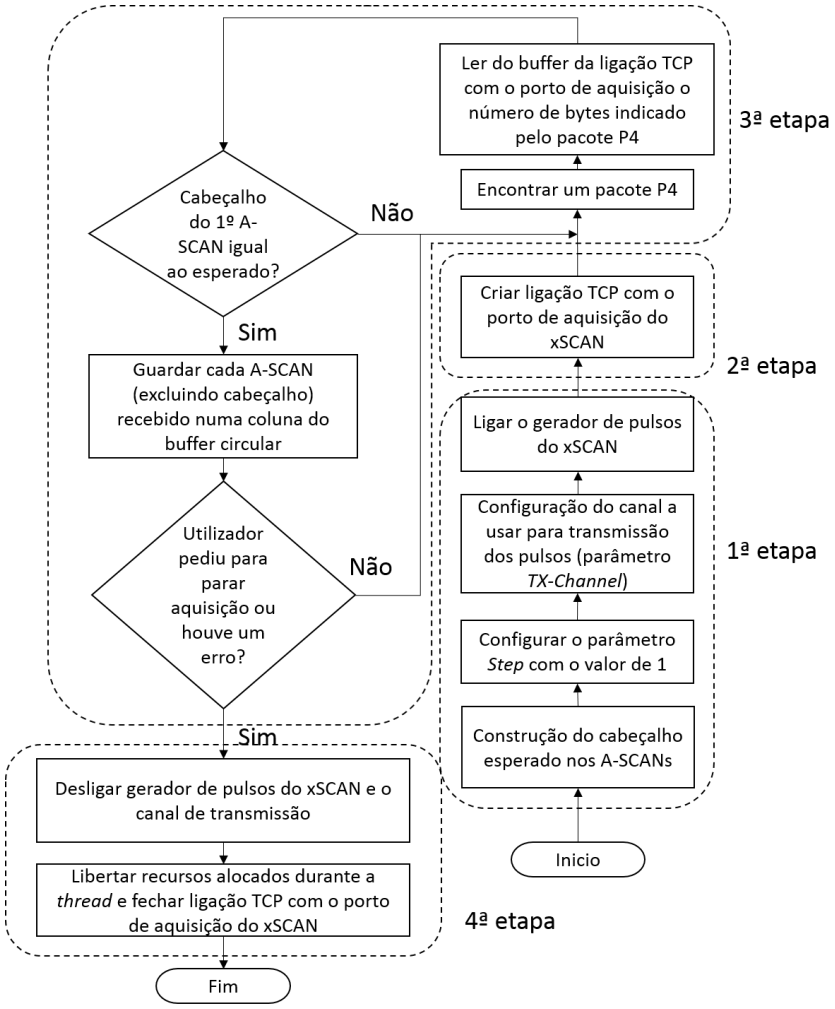

Figura 4.5 - Fluxograma da thread

4.5 Incorporação do xSCAN no sistema de aquisição de sinais

Com a API desenvolvida, foi possível desenvolver um programa *mex* para controlar o processo de aquisição em tempo real e ao mesmo tempo ter acesso aos dados no Matlab. *Mex* é um programa externo escrito em C, implementado como uma biblioteca de rotinas dinâmica (DLL) e pode ser evocado no Matlab como qualquer comando do ambiente Matlab. A escolha de fazer o programa em *mex* foi pela facilidade em este ser integrado no software de visualização que foi desenvolvido no ambiente do Matlab.

Este comando, denominado por **`xSCANAcquisitionV2`**, é constituído por 2 argumentos de entrada e 2 argumentos de saída. Cada argumento de entrada é uma estrutura cujos campos podem ser consultados na seguinte tabela 4.5. O comando pode ser evocado da seguinte maneira:

```
[X, config] = xSCANAcquisitionV2(DispConf, Filtro)
```

onde **X** é a matriz dos dados recolhidos; **config** é uma estrutura com os valores dos parâmetros de configuração do xSCAN; **DispConf** é uma estrutura que contém o gráfico onde se visualiza os sinais adquiridos, o ficheiro de configuração do xSCAN, etc; **Filtro** é uma estrutura que contém informação sobre o filtro, por exemplo, os coeficientes do numerador e denominador.

Tabela 4.5 - Campos das estruturas do 1º e 2º argumento de entrada

Nome do argumento de entrada	**Nome do campo**	**Descrição do campo**	**Campo obrigatório?**	**Tipo do campo**
DispConf	PlotSignal	Ponteiro para o *plot* do matlab onde vão ser desenhados os sinais	Sim	*Handle* do *plot*
	AxesHandle	Ponteiro para o *axes* do matlab para mudar a janela de visualização	Sim	*Handle* do *axes*
	CounterLabel	Ponteiro para uma *label* onde mostrará o número de vezes que foi feita uma captura automática	Não	*Handle* da *label*
	ConfigFile	Ficheiro de configuração do xSCAN	Sim	Vetor de caracteres (*char array*)
	WriteAcquisitionFile	O nome do ficheiro onde são guardadas as aquisições assim que o utilizador clicar no botão do rato	Sim	Vetor de caracteres (*char array*)
	t0ViewWindow	O limiar inferior do eixo temporal (eixo das abcissas) da janela de visualização em us	Não	*Double* de 64 bits
	t1ViewWindow	O limiar superior do eixo temporal (eixo das abcissas) da janela de visualização em us	Não	*Double* de 64 bits
	SecondsToAcquire	Número de segundos que se deve guardar no WriteAcquisitionFile	Não	Double de 64 bits
	AverageNumber	Média de A-Scans para display	Não	Inteiro de 32 bits
Filtro	NumeratorFilter	Vetor com os coeficientes do numerador do filtro	Sim	Vetor de doubles de 64 bits
	DenominatorFilter	Vetor com os coeficientes do denominador do filtro	Sim	Vetor de doubles de 64 bits

Os argumentos de saída são:

1. Matriz do tipo short (*int16*) que contém os últimos A-Scans guardados no *buffer* circular alocado à aquisição. A matriz possui um número de colunas igual ao número de A-Scans guardados no *buffer* e um número de linhas igual ao tamanho de um A-SCAN em amostras.

2. Uma estrutura que contém os valores dos parâmetros de configuração do xSCAN. Os campos que constituem esta estrutura podem ser consultados na seguinte tabela:

Tabela 4.6 - Campos da estrutura do 2º argumento de saída

Campo da estrutura	**Parâmetro do xSCAN**	**Tipo do campo**
PRF	PRF	Inteiro de 32 bits
Pre_Amp_Filter	*Pre Amplifier Filter*	Inteiro de 32 bits
Post_Amp_Filter	*Post Amplifier Filter*	Inteiro de 32 bits
Pulse_Type	*Pulse Type*	Inteiro de 32 bits
Pulse_Clamp	*Pulse Clamp*	Inteiro de 32 bits
Pulse_Width	*Pulse Width*	Inteiro de 32 bits
Pulse_Train_Iterations	*Pulse Train Iterations*	Inteiro de 32 bits
Range_us	*Range* (unidades us)	Float de 32 bits
Delay_us	*Delay* (unidades us)	Float de 32 bits
Gain_dB	*Gain* (unidades dB)	Float de 32 bits
Low_Noise_Amplifier	*Low Noise Amplifier*	Inteiro de 32 bits

O funcionamento deste comando é descrito nos seguintes passos:

1. Leitura dos argumentos de entrada, configuração do xSCAN e da janela de visualização, **`AxesHandle`**, dos A-Scans e iniciação da aquisição;

   1.1. Iniciar uma ligação ao porto configuração do xSCAN através do método da API denominado **`startConnectionConfigurato`**;

   1.2. Carregar a configuração do xSCAN. Isto é feito com auxílio do método da API denominado **`Load_Configuration_From_File`**;

   - O ficheiro de configuração contém a configuração do xSCAN no formato JSON (Tabela 4.7).

   1.3. Escrita do cabeçalho provisório no ficheiro `WriteAcquisitionFile` através da chamada do método da API denominado **`Write_Header_to_File`**;

- O processo de escrita do cabeçalho e dos A-Scans neste ficheiro é explicado no Anexo C.

1.4. Ajuste do eixo temporal (eixo das abcissas) da janela de visualização, **`AxesHandle`**, para que o utilizador possa visualizar apenas a zona temporal do A-Scan que considera importante;

- Caso o utilizador não mencione nos argumentos de entrada a zona temporal de interesse, a janela de visualização é ajustada de modo a que se veja um A-Scan completo;

1.5. Inicio da aquisição através da chamada ao método da API **`startGettingAScans_in_seconds`**.

2. Iniciação do processo de visualização em tempo real dos A-Scans extraídos do xSCAN. Este processo é cíclico e está dividido em 6 etapas;

2.1. Inicialização de um temporizador, **`refreshDataTimer`**, para atualização da janela de visualização dos dados adquiridos;

2.2. Realização de médias dos A-Scans através da chamada ao método da API **`makeNMeans`**;

- A quantidade de A-Scans utilizados para efetuar médias é definido pelo valor do campo **`AverageNumber`** (tabela 4.5). Caso este campo não esteja definido as médias não são realizadas.

2.3. Filtragem do A-Scan que resultou da execução da etapa anterior através do método **`FilterSignal`** seguido pela visualização do A-Scan filtrado no plot indicado pelo argumento de entrada **`PlotSignal`** (tabela 4.5);

- Caso não tenham sido feitas médias, o A-Scan utilizado nesta etapa para a filtragem é o mais recente.

2.4. Escrita dos A-Scans adquiridos no ficheiro através do método da API denominado **`Write_Ascans_To_File`**;

- Esta etapa só é executada caso o processo de escrita dos A-Scans num ficheiro tenha sido iniciado.

2.5. Caso o campo **`CounterLabel`** (tabela 4.5) esteja contido na estrutura do 1º argumento de entrada, é atualizado o *display* que indica o número de vezes que se escreveram A-Scans no ficheiro;

2.6. Averiguar se o temporizador **`refreshDataTimer`** excedeu os 100 ms. Caso não tenha passado os 100 ms, o programa principal ,excluindo a *thread* de aquisição, é adormecido o tempo que resta para fazer os 100 ms.

- Foi verificado durante testes que o xSCAN só envia os dados adquiridos a cada 100 ms (capítulo 4.3). Então foi decidido que não era necessário ter uma taxa de refrescamento da visualização do sinal e fazer este processamento mencionado anteriormente a uma frequência maior que 10 Hz.

3. Verificar se o utilizador clicou no botão central do rato.

3.1. Caso o utilizador tenha clicado pela primeira vez no botão do rato, é iniciado o processo de escrita de A-Scans no ficheiro e é iniciado um temporizador para saber quanto tempo de aquisição está a ser guardado no ficheiro, **`WriteAcquisitionTimer`**;

3.2. Caso tenha sido uma segunda vez, o processo de escrita e o processo de aquisição e visualização em tempo real dos A-Scans são interrompidos.

4. Verificação se o temporizador **`WriteAcquisitionTimer`** excedeu o número de segundos a adquirir, estipulado pelo argumento de entrada **`SecondsToAcquire`** (tabela 4.5) ou se o utilizador deseja interromper o processo de visualização e aquisição de dados. Caso uma destas situações se verifique, o processo de escrita e o processo de aquisição e visualização dos em tempo real A-Scans são interrompidos através da chamada do método **`stopAcquisitionThread`** da API e da interrupção do ciclo de visualização (passo nº 2) respetivamente;

5. Atualização do cabeçalho do ficheiro devido à alteração do valor de um campo.

Tabela 4.7 - Atributos e o tipo do atributo da configuração do xSCAN

Atributo	**Tipo do atributo**
PRF	Inteiro
PostAmpFilter	Inteiro
PreAmpFilter	Inteiro
TX Channel	Inteiro
RX Channel	Inteiro
Pulse Type	Inteiro
Pulse Width	Inteiro
Pulse Clamp	Inteiro
Pulse Train Iteration	Inteiro
Low Noise Amplifier	Booleano
Gain (db)	Float (2 casas decimais)
Range	Float (2 casas decimais)
Delay	Float (2 casas decimais)

O fluxograma do programa *mex* descrito anteriormente pode ser consultado na figura 4.6.

Com os sistemas de aquisição de sinais já apresentados e discutidos, é apresentado de seguida o software de visualização dos dados adquiridos para ambos sistemas de aquisição de sinais juntamente com o sistema de diretorias e base de dados desenvolvidos. Neste software de visualização são integrados os programas desenvolvidos nos capítulos anteriores.

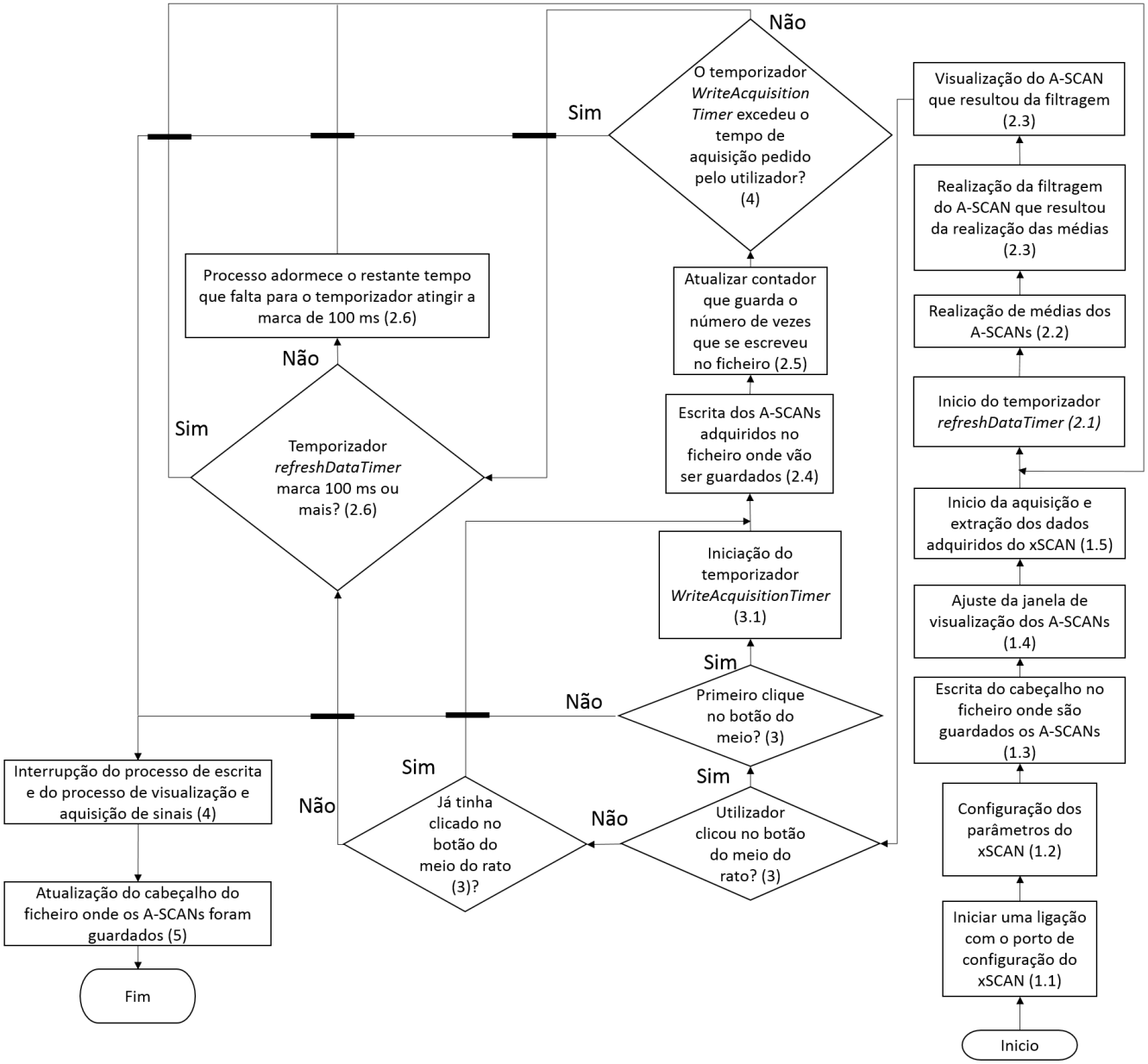

Figura 4.6 - Fluxograma do programa *mex* desenvolvido para extrair os dados adquiridos pelo xSCAN e visualizá-los

5. Software de Visualização de sinais em tempo-real

Neste capítulo é apresentado o software de visualização de sinais em tempo-real. Inicialmente são listados os requisitos funcionais e não funcionais do software. De seguida é apresentada a interface gráfica desenvolvida e os seus componentes. Por fim, é descrito o sistema de diretorias onde são guardados os ficheiros que contêm os A-Scans adquiridos e a base de dados desenvolvida para guardar informação relativa a cada aquisição, respetivamente.

5.1 Requisitos funcionais e não funcionais do software

O software de visualização foi desenvolvido para ser utilizado em ambiente clínico com objetivo principal de fazer a aquisição e visualização de sinais em tempo-real, bem como a correta salvaguarda dos dados numa base de dados para posterior processamento e estudo. Tendo por base esses pressupostos e objetivos foi feito um levantamento dos requisitos funcionais (ações que um software tem de ter) e não funcionais (característica que o software deve ter). Estes requisitos podem ser consultados na tabela 5.1.

Tabela 5.1 - Requisitos funcionais e não-funcionais do software

Número	**Requisito funcional**	**Requisito não funcional**
1	Visualizar sinais em tempo-real	Utilização simples
2	Realizar aquisições de sinais em tempo-real	Poucos campos de preenchimento e apenas os necessários para identificar o paciente
3	Guardar múltiplas aquisições associadas a um paciente	Boa fluidez entre ações
4	Guardar vários pacientes	Ter um sistema de registos e de diretorias bem estruturado
5	Guardar notas sobre uma dada aquisição	Funcionar para o sistema de aquisição com xSCAN
6	Configurar janela de visualização	Funcionar para o sistema de aquisição com a placa PCIe-9842
7	Guardar sinais adquiridos automaticamente	
8	Emitir som quando o número de sinais guardados for suficiente	
9	Desativação dos elementos da interface durante aquisição	

Os requisitos funcionais com número 1 e 2 são os requisitos com maior prioridade. A visualização em tempo-real é de extrema importância para que se saiba o que se está a adquirir e para verificar se a qualidade dos sinais adquiridos/guardados é satisfatória, ou seja, se é possível visualizar a interface anterior e posterior do cristalino e verificar a ocorrência de uma interface entre estas duas aquando de testes em pacientes com catarata. Para cumprir este requisito foi desenvolvido o comando externo Matlab descrito no capítulo 4.5 para o xSCAN e para a placa PCIe-9842 foi desenvolvido o programa descrito no capítulo 3.3.

Os requisitos funcionais 7 e 8 são essenciais para que o médico/utilizador do software consiga realizar a aquisição de sinais com sucesso e sem provocar desconforto no olho do paciente. Estes requisitos vão permitir ao médico focar-se mais na operação de manusear a sonda. Caso contrário, o médico teria de focar-se em três tarefas: manusear a sonda, visualizar se os sinais adquiridos são satisfatórios e mandar guardar os sinais adquiridos.

O requisito funcional 9 permite mostrar ao médico que uma aquisição de sinais está a decorrer. Caso o médico queira fazer alterações aos dados do paciente, este terá de interromper a aquisição. Assim o processo de inserir dados sobre o paciente e o processo de aquisição de sinais ficam separados, o que confere ao software a característica de simplicidade na utilização.

Os requisitos não funcionais 5 e 6 são compridos através da integração dos métodos descritos no capítulo 4.5 e capítulo 3.3 respetivamente no software de visualização.

5.2 Interface gráfica do utilizador

A interface gráfica do utilizador (GUI) foi desenvolvida em ambiente Matlab devido à facilidade de implementação nesse ambiente das componentes gráficas e também porque o Matlab é um software multiplataforma.

Na figura 5.1 pode-se visualizar a GUI desenvolvida. Já na tabela 5.2 pode-se ver uma lista dos componentes da GUI juntamente com uma descrição da sua finalidade.

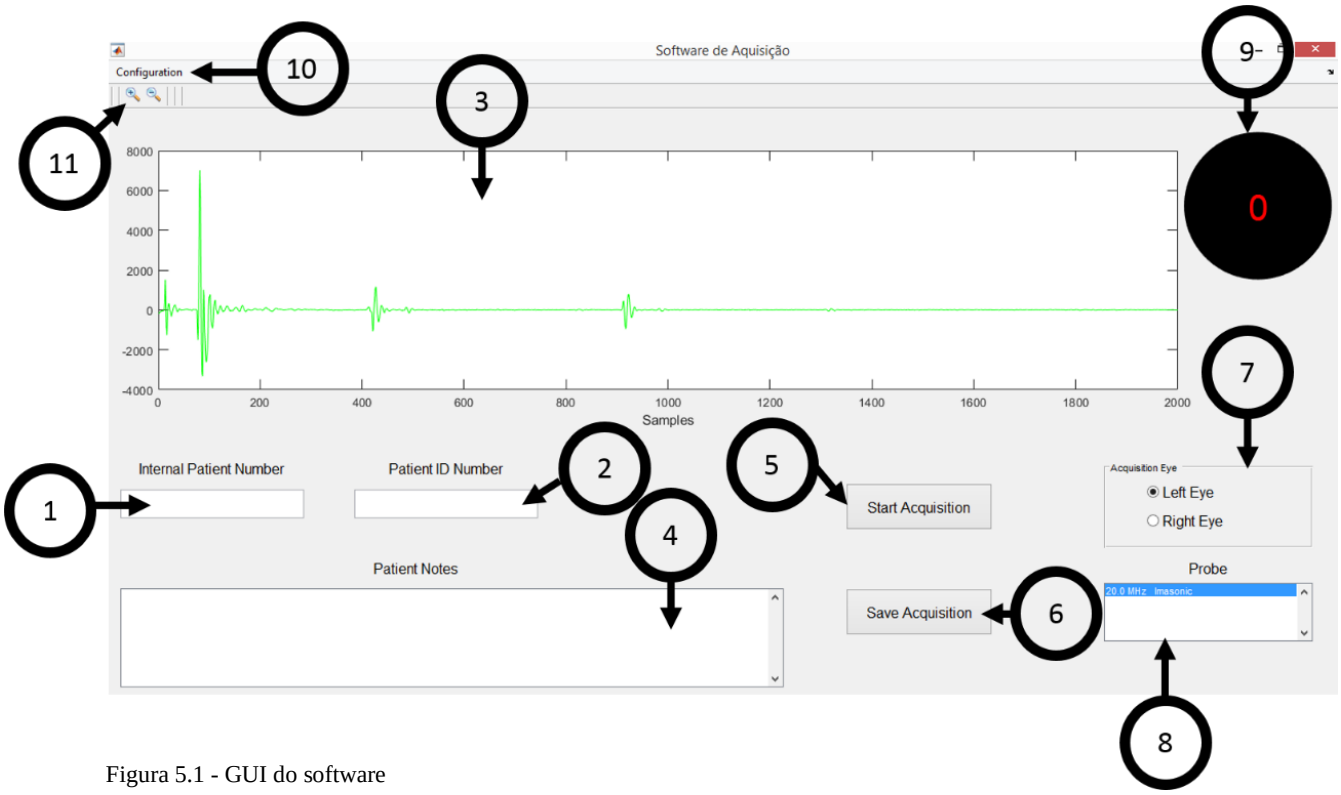

Figura 5.1 - GUI do software

Tabela 5.2 - Elementos que constituem o GUI e a sua funcionalidade

Número	**Descrição**	**Funcionalidade**
1	Caixa de texto	Número clínico do paciente
2	Caixa de texto	Número de identificação do paciente (ex: Bilhete de identidade)
3	*Plot*	Visualização dos A-Scans adquiridos em tempo-real
4	Caixa de texto	Notas sobre a aquisição realizada
5	Botão	Iniciar aquisição
6	Botão	Guardar a aquisição definitivamente
7	*Radio Button Box*	Indicação do olho em que foi feita a aquisição
8	Lista	Escolha da sonda para a realização da aquisição
9	Label	Mostrador da quantidade de vezes que foram escritos A-Scans no ficheiro onde estes são guardados
10	Botão de menu	Permite configurar o tamanho da janela de visualização dos A-Scans
11	Zoom in e Zoom Out	Permite fazer zoom in ou zoom out na visualização dos A-Scans

5.3 Base dados e sistema de diretorias

Devido à necessidade de guardar informação sobre cada paciente e de lhe ser associado ao longo do tempo as diversas aquisições realizadas (por exemplo, para verificar a evolução da catarata), foi desenvolvida uma base dados relacional em SQL. Complementarmente, foi desenvolvido um sistema de diretorias para que cada ficheiro de aquisição fique associado a uma data de aquisição e a um dado paciente, sendo ainda

previsto o suporte da informação proveniente de outros exames clínicos complementares (e.g. LOCS3), para além dos resultados obtidos com o sistema ESUS. Desta forma, os ficheiros resultantes e guardados de cada aquisição ficam convenientemente organizados no disco do computador.

5.3.1 Sistema de diretorias do sistema ESUS

O sistema de diretorias funciona da seguinte maneira:

- Para cada paciente que realizou no mínimo uma aquisição e que foi guardada existe uma diretoria denominada diretório_paciente (círculos azuis com o número 1 na figura 5.2);
- Para cada aquisição de um dado paciente existe um diretório denominado diretório_aquisição (círculos verdes com o número 2 na figura 5.2);
- Dentro de cada diretório_aquisição, existem outros 2 diretórios:
    - Diretório, denominado diretório_sinais, onde é guardado o ficheiro de aquisição, que contém os A-Scans adquiridos com o sistema ESUS (círculos vermelhos com o número 3 na figura 5.2);
    - Diretório, denominado diretório_imagem, onde são guardadas as imagens de lâmpada de fendas associadas à aquisição realizadas (círculos castanhos com o número 4 na figura 5.2);

Na figura 5.2 é possível observar o sistema de diretorias em formato de árvore.

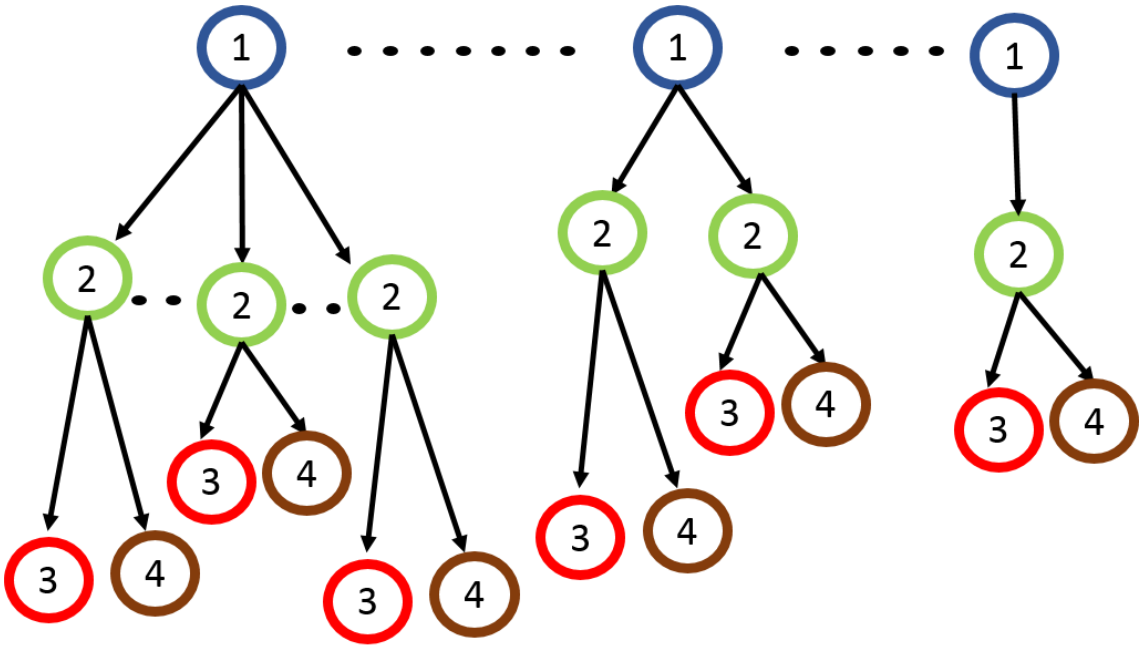

Figura 5.2 - Sistema de diretorias em formato arvore

5.3.2 Base de dados do sistema ESUS

A base dados foi desenvolvida com objetivo de cumprir os seguintes requisitos:

- Guardar informação sobre cada paciente que realizou aquisições:
  - Número clínico;
  - Número de identificação (e.g.: Cartão de Cidadão);
  - Diretório_paciente;
- Guardar informação sobre a aquisição realizada:
  - A data da aquisição;
  - Nome do ficheiro de aquisição que contém os A-Scans adquiridos;
  - Notas escritas pelo utilizador relativas à aquisição;
  - Classificação LOCS III;
  - O olho em que foi realizada a aquisição;
  - Diretório_aquisição;
- Associar uma/múltiplas aquisição/aquisições a um dado paciente;

- Guardar informação sobre as possíveis sondas a serem utilizadas na aquisição:
    - Frequência central;
    - Fabricante/nome;
    - Ficheiro que contém os valores dos parâmetros de configuração para a utilização da sonda)
- Associar o tipo de sonda utilizada à aquisição realizada;
- Guardar informação sobre cada imagem de lâmpada de fenda:
    - Nome do ficheiro que contém a imagem de lâmpada de fenda;
- Associar a uma dada aquisição, imagens de lâmpada de fenda;

Com base nestes requisitos foi originado o diagrama relacional[11] da figura 5.3.

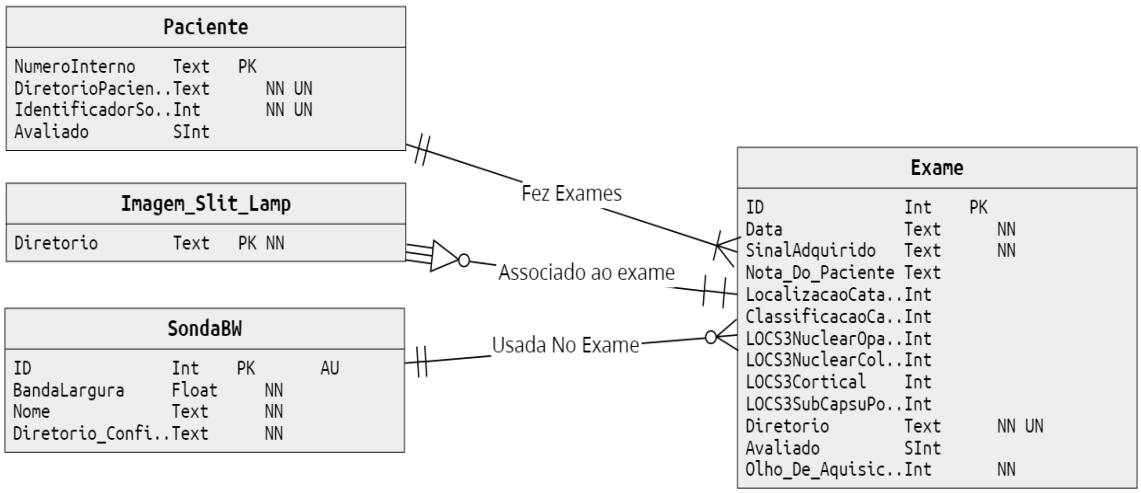

Figura 5.3 - Diagrama relacional da base dados (PK – Chave Primária; NN – Não Nulo; UN – Valor Único; AU – Incrementado Automaticamente)

Com o diagrama relacional produzido foi possível implementar a constituição da base de dados com auxílio do software *DB Browser for SQLite*[12]. De modo a que o software de visualização insira novos registos na base de dados foi necessário instalar a *Database Toolbox*[13] API do Matlab.

[11] Diagrama relacional desenvolvido com auxílio do software online disponível em http://onda.dei.uc.pt/v3/ e de acordo com a notação de James Martin (Hitchman, 2002)
[12] https://sqlitebrowser.org
[13] https://www.mathworks.com/products/database.html

6. Conclusão e trabalho futuro

Este trabalho conduziu ao desenvolvimento de dois sistemas de aquisição e visualização de sinais em tempo real, a serem usados em ambiente clínico. Dos dois sistemas desenvolvidos, um utiliza o dispositivo *Panametrics* e a placa PCIe-9842 e o outro utiliza o sistema xSCAN.

No sistema de aquisição de sinais com a placa PCIe-9842 foi desenvolvido um programa que permite em tempo real extrair os dados adquiridos pela placa PCIe-9842 e mostrá-los ao utilizador. Este programa foi desenvolvido em C++ com o auxílio da API DAQPilot, disponível para arquiteturas x86 de 32 bits. A placa PCIe-9842 usa um *bus* PCIe que tem de existir no computador hospedeiro. Devido a estes fatores, o sistema de aquisição com a placa fica constrangida às plataformas em que pode ser integrado.

No sistema de aquisição de sinais com o xSCAN foi desenvolvido uma API com o objetivo de suportar aquisição e visualização dos sinais adquiridos em tempo-real. Conjuntamente com a API foi desenvolvido um programa para integrar o xSCAN e permitir o acesso e visualização dos dados adquiridos no sistema de aquisição e visualização de sinais.

Para a visualização dos sinais acústicos foi desenvolvido um software de visualização de sinais em tempo real em ambiente Matlab que compreende uma interface gráfica, um sistema de diretorias e uma base de dados. Este software tem a capacidade de ser integrado em qualquer dos sistemas de aquisição de sinais.

Adicionalmente, foi ainda desenvolvido um algoritmo de deteção automática de reflexões das interfaces do olho, onde se valida o A-Scan em que estas têm amplitudes adequadas. Devido à falta de diversidade de sinais, com e sem catarata, para testar este algoritmo, este ainda não foi integrado no sistema de aquisição e visualização de sinais.

6.1 Trabalho Futuro

Futuramente, irá ser realizado um estudo clínico para a extração de sinais em humanos com cataratas. Este estudo está atualmente pendente da aprovação da Comissão de Ética para a Investigação Clínica (CEIC) e da Autoridade Nacional do Medicamento e Produtos de Saúde (INFARMED).

Com a aprovação do estudo clínico e com os sinais adquiridos, será testado e trabalhado o algoritmo de deteção automática de reflexões das interfaces do olho, visto que já teremos nesta etapa uma variedade de sinais com e sem catarata.

Posteriormente será feito o condicionamento e processamento dos sinais acústicos adquiridos no domínio do tempo e da frequência. Esta etapa incluirá o desenvolvimento de algoritmos para deteção das interfaces do olho humano como a córnea e o cristalino, bem como a deteção da catarata. Estes algoritmos terão como base os algoritmos desenvolvidos no projeto FCT CATARATA para a deteção das interfaces dos olhos em ratos Wistar.

Com os algoritmos de deteção das várias interfaces do olho e da catarata desenvolvidos, serão desenvolvidos algoritmos para extração de *features* que mais tarde serão usadas em métodos de *machine learning* para estimação da dureza da catarata tendo por base os métodos que obtiveram resultados positivos no projeto FCT CATARATA.

Por fim, iremos integrar o método de *machine learning* com melhor performance no sistema de aquisição de sinais.

Espera-se que este trabalho contribua para um avanço significativo nesta área de atuação médica.

Referências

Abdallah, M., & Elkeelany, O. (2009). A survey on data acquisition systems DAQ. *ICC2009 - International Conference of Computing in Engineering, Science and Information*, 240–243. https://doi.org/10.1109/ICC.2009.24

AD-LINK. (2014). PCIe/PXIe-9852 Datasheet, (September).

AD-LINK. (2016). PCIe-9814/9834 Datasheet, (August).

AlazarTech. (2019a). Ats9350 Datasheet, (May).

AlazarTech. (2019b). Ats9870 Datasheet, (May).

Beebe, D. C., Holekamp, N. M., & Shui, Y. B. (2010). Oxidative damage and the prevention of age-related cataracts. *Ophthalmic Research*, *44*(3), 155–165. https://doi.org/10.1159/000316481

Boyd, K. (2018). What are cataracts? Retrieved from https://www.aao.org/eye-health/diseases/what-are-cataracts

Caixinha, M. (2016). *In Vivo Automatic Cataract Detection and Classification using Ultrasound Technique*.

Caixinha, M., Jesus, D. A., Velte, E., Santos, M., & Santos, J. (2014). Using ultrasound backscattering signals and Nakagami statistical distribution to assess regional cataract hardness. *IEEE Transactions on Biomedical Engineering*, *61*(12), 2921–2929. https://doi.org/10.1109/TBME.2014.2335739

Chaudhari, H., Thakkar, G., & Gandhi, V. (2013). Role of ultrasonography in evaluation of orbital lesions.

Chylack, L. T., Leske, M. C., Sperduto, R., Khu, P., & McCarthy, D. (1988). Lens Opacities Classification System. *Archives of Ophthalmology*, *106*(3), 330–334. https://doi.org/10.1001/archopht.1988.01060130356020

Chylack, Leo T, Leske, M. C., Mccarthy, D., Khu, P., Kashiwagi, T., & Sperduto, R. (1989). Lens Opacities Classification System II (LOCS II), (Locs Ii).

Chylack, Leo T, Wolfe, J. K., Singer, D. M., Leske, M. C., Bullimore, M. a, & Ian, L. (1993). The Lens Opacities Classification System III, 6–11.

https://doi.org/10.1001/archopht.1993.01090060119035

El-Brawany, M. (2009). Ultrasound-based noninvasive measurement of cataract hardness. *The Online Journal on Electronics and Electrical Engineering (OJEEE)*, (2), 246–249. Retrieved from http://infomesr.org/attachments/W09-0044.pdf

Eric J. Linebarger, David R. Hardten, Gaurav K. Shah, & Richard L. Lindstrom. (1999). Phacoemulsification and Modern Cataract Surgery. *Survey Of Ophthalmology*, *44*(2), 123–147.

Fan, S., Dyer, C. R., Hubbard, L., & Klein, B. (2003). An Automatic System for Classification of Nuclear Sclerosis from Slit-Lamp Photographs. *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003*, (1), 592–601. https://doi.org/10.1007/978-3-540-39899-8_73

Fine, I. H., Packer, M., & Hoffman, R. S. (2002). New phacoemulsification technologies. *Journal of Cataract and Refractive Surgery*, *28*(6), 1054–1060. https://doi.org/10.1016/S0886-3350(02)01399-8

Gross, H., Blechinger, F., & Achtner, B. (2008). *Handbook of optical systems, Volume 4: Survey of optical instruments*. *Handbook of optical systems*.

Guo, L., Yang, J. J., Peng, L., Li, J., & Liang, Q. (2015). A computer-aided healthcare system for cataract classification and grading based on fundus image analysis. *Computers in Industry*, *69*, 72–80. https://doi.org/10.1016/j.compind.2014.09.005

Harini, V., & Bhanumathi, V. (2016). Automatic cataract classification system. *International Conference on Communication and Signal Processing, ICCSP 2016*, 815–819. https://doi.org/10.1109/ICCSP.2016.7754258

Hitchman, S. (2002). The Details of Conceptual Modelling Notations are Important - A Comparison of Relationship Normative Language. *Communications of the Association for Information Systems*, *9*(September). https://doi.org/10.17705/1cais.00910

Huang, C. C., Zhou, Q., Ameri, H., Wu, D. W., Sun, L., Wang, S. H., … Shung, K. K. (2007). Determining the Acoustic Properties of the Lens Using A High-Frequency Ultrasonic Needle Transducer. *Ultrasound in Medicine and Biology*, *33*(12), 1971–1977. https://doi.org/10.1016/j.ultrasmedbio.2007.06.004

Imasonic. (2014). Measurement report 20 MHz A-Scan probe.pdf.

Jesus, D., Caixinha, M., Santos, M., & Santos, J. (2012). Ultrasound techniques for lens hardness characterization: A comparison study. *IEEE International Ultrasonics Symposium, IUS*, *1*, 2376–2379. https://doi.org/10.1109/ULTSYM.2012.0594

Kassoff, A., Kassoff, J., Mehu, M., Buehler, J. A., Eglow, M., Kaufman, F., … Strahlman, E. (2001). The Age-Related Eye Disease Study (AREDS) system for classifying cataracts from photographs: AREDS Report No. 4. *American Journal of Ophthalmology*. https://doi.org/10.1016/S0002-9394(00)00732-7

Keithley. (2001). *Data Acquisition and Control Handbook*.

Kolhe, S., & K. Guru, S. (2015). Cataract Classification and Grading: A survey, *Vol3*.

Laugier, P., & Haïat, G. (2011). Introduction to the physics of ultrasound. In *Bone Quantitative Ultrasound*. https://doi.org/10.1007/978-94-007-0017-8_2

Li, H., Lim, J. H., Liu, J., Wong, D. W. K., Foo, Y., Sun, Y., & Wong, T. Y. (2010). Automatic detection of posterior subcapsular cataract opacity for cataract screening. *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC'10*, 5359–5362. https://doi.org/10.1109/IEMBS.2010.5626467

Li, H., Lim, J. H., Liu, J., Wong, T. Y., Tan, A., Wang, J. J., & Mitchell, P. (2008). Image based grading of nuclear cataract by SVM regression, *6915*, 691536. https://doi.org/10.1117/12.769975

National Eye Institute. (2015a). Cataract What You Should Know. Retrieved from www.nei.nih.gov

National Eye Institute. (2015b). Facts About Cataract. Retrieved from https://www.nei.nih.gov/health/cataract/cataract_facts

NICE. (2018). Catar Cataracts acts in adults : management, (October 2017).

Olson, R. J., Mamalis, N., Werner, L., & Apple, D. (2003). Cataract treatment in the beginning of the 21st century. *American Journal of Ophthalmology*, *136*(1), 146–154. https://doi.org/10.1016/S0002-9394(03)00226-5

Paunksnis, A., Kurapkiene, S., Maciulis, A., Kopustinskas, A., & Paunksniene, M.

(2007). Ultrasound quantitative evaluation of human eye cataract. *Informatica*, *18*(2), 267–278.

"Royal National Institute of Blind People." (2019). Understanding Cataracts. *Royal National Institute of Blind People*.

Sasaki, K., & Sakamoto, Y. (1997). A new grading system for nuclear cataracts, *27*, 42–49.

Sasaki, K., Shibata, T., & Obazawa, T. (1990). Classification System for Cataracts, *91*, 399–404.

Shlensky, D., & Ringeisen, A. (2015). Ophthalmologic Ultrasound. Retrieved from http://eyewiki.aao.org/Ophthalmologic_Ultrasound

Sparrow, J. M., Bron, A. J., Brown, N. A. P., Ayliffe, W., & Hill, A. R. (1986). The Oxford Clinical Cataract Classification and Grading System. *International Ophthalmology*, *9*(4), 207–225. https://doi.org/10.1111/1756-185X.13281

Tabandeh, H., Wilkins, M., Ramnarine, K., Nassiri, D., Karim, A., & Thompson, G. M. (2000). Hardness and ultrasonic characteristics of human crystalline lens. *Investigative Ophthalmology and Visual Science*, *37*(3), 0–3.

The Royal College of Ophthalmologists (Scientific Department). (2010). Practice Guidance: Cataract Surgery Guidelines. Retrieved from https://www.rcophth.ac.uk/wp-content/uploads/2014/12/2010-SCI-069-Cataract-Surgery-Guidelines-2010-SEPTEMBER-2010-1.pdf

West, S. K., Rosenthal, F., Newland, H. S., & Taylor, H. R. (1988). Use of photographic techniques to grade nuclear cataracts. *Investigative Ophthalmology and Visual Science*, *29*(1), 73–77.

WHO. (1997). Strategies for Prevention of Blindness in National Programmes: A primary health care approach. https://doi.org/WHO/PBL/83.8

WHO. (2004). Global Cataract Surgical Rates 2004. Retrieved September 1, 2019, from https://www.who.int/blindness/data_maps/CSR_WORLD_2004.jpg?ua=1

WHO. (2010). Global data on visual impairments 2010, 17. Retrieved from http://www.who.int/blindness/GLOBALDATAFINALforweb.pdf

WHO. (2012). Cataract Grading System. Retrieved from http://www.who.int/ncd/vision2020_actionplan/documents/WHO_PBL_01.81.pdf%5Cnpapers2://publication/uuid/DE65271D-6622-4D3A-8558-242C74C1D6D1

Xu, Y., Gao, X., Lin, S., Wong, D. W. K., Liu, J., Xu, D., … Wong, T. Y. (2013). Automatic grading of nuclear cataracts from slit-lamp lens images using group sparsity regression. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8150 LNCS*(PART 2), 468–475. https://doi.org/10.1007/978-3-642-40763-5_58

Anexo A

Development of an Eye Scan Ultrasound System for Human Cataract Detection

Carlos Pinto[1,2], Paulo Fernandes[1,3], Miguel Caixinha[4,5], Sandrina Nunes[6], Miguel Morgado[3], Mário Santos[2,5], Jaime Santos[2,5], Lorena Petrella[2], Marco Gomes[1,2], Fernando Perdigão[1,2]

(1) *Instituto de Telecomunicações*, Coimbra, Portugal
(2) *Department of Electrical and Computer Engineering, University of Coimbra, Coimbra, Portugal*
(3) *Department of Physics, University of Coimbra, Coimbra, Portugal*
(4) *Department of Physics, University of Beira Interior, Covilhã, Portugal*
(5) *CEMMPRE, University of Coimbra, Coimbra, Portugal*
(6) *Association for Innovation and Biomedical Research on Light and Image, 3000-548, Coimbra*
{carlos.pinto, paulo.fernandes, marco, fp, lorena.petrella}@co.it.pt; {miguel.caixinha, migmorgado, nunes.sandrina}@gmail.com; {jaime, marioj}@deec.uc.pt

***Abstract*— Cataract is an ocular condition that affects more than 20 million people worldwide and is the leading cause of vision loss. Currently, the phacoemulsification is the most used procedure to extract cataract and recover visual acuity. For the surgery, the most important goal is the correct evaluation of cataract hardness. This paper addresses the development of a medical prototype to detect a cataract and classify its hardness, in the scope of the FCT project named CATARACTUS. In this work, an ultrasound signal acquisition system is presented to be used in clinical environment. The paper also focuses on the communication process between the system and a host computer. Finally, an overview of the tasks defined within the CATARACTUS project are presented that will follow the current stage of prototype development.**

Keywords— Cataract, Ultrasound, TCP, ESUS, CATARACTUS.

I. Introduction

Cataract is an ocular condition that affects the world population, and in the present, it is the number one cause of blindness worldwide. The cataract is a clouding or opacity of the normally transparent lens of the eye (Fig. 1) which is mainly constituted by proteins and fibers. In a healthy lens, these proteins and fibers are organized in such a way that they don't interact with the visible light spectrum.

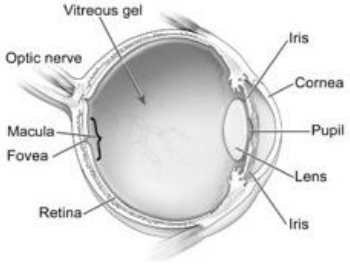

Fig .1. Ocular structures from a human eye (National Eye Institute, National Institutes of Health).

At the present, there is no pharmacological treatment available, which means that the only option for removing cataract is by a surgical technique. The standard phacoemulsification or femtosecond laser-assisted cataract surgery are the most used methods for cataract extraction [1].

The phacoemulsification technique uses an ultrasonic device to fragment the cataractous lens in small pieces, which are afterwards aspirated. An inappropriate phacoemulsification energy level selection, by overestimation of the cataract hardness, may result in the disruption of the posterior lens capsule and corneal endothelial cells loss. Since the ultrasound energy level to use in the surgery is directly related to the cataract hardness, its correct estimation is of great importance in the surgical planning context in order to minimize surgical complications [2].

Nowadays, the cataract classification is made through visual methods such as slit-lamp and retro-illumination images, with the aid of classification systems such as LOCS III, the Oxford Clinical Cataract Classification, and Grading Systems [3][4][5].

In a previous work [6][7], a system named Eye Scan Ultrasound System (ESUS) has been developed for cataract detection and classification, based on studies carried out in Wistar rats. Now, the authors are taking a step forward, trying to evolve the preclinical ESUS prototype to a medical device, for detection and classification of cataracts in humans by ultrasounds, in the scope of a new project funded by Fundação para a Ciência e a Tecnologia (FCT)[8]. An important advantage of ultrasound waves over the slit-lamp method is that reflections occur regardless the lens opacity and hardness. Cataracts' opacity and hardness are not, in fact, linearly correlated, although a higher level of opacity may indicate a

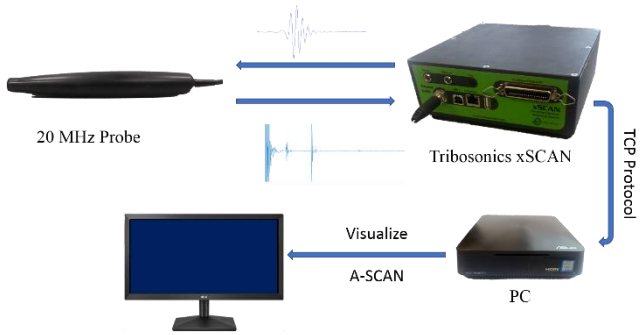

Fig. 2. Components of the ESUS

harder cataract [6]. It means that different lens with the same opacity but different hardness, will reflect the ultrasound waves in a different way. Being so, the methods for cataract detection based on ultrasounds could detect incipient cataracts, i.e., cataracts with a very low opacity level, which are not detectable by visual methods.

At the present, the ESUS (Fig. 2) is being evaluated to comply with EU regulation 2017/745 [9] which is mandatory to proceed with clinical trials. This system is composed by several components, namely a 20 MHz ultrasound probe (custom made by Imasonic SAS, Bourgogne-Franche-Conté, France), the probe excitation and signal acquisition by using the xSCAN equipment (custom made by Tribosonics Ltd., Sheffield, UK), which consists of a pulser-receiver module and an acquisition board operating at 100 Mega Samples per Second (MSPS) and 14 bits per sample. The excitation pulse repetition frequency (PRF) is 1 kHz. Finally, a host computer (PC) receives the acquired signals and processes them.

In this paper, the problem of data transfer between the acquisition system, Tribosonics xSCAN, and a host computer is addressed. An Application Program Interface (API) was specifically developed to be able to communicate directly with the xSCAN through a PC, to configure the xSCAN and to collect the echo signals (A-Scans). Finally, it is discussed the next steps to implement cataract detection in human and classification of its hardness.

II. Acquisition and Transmitter-Receiver Device

Tribosonics xSCAN5 has 3 main features: send a excitation pulse to the ultrasonic probe, receives and amplify the echoes and makes the analogic to digital conversion (ADC), of the received signal.

This equipment has an associated software that enables their control; however, it lacks an API to communicate directly with xSCAN. This makes impossible to integrate the Tribosonics xSCAN with our ultrasound signal visualization software, adapted to be used by healthcare professionals to perform the eye's ultrasound acquisition in patients. To overcome this problem, we developed an API to integrate the xSCAN with our software.

TABLE I. xSCAN CONFIGURABLE PARAMETERS

N	Parameter	Possible values
1	PRF	1Hz -100 kHz
2	Pre-Amplifier Filter	Several Bandwidths: [0.25 MHz - 1 MHz] to [10 MHz - 20 MHz]; none
3	Post-Amplifier Filter	Same value as Pre-Amplifier Filter
4	Transmission Channel	1 or 5
5	Receiver Channel	1 or 5
6	Pulse Type	8 different types, from "Positive square wave" to "Bipolar Pulse Train"
7	Pulse Width	steps of 10 ns
8	Pulse Clamp	steps of 10 ns
9	Pulse Train Iterations	[1 - 100]
10	Low Noise Amplifier	on/off
11	Gain	0 - 80dB
12	Delay	0 - 100µs
13	Range	1µs – 5000µs

This equipment has several configurable parameters listed in Table I, such as: the pulse repetition frequency (PRF); Range (number of samples in each A-Scan); gain; pulse type and width; transmitter and receiver channels and the channel mode, selectable between Pulse-Echo and Pitch-Catch.

When the xSCAN is powered on, it will start with a default value for each parameter. The acquisition starts when a transmission control protocol (TCP) connection is created between the host computer and the xSCAN. Additionally, the xSCAN can make up to sixteen different acquisitions simultaneously, each one called a step. Each parameter from the Transmission Channel (4rd parameter in Table I) to the Range (13th parameter in Table I) can have a different value across all the steps. Currently the API considers only one step.

The xSCAN uses two TCP communication ports. The port 2000, which in the remain of the article will be called the configuration port, is used to exchange (SET and GET) commands between the PC and the xSCAN; e.g., to set a new value for the PRF parameter or to get the value of the pulse width parameter, respectively. To each SET/GET command

[5] Website of Tribosonics http://www.tribosonics.com/

the xSCAN always returns the parameter value or an error message in case of fail. The second port is the port 3000 which will be called the acquisition port. This port is used by xSCAN to send the acquired A-Scans. In fact, the creation of this connection is used to start the acquisition of the A-Scans.

The process of communication between the PC and the configuration port is based on a client-server architecture. This means that the xSCAN (the server) only sends a message to the PC (as client), when the PC asks for a service (SET/GET parameter value). Regarding the acquisition port, the communication is done in one direction from the xSCAN to the PC, apart from the TCP acknowledges packets that are inherent to the TCP protocol.

Fig. 3 and Fig. 4 show diagrams exemplifying the communication session between the PC and the configuration port or the acquisition port, respectively.

In Fig. 3, at t0 the host PC application opens a TCP connection with the configuration port. At t2, the host PC sends a GET command to the xSCAN to determine the current value of a given parameter or group of parameters. At t4, the xSCAN sends the response to the GET command. This response could be the value(s) of the parameter(s) asked for, or an error message indicating that the content of the command GET sent by the PC was incorrect. At t6, the PC application sends a SET command to configure a given parameter with a value. The response to the SET command comes at t8 and is originated in the xSCAN. The content of

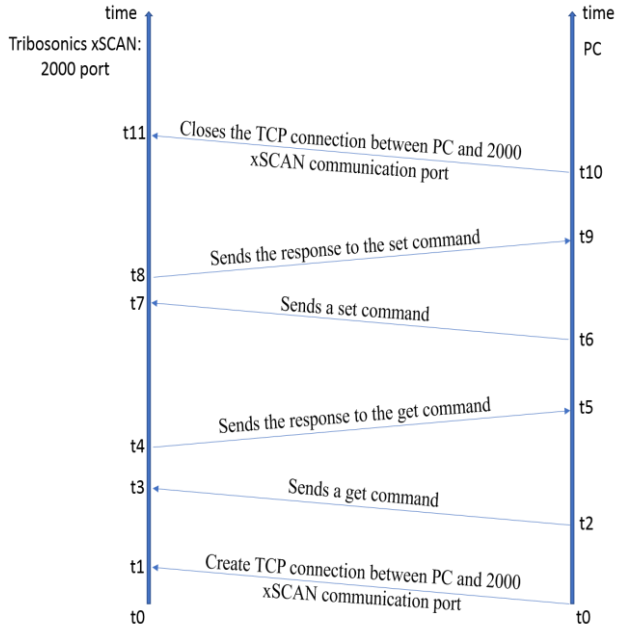

Fig. 4. Diagram exemplifying a communication session between the PC and the xSCAN 3000 port.

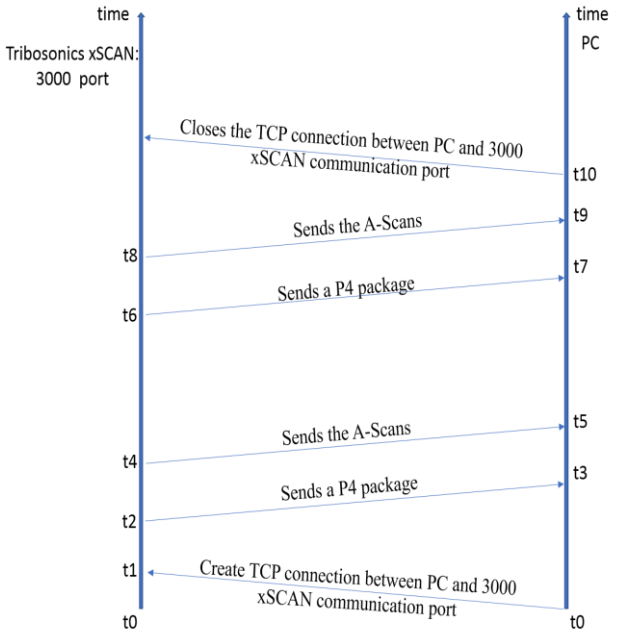

Fig. 3. Diagram exemplifying a communication session between the PC and the xSCAN 2000 port.

the response could be an error message or the current value of the parameter that was configured by the SET command at t6. At t10, the PC application closes the TCP connection with the configuration port.

In Fig. 4, at t0 the host PC application opens a TCP connection with the acquisition port. At t1, the acquisition will start because a connection was established between the PC application and the acquisition port, as previously mentioned. At t2, the xSCAN sends a special packet, with a payload size of 4 bytes indicating the number of bytes of useful information (A-Scans) that will be sent following this TCP packet. This special packet will be referred as a P4 packet, as shown in Fig. 4. If we divide this number of bytes by the size of the A-Scan, plus a header (typically 12 bytes), it will give the number of A-Scans that xSCAN will send following the P4 packet. This corresponds to Eq. 1. The number of bytes in P4 packet is dependent of the PRF and Range parameter (desired depth of A-Scan in the human eye). Range is measured in µs, assuming a constant propagation velocity in the media. The A-Scan size is the number of samples of a single A-Scan which is given by the product of the Range parameter by the sampling frequency (Fs), fixed on 100 MHz.

$$N\_A\text{-}SCANs = \frac{Number\ of\ bytes\ indicated\ in\ P4\ packet}{Range(\mu s) * Fs(MHz) * 2 + header\_size}$$

In Fig. 4, at t4 the xSCAN sends the acquired A-Scans to the PC. Between t6 and t8 included, the same events mentioned between t2 and t4 will occur. The number of A-Scans sent can be different if the PRF value changes between t4 and t6, or the A-Scan size can be different if the value of the Range parameter changes on this interval. At t10, the PC

application closes the TCP connection with the acquisition port.

Some restrictions/disadvantages were found during the use of xSCAN. One restriction is the absence of a reset command, which means that the only way to reset the xSCAN equipment is to unplug and plug it back whenever the xSCAN faces a malfunction, usually due to a too high baud exceeding the available ethernet bandwidth. The way found to circumvent that problem was to limit the throughput of the connection. The connection in both ports can also be properly closed with an API call (t10 in Fig. 3 and Fig. 4). Other restriction is related with the steps, referred in section II, where all 16 steps must be indicated in a GET/SET command, even if not used.

III. Application Programming Interface (API)

As previously mentioned the xSCAN is only controlled by a manufacturer software. In order to integrate the xSCAN with our ESUS ultrasound signal visualization software, designed to help the ophthalmology professionals, an API was developed that enables the xSCAN control. The API was built with four purposes:

i) Configure and obtain information of the current values of the xSCAN parameters for one step only;
ii) Pause the excitation process of the ultrasound probe;
iii) Enable to switch between two probes;
iv) Receive and provide to the user's application the A-Scans sent by xSCAN.

The API was developed in C++ language with the Microsoft Visual Studio Tool. It was defined as a class, with several (16) public methods. These methods allow the user to create a connection between the PC and the configuration port, and to configure or obtain information of the current parameter value. Additionally, there is a method that allows to save and load the xSCAN configuration to and from a file. Among the most important public implemented methods, two of them allow the user to set the rate of A-Scans to be acquired. The user can specify the circular buffer length in terms of A-Scans number or in seconds. In addition, methods were also defined to get the message that is linked to a certain error code, stop the acquisition of A-Scans and terminate all open connections to the xSCAN, releasing the resources that were being used by those connections.

The creation of a link between the PC and the configuration port goes through four main steps. The first one is the execution of a ping command to find out if the xSCAN is at least connected to the same Local Area Network (LAN). The second step is to check if any application running on the PC has already a connection with the configuration port. The third is to open a temporary connection to the acquisition port and wait for any message sent through this link. This step is fundamental because allows the API to know that xSCAN is ready to accept commands. The final step is the creation of the link between the PC and the configuration port.

For the A-Scans acquisition, the user has two methods as mentioned before, that allow to specify in two different ways the acquisition duration/length of the circular buffer. Either methods launch a thread that starts the A-Scans acquisition. The thread encompasses 4 main stages. The first is the configuration of the transmission channel and turn on the supply voltage to the ultrasound probe. Until this point, the transmission channel was not truly configured. The reason for this is to protect the probe from long excitation periods. The second stage is the creation of the TCP connection between the PC and the acquisition port. The third stage is a cyclic procedure involving 2 steps: find a P4 packet to extract the number of bytes of coming A-Scans and save the A-Scans in the user circular buffer. The fourth stage consists of releasing the resources allocated in the thread and close the connection with the acquisition port.

IV. Future Work

A clinical dossier is almost concluded for submission to the National Ethics Committee for Clinical Research (CEIC) and INFARMED (National Authority of Medicines and Health Products) in order to get the clinical study approval. After that, the clinical study will be started in CCC (surgical center of Coimbra) where ultrasound acquisitions in human eyes will be carried out in the clinical environment. In this study the equipment to be used is illustrated in Fig. 2.

The next task will deal with the conditioning and processing of the acoustic signals acquired during the clinical study, both in time and frequency domains. That will include the built algorithms to detect the interfaces of the human eye, such as cornea and lens, as well as the detection of cataract. These algorithms will be based on the previous ones that were developed for Wistar rat eyes [6].

After the successful detection of the several interfaces of the human eye and the cataract, algorithms will be built in order to extract a feature set from the acoustic signal. These features then will be used in machine learning methods for cataract classification into several degrees of severity.

V. Conclusion

The development of a medical device to detect a cataract and classify its hardness, in the scope of the FCT project CATARACTUS was described. The ultrasound signal acquisition system to be used in clinical environment, named xSCAN is fully described on section II, as well as an API to directly interact with the host computer. The main functionalities of the API were also presented. Finally, an overview of the tasks to be accomplished in the near futures under the CATARACTUS project were briefly described.

ACKNOWLEDGMENT

This work is funded by FCT/MEC through national funds and co-funded by POCI – COMPETE2020 and by FEDER – PT2020 partnership agreement under the projects POCI-01-0145-FEDER-028758 (CATARACTUS) and UID/EEA/50008/2019.

REFERENCES

[1] Parikshit Gogate, "Comparison of various techniques for cataract surgery, their efficacy, safety, and cost," Oman Journal of Ophthalmology, vol. 3, 2010.

[2] Miguel Caixinha, Danilo A. Jesus, Elena Velte, Mário J. Santos and Jaime B. Santos, "Using Ultrasound Backscattering Signals and Nakagami Statistical Distribution to Assess Regional Cataract Hardness," IEEE Transactions on Biomedical Engineering, Vol. 61, December 2014.

[3] Danilo Jesus, Miguel Caixinha, Mário Santos and Jaime Santos, "Ultrasound Techniques for Lens Hardness Characterization: A comparison Study", IEEE International Ultrasonics Symposium Proceedings, pp. 2376–2379, 2012.

[4] Leo T. Chylack, John K. Wolfe, David M. Singer , "The Lens Opacities Classification System III", Arch Ophthalmol, Vol. 111, June 1993.

[5] J. M. Sparrow,A. J. Bron, N.A.P. Brown, "The Oxford Clinical Cataract Classificationand Grading System", International Ophthalmology 9, pp. 207-225, 1986.

[6] Miguel Caixinha, João Amaro, Mário Santos, Fernando Perdigão, Marco Gomes and Jaime Santos, "In-Vivo Automatic Nuclear Cataract Detection and Classification in an Animal Model by Ultrasounds," IEEE Transactions on Biomedical Engineering, Vol.63, November 2016.

[7] "CATARATA – Development Of New Methodologies Based On Ultra-sound Techniques For In Vivo Cataract Characterization (PTDC/DTP-PIC/0419/2012)." FCT - Portuguese Foundation for Science and Technology. [Online].Available: https://www.fct.pt/apoios/projectos/consulta/vglobal_projecto.phtml.en?idProjecto=124807&idElemConcurso=7436

[8] "CATARACTUS – Development of a medical device based on ultrasounds for objective cataract characterization and optimal phacoemulsification energy evaluation", POCI-01-0145-FEDER-028758, [Online].Available: https://www.it.pt/Projects/Index/4579.

[9] OJ No L 117/1, Regulation (EU) 2017/745 of the European Parliament and of the Council of 5 April 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC.

[10] Clinical investigation of medical devices for human subjects -- Good clinical practice, ISO 14155:2011, 2011.

Anexo B

Este anexo contém ficheiro *.h* da API. Aqui podem ser consultados os diversos métodos da classe e os respetivos objetivos, estruturas e macros definidos.

```
#pragma once
#include <stdio.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <icmpapi.h>
#include <string>
#include <thread>
#include <rapidjson/document.h>
#include <rapidjson/prettywriter.h>
#include <mstcpip.h>
#include <time.h>
#pragma comment (lib, "iphlpapi.lib")
#pragma comment (lib, "Ws2_32.lib")

#ifndef InfoToUser
typedef struct InfoToUser
{
	short *pointerToBuffer; // Ponteiro para o buffer circular onde os A-SCANs
são guardados
	int numeroDeAscansNoBuffer; //Número de A-SCANs que podem ser guardados no
buffer
	const int *MostRecentASCAN; //Supondo que o buffer circular é uma matriz
em que cada coluna é um A-SCAN, este ponteiro aponta para a variável que diz em
que coluna esta o A-SCAN mais recente.
	bool *threadStatus; //Indica se a thread ainda esta a executar
	int sizeOfClientBuffer; //Indica o tamanho do buffer circular em bytes
	int lengthOfAscan; //Indica o tamanho de um A-SCAN em amostras;
}InfoToUser;


#endif //InfoToUser

#ifndef xSCAN_Parameters
typedef struct xSCAN_Parameters
{
	int PRF; //Indica o valor do parâmetro PRF
	int Pre_Amp_Filter; //Indica o valor do parâmetro Pre Amplifer Filter
	int Post_Amp_Filter;//Indica o valor do parâmetro Post Amplifer Filter
	int TX_Channel; //Indica o valor do parâmetro Tx-Channel
	int RX_Channel; //Indica o valor do parâmetro Rx-Channel
	int Pulse_Type; //Indica o valor do parâmetro Pulse Type
	int Pulse_Width; //Indica o valor do parâmetro Pulse Width
	int Pulse_Clamp; //Indica o valor do parâmetro Pulse Clamp
	int Pulse_Train_Iteration; //Indica o valor do parâmetro Pulse Train
Iterations
	bool Low_Noise_Amplifier; //Indica o valor do parâmetro Low Noise
Amplifier
	float Range; //Indica o valor do parâmetro Range
	float Delay;//Indica o valor do parâmetro Delay
	float Gain; //Indica o valor do parâmetro Gain
}xSCAN_Parameters;
```

```
#endif

//Valores validos para os parâmetros Pre Amplifier Filter e Post Amplifier Filter
#define       Off  0 //Off
#define       Thru  1 //Thru
#define       Mhz_0_25_TO_1  2 //0.25 MHz a 1 MHz
#define       Mhz_0_75_TO_2  4 //0.75 MHz a 2 MHz
#define       Mhz_1_75_TO_4  8 //1.75 MHz a 4 MHz
#define       Mhz_2_5_TO_5  16 //2.5 MHz a 5 MHz
#define       Mhz_4_5_TO_9  32 //4.5 MHz a 9 MHz
#define       Mhz_6_TO_12  64 //6 MHz a 12 MHz
#define       Mhz_10_TO_20 128 //10 MHz a 20 MHz
//xScan ON e OFF
#define x_Scan_ON 1 //Ligar o gerador de pulsos
#define x_Scan_OFF 0 //Desligar o gerador de pulsos
//Valores validos para o parâmetro TX-Channel
#define TX_CHANNEL_0FF 0 //Nenhum canal de transmissão
#define TX_CHANNEL_1 1 //Canal 1
#define TX_CHANNEL_5 5 //Canal 5
//Valores validos para o parâmetro RX-Channel
#define RX_CHANNEL_0FF 0 //Nenhum canal de recção
#define RX_CHANNEL_1 1 //Canal 1
#define RX_CHANNEL_5 5 //Canal 5
//Valores válidos para o parâmetro Pulse Type
#define PositiveSquareWave 0 //Onda quadrada positiva
#define NegativeSquareWave 1 //Onda quadrada negativa
#define BipolarSquareWave_Pos_Neg 2 //Onda quadrada bipolar positiva-negativa
#define BipolarSquareWave_Neg_Pos 3 //Onda quadrada bipolar negativa-positiva
#define Positive_Pulse_Train 4 //Comboio de pulsos quadrados positivos
#define Negative_Pulse_Train 5 //Comboio de pulsos quadrados negativos
#define Bipolar_Pulse_Train_Pos_Neg 6 //Comboio de pulsos quadrados bipolares
positivo-negativo
#define Bipolar_Pulse_Train_Neg_Pos 7 //Comboio de pulsos quadrados bipolares
negativo-positivo
//Valores possíveis para o parâmetro Low Noise Amplifier
#define Low_Noise_Amplifier_ON true //Ligado
#define Low_Noise_Amplifier_OFF false //Desligado
#define SamplingFrequencyMhz 100 //Frequência de amostragem em MHz

#define PRF_Property 0
#define Pre_Amp_Filter_Property 1
#define Post_Amp_Filter_Property 2
#define TX_Channel_Property 3
#define RX_Channel_Property 4
#define Pulse_Type_Property 5
#define Pulse_Width_Property 6
#define Pulse_Clamp_Property 7
#define Pulse_Iterations_Property 8
#define Low_Noise_Amplifier_Property 9
#define Gain_Property 10
#define Delay_Property 11
#define Range_property 12


class X_SCAN_API
{
private:
	clock_t start; //timestamp que indica quando é que foi enviada a última
mensagem para o porto 2000 do xSCAN
	clock_t end; //timestamp que indica o tempo atual
	const double timeOutOfXSCAN2000port = 60;
```

```
	const int ChunckMagicNumber = 666; //Número magico que é escrito cada vez
que uma secção é escrita no ficheiro onde são guardados os A-SCANs
	int numberOfChunksWritten; //Número de secções que foram escritas no
ficheiro os A-SCANs são guardados
	InfoToUser *APIinfo; //Ponteiro para uma estrutura do tipo InfoToUser.
	xSCAN_Parameters * xSCAN_Parameters_Group; //Ponteiro para uma estrutura
do tipo xSCAN_Parameteres;
	SOCKET Socket2000; //Socket que é usada para estabelecer uma ligação com o
porto 2000 do xSCAN
	SOCKET Socket3000; //Socket que é usada para estabelecer uma ligação com o
porto 3000 do xSCAN
	WSADATA wsaData;
	char * erroString; //Vetor de caracteres que contem informação adicional
sobre um codigo de erro

	char *clientBuffer; //Buffer circular onde os A-SCANs são guardados
	bool StopCommThread; //Permite ao utilizador interromper a aquisição
	std::thread WorkerThreadComms; //Thread que vai realizar a aquisição dos
A-SCANs
	bool ThreadInicializada; //Indica se a thread esta a executar ou não
	int clientBufferSizeINbytes; //Indica o tamanho do buffer circular em
bytes
	int Size_Ascan_plus_Header_in_bytes; //Indica o tamanho de um A-SCAN em
bytes incluindo o cabeçalho do A-SCAN

	int    PRF; //Guarda o valor do parâmetro PRF
	int PreAmpFilter; //Guarda o valor do parâmetro Pre Amplifier filter
	int PosAmpFilter;//Guarda o valor do parâmetro Post Amplifier filter
	int Tx_Channel; //Guarda o valor do parâmetro TX-Channel
	int Rx_Channel;//Guarda o valor do parâmetro RX-Channel
	int PulseType;//Guarda o valor do parâmetro Pulse Type
	int PulseWidth_ns;//Guarda o valor do parâmetro Pulse Width
	int PulseClamp_ns;//Guarda o valor do parâmetro Pulse Clamp
	int PulseIterations;//Guarda o valor do parâmetro Pulse Train Iterations
	int Low_Noise_Amplifier;//Guarda o valor do parâmetro Low Noise Amplifier
	float Gain_db; //Guarda o valor do parâmetro Gain
	float delay_us; //Guarda o valor do parâmetro Delay
	float range_us;//Guarda o valor do parâmetro Range

	bool FirstConfiguration; //
	int lastErrorID; //Guarda o último codigo de erro
	char **ptrToColClientBuffer; // Vetor de ponteiros que aponta para coluna
do buffer circular se este for visto como uma matriz em que cada coluna é um A-
SCAN
	int numeroDeAScansNoClientBuffer; //Indica o numero de A-SCANs que podem
ser guardados no buffer circular
	int whatUnitsDidIUseForClientBuffer;//Indica as unidades que o utilizador
usou para indicar o tamanho do buffer circular. 1 para unidades em A-SCANs e 2
para unidades em segundos.
	double myVariableForBufferSize; //Qual foi o valor que o utilizador
indicou para o tamanho do buffer circular. Este valor pode estar em unidades A-
SCAN ou em segundos.
	int contadorParaOUser; //Indica o numero da coluna do buffer onde está o
A-SCAN mais recente


public:
	/*
	Função: X_SCAN_API
	Objetivo: Criar um objeto da classe X_SCAN_API. Também inicializa as
variaveis.
	*/
```

```
        X_SCAN_API();
        /*
        Função: X_SCAN_API
        Objetivo: Destrutor de um objeto da classe X_SCAN_API.
        */
        ~X_SCAN_API();
        /*
        Função: X_SCAN_API
        Objetivo: Criar uma ligação com o porto 2000 do xSCAN.
        */
        int startConnectionConfigurator(void);
        /*
        Função: Save_Configuration_To_File
        Objetivo: Guardar os valores atuais de todos os parâmetros num ficheiro
cujo nome é dado pelo argumento de entrada filename.
        */
        int Save_Configuration_To_File(char *filename);
        /*
        Função: Load_Configuration_From_File
        Objetivo: Configurar os parâmetros com valores que estão escritos num
ficheiro cujo nome é dado pelo argumento de entrada filename.
        */
        int Load_Configuration_From_File(char *filename);
        /*
        Função: Set_X_SCAN_Property
        Objetivo: Configurar o parâmetro dado pelo argumento de entrada Property
com o valor dado pelo argumento de entrada Value.
        */
        int    Set_X_SCAN_Property(int Property, int Value);
        /*
        Função: Set_X_SCAN_Property
        Objetivo: Configurar o parâmetro dado pelo argumento de entrada Property
com o valor dado pelo argumento de entrada Value.
        Esta função só é usada para configurar parâmetros com um valor float.
Parâmetros esses, que só podem ser o Range, Gain e Delay.
        */
        int    Set_X_SCAN_Property(int Property, float Value);
        /*
        Função: Get_X_SCAN_Property
        Objetivo: Obter o valor atual do parâmetro dado pelo argumento de entrada
Property.
        */
        void * Get_X_SCAN_Property(int Property);
        /*
        Função: Get_Size_Of_Ascan_IN_BYTES
        Objetivo: Obter o tamanho de um A-SCAN em bytes (excluindo o cabeçalho).
        */
        int Get_Size_Of_Ascan_IN_BYTES(void);
        /*
        Função: getLastErrorCode
        Objetivo: Obter informação adicional sobre o último código de erro.
        */
        char * getLastErrorCode(void);
        /*
        Função: getErrorStringForCode
        Objetivo: Obter informação adicional sobre um código de erro determinado
pelo argumento de entrada errorCode.
        */
        char * getErrorStringForCode(int errorCode);
        /*
        Função: startGettingAScans_in_seconds
```

```
        Objetivo: Inicializar a extração de dados adquiridos pelo xSCAN. O tamanho
do buffer circular é determinado pelo número de segundos que o utilizador deseja
guardar.
        */
        const InfoToUser * startGettingAScans_in_seconds(double segundo);
        /*
        Função: startGettingAScans_in_seconds
        Objetivo: Inicializar a extração de dados adquiridos pelo xSCAN. O tamanho
do buffer circular é determinado pelo número de A-SCANs que o utilizador deseja
guardar.
        */
        const InfoToUser * startGettingAScans_in_NumberOfASCANS(int
numberOfAscans);
        /*
        Função: reset_Configuration_connection_XSCAN
        Objetivo: Reiniciar a ligação com o porto 2000 do xSCAN.
        */
        int reset_Configuration_connection_XSCAN(void);
        /*
        Função: stopAcquisitionThread
        Objetivo: Parar a execução da thread responsável por extrair os dados
adquiridos do xSCAN.
        */
        void stopAcquisitionThread(void);
        /*
        Função: statusOfAcquisitionThread
        Objetivo: Devolve o estado da thread responsável por extrair os dados
adquiridos do xSCAN. Se estiver a ser executada, retorna true. Caso contrário
retorna false.
        */
        bool statusOfAcquisitionThread(void);
        /*
        Função: pointerToTheLastASCAN
        Objetivo: Devolver um ponteiro para uma variável que indica em que coluna
do buffer circular esta o A-SCAN mais recente. O buffer circular deve ser visto
como uma matriz em que cada coluna é um A-SCAN
        */
        const int * pointerToTheLastASCAN(void);
        /*
        Função: numberOfASCANS_in_Buffer
        Objetivo: retorna o número de A-SCANs que podem ser guardados no buffer
circular
        */
        int numberOfASCANS_in_Buffer(void);
        /*
        Função: bufferWithTheASCANS
        Objetivo: retorna um ponteiro para o início do buffer circular, ou seja
para a primeira coluna da matriz buffer circular se este for visto como uma
matriz em que cada coluna é um A-SCAN
        */
        const short * bufferWithTheASCANS(void);
        /*
        Função: makeNMeans
        Objetivo: Realizar média de N A-SCANs recolhidos e guardar o resultado
dessa média na zona de memória apontada pelo ponteiro ptr. A média é feita a
partir do A-SCAN mais recente
        */
        int makeNMeans(int N, int *ptr, int length);
        /*
        Função: FilterSignal
        Objetivo: Filtragem de um A-SCAN com o filtro cujo coeficientes do
numerador são dados pelo argumento de entrada filter_numerator, coeficientes do
```

```
denominador são designados pelo argumento de entrada filter_denominator,
condições iniciais são dadas pelo argumento de entrada InitialConditions e ganho
do filtro que é dado pelo argumento de entrada Filtergain.
	*/
	int FilterSignal(const int *InputSignal, const int lengthInputSignal, int
* OutputSignal, const int lengthOutputSignal, const double *filter_numerator,
const double  *filter_denominator, const int filterlength, const double *
InitialConditions, const float Filtergain);
	/*
	Função: Get_Struct_All_xSCAN_Parameters
	Objetivo: Retorna os valores atuais de cada parâmetro numa estrutura do
tipo xSCAN_Parameters.
	*/
	const xSCAN_Parameters * Get_Struct_All_xSCAN_Parameters(void);
	/*
	Função: Write_Ascans_To_File
	Objetivo: Escrever A-SCANs num ficheiro. Supondo que o buffer circular é
uma matriz em que cada coluna é um A-SCAN, então esta função escreve a partir da
coluna dado pelo argumento de entrada initial. E depois escreve um número de
colunas para trás desse dado pelo argumento de entrada write_from_back e escreve
um numero de colunas para à frente dado pelo argumento de entrada
write_from_forward. Cada vez que esta função é chamada, é escrita uma secção de
A-SCANs no ficheiro.
	*/
	int Write_Ascans_To_File(FILE *fid, const int initial, const int
write_from_back, const int write_from_forward);
	/*
	Função: Write_Full_Buffer_In_Time_Order
	Objetivo: Escreve todos os A-SCANs que estão no buffer circular por ordem
temporal num ficheiro. Cada vez que esta função é chamada, é escrita uma secção
de A-SCANs no ficheiro.
	*/
	int Write_Full_Buffer_In_Time_Order(FILE *fid);
	/*
	Função: Write_Header_to_File
	Objetivo: Escreve o cabeçalho no ficheiro onde são guardados os A-SCANs.
	*/
	int Write_Header_to_File(FILE *fid);
	/*
	Função: How_Many_ASCANS_Written_in_File
	Objetivo: Retorna quantas secções de A-SCAN foram escritas no ficheiro.
	*/
	int How_Many_ASCANS_Written_in_File(void);
	/*
	Função: Reset_A_SCAN_In_File_Counter
	Objetivo: Reinicia a variável que guarda o número de secções escritas no
ficheiro
	*/
	void Reset_A_SCAN_In_File_Counter(void);


private:
	/*
	Função: clearBufferConnection
	Objetivo: Limpar o buffer de comunicação que foi alocado para a ligação
com o porto 2000 do xSCAN para a próxima troca de mensagens.
	*/
	void clearBufferConnection(void);
	/*
	Função: Get_PRF
	Objetivo: Obter o valor atual do parâmetro PRF.
	*/
```

```
        int Get_PRF(void);
        /*
        Função: Get_AmpFilter
        Objetivo: Obter o valor atual do parâmetro Pre Amplifier Filter ou Post
Amplifier Filter consoante o valor do argumento de entrada.
        Para o argumento de entrada igual a Pre_Amp_Filter_Property retorna o
valor do parâmetro Pre Amplifier Filter.
        Para o argumento de entrada igual a Post_Amp_Filter_Property retorna o
valor do parâmetro Post Amplifier Filter.
        */
        int Get_AmpFilter(int Qual_Retorna);
        /*
        Função: Get_TXChannel
        Objetivo: Obter o valor atual do parâmetro TX-Channel.
        */
        int Get_TXChannel(void);
        /*
        Função: Get_RXChannel
        Objetivo: Obter o valor atual do parâmetro RX-Channel.
        */
        int Get_RXChannel(void);
        /*
        Função: Get_Range
        Objetivo: Obter o valor atual do parâmetro Range.
        */
        float Get_Range(void);
        /*
        Função: Get_Delay
        Objetivo: Obter o valor atual do parâmetro Delay.
        */
        float Get_Delay(void);
        /*
        Função: Get_PulseWidth
        Objetivo: Obter o valor atual do parâmetro Pulse Width.
        */
        int Get_PulseWidth(void);
        /*
        Função: Get_PulseClamp
        Objetivo: Obter o valor atual do parâmetro Pulse Clamp.
        */
        int Get_PulseClamp(void);
        /*
        Função: Get_PulseType
        Objetivo: Obter o valor atual do parâmetro Pulse Type.
        */
        int Get_PulseType(void);
        /*
        Função: Get_PulseTrainIterations
        Objetivo: Obter o valor atual do parâmetro Pulse Train Iterations.
        */
        int Get_PulseTrainIterations(void);
        /*
        Função: Get_LowNoiseAmplifier
        Objetivo: Obter o valor atual do parâmetro Low Noise Amplifier.
        */
        int Get_LowNoiseAmplifier(void);
        /*
        Função: Get_GaindB
        Objetivo: Obter o valor atual do parâmetro Gain.
        */
        float Get_GaindB(void);
        /*
```

```
        Função: Set_PRF
        Objetivo: Configurar o parâmetro PRF com o novo valor PRFconf.
        */
        int Set_PRF(int PRFconf);

        /*
        Função: Set_AmpFilter
        Objetivo: Configurar os parâmetros Pre Amplifier Filter e Post Amplifier
filter com os novos valores PreAmplID e PostAmpID respetivamente.
        */
        int Set_AmpFilter(int PreAmplID, int PostAmpID);
        /*
        Função: Set_XScanONorOFF
        Objetivo: Ligar ou desligar consoante o valor de Status, o gerador de
pulsos.
        */
        int Set_XScanONorOFF(int Status);
        /*
        Função: Set_TXChannel
        Objetivo: Configurar o parâmetro TX-Channel com o valor de ChannelID. Caso
a flag seja false, é só guardado o valor a configurar. Caso contrário, a
configuração é feita.
        */
        int Set_TXChannel(int ChannelID, bool flag);
        /*
        Funçao: Set_RXChannel
        Objetivo: Configurar o parâmetro RX-Channel com o valor de ChannelID.
        */
        int Set_RXChannel(int ChannelID);
        /*
        Funçao: Set_RXChannel
        Objetivo: Configurar o parâmetro Pulse Type com o valor de PulseTypeID.
        */
        int Set_PulseType(int PulseTypeID);
        /*
        Função: Set_PulseClamp
        Objetivo: Configurar o parâmetro Pulse Clamp com o valor PulseClamp.
        */
        int Set_PulseClamp(int PulseClamp);
        /*
        Função: Set_PulseIterations
        Objetivo: Configurar o parâmetro Pulse Train Iterations com o valor
PulseIterationsCount.
        */
        int Set_PulseIterations(int PulseIterationsCount);
        /*
        Função: Set_LowNoiseAmplifier
        Objetivo: Configurar o parâmetro Low Noise Amplifier com o valor
StatusLNA.
        */
        int Set_LowNoiseAmplifier(bool StatusLNA);
        /*
        Função: Set_GainDb
        Objetivo: Configurar o parâmetro Gain com o valor Gain.
        */
        int Set_GainDb(float Gain);
        /*
        Função: Set_GainDb
        Objetivo: Configurar o parâmetro Delay com o valor delay.
        */
        int Set_Delay_us(float delay);
        /*
```

```
        Função: Set_Range_us
        Objetivo: Configurar o parâmetro Range com o valor Range.
        */
        int Set_Range_us(float Range);
        /*
        Função: Set_PulseWidth
        Objetivo: Configurar o parâmetro Pulse Width com o valor PulseWidth.
        */
        int Set_PulseWidth(int PulseWidth);
        /*
        Função: getError
        Objetivo: Obter mais informação sobre um código de erro determinado por
errorID.
        */
        char * getError(int errorID);
        /*
        Função: threadWorkComms
        Objetivo: Função que é executada pela thread responsável pela extração dos
dados adquiridos pelo xSCAN.
        */
        void threadWorkComms(void);
        /*
        Função: startGettingAScans
        Objetivo: Lançar a thread que vai extrair os dados adquiridos pelo xSCAN.
        */
        void startGettingAScans(void);
        /*
        Função: MakePingToXSCAN
        Objetivo: Fazer ping para o xSCAN para saber se este esta na mesma LAN que
nos.
        */
        int MakePingToXSCAN(int try_x_times = 1, long time_to_wait_between_trials=
1000);
        /*
        Função: MakeConnectionTo3000_for_good_XSCAN_start
        Objetivo: Fazer uma ligação temporária ao porto 3000 do xSCAN. Em caso de
sucesso, significa que o xSCAN esta pronto para receber comandos de configuração.
        */
        int MakeConnectionTo3000_for_good_XSCAN_start(int try_x_times = 1, long
time_to_wait_between_trials = 400);
        /*
        Função: Check_if_somebody_is_already_connected_to_XSCAN
        Objetivo: Verificar se existe alguma aplicação no computador a usar o
xSCAN.
        */
        int Check_if_somebody_is_already_connected_to_XSCAN(void);
        /*
        Função: dotted
        Objetivo: Converter um endereço IP em unsigned long para um vetor de
caracteres.
        */
        const char * dotted(unsigned long input);
        /*
        Função: Get_Parameter_Values_From_X_SCAN
        Objetivo: Obtém a resposta do xSCAN a um comando GET.
        */
        char * Get_Parameter_Values_From_X_SCAN(void);
        /*
        Função: Get_Confirmation_Values_From_X_SCAN
        Objetivo: Obtém a resposta do xSCAN a um comando SET.
        */
        char * Get_Confirmation_Values_From_X_SCAN(void);
```

```
	/*
	Função: allocatePointerToCollunsOfClientBuffer
	Objetivo: Cria um vetor de ponteiros em que cada ponteiro aponta para uma
coluna do buffer circular se pensarmos este como uma matriz em que cada coluna é
um A-SCAN.
	*/
	int allocatePointerToCollunsOfClientBuffer(void);
	/*
	Função: allocateClientBuffer
	Objetivo: Aloca o buffer circular para guardar os A-SCANs durante a
aquisição.
	*/
	int allocateClientBuffer(void);
	/*
	Função: CalculateClientBufferSizeInBytes_seconds
	Objetivo: Calcula o tamanho do buffer circular em bytes caso o utilizador
tenha definido o seu tamanho em segundos.
	*/
	int CalculateClientBufferSizeInBytes_seconds(double segundos);
	/*
	Função: CalculateClientBufferSizeInBytes_NumberOfASCANS
	Objetivo: Calcula o tamanho do buffer circular em bytes caso o utilizador
tenha definido o seu tamanho em A-SCANs.
	*/
	int CalculateClientBufferSizeInBytes_NumberOfASCANS(int number);
	/*
	Função: processForTerminatingCommunicationThread
	Objetivo: Função que é chamada quando a thread esta prestes a acabar de
executar. Esta thread tem como objetivo restabelecer o estado do xSCAN que estava
antes da thread ser executada.
	Consoante o valor de levelOfCleaning e erro são tomadas diversas ações.
	*/
	void processForTerminatingCommunicationThread(int levelOfCleaning, int
erro = 0);
	/*
	Função: Set_StepCount
	Objetivo: Configurar o parâmetro Step com o valor 1
	*/
	int Set_StepCount(void);
	/*
	Função: Get_Confirmation_Values_From_X_SCAN_TX_RX
	Objetivo: Obter o valor que vem na resposta aos comandos SET dos
parâmetros TX-Channel e RX-Channel. O valor a retornar depende do parâmetro de
entrada Qual.
	Caso Qual seja igual a TX_Channel_Property, então retorna o valor que vem
na resposta ao comando SET do parâmetro TX-Channel.
	Caso Qual seja igual a RX_Channel_Property, então retorna o valor que vem
na resposta ao comando SET do parâmetro RX-Channel.
	*/
	int Get_Confirmation_Values_From_X_SCAN_TX_RX(int Qual);
	/*
	Função: Make_Connection_to_2000_XSCAN_port
	Objetivo: Criar uma ligação com o porto 2000 do xSCAN.
	*/
	int Make_Connection_to_2000_XSCAN_port(int try_x_times = 10, long
time_to_wait_between_trials = 10);
```

```
        /*
        Função: kill_XSCAN_Acquisition
        Objetivo: Desalocar recursos alocados (buffer circular, ligações com o
porto 2000 e 3000 do xSCAN).
        */
        int kill_XSCAN_Acquisition(void);
};
```

Anexo C

Neste anexo é apresentado a estrutura do ficheiro onde são guardados os dados adquiridos pelo sistema de aquisição de sinais com xSCAN (capítulo 4).

Este ficheiro é constituído por um cabeçalho principal e diversos *chunks*. Cada *chunk* contém um cabeçalho *chunk* e os A-SCANs escritos. Todos os A-SCANs de um dado *chunk* tem o mesmo tamanho.

O cabeçalho principal tem um tamanho de 57 bytes e é constituído pelos seguintes elementos:

Tabela C.1 - Constituição do cabeçalho principal do ficheiro onde os A-SCANs são guardados

Campo do cabeçalho	**Tipo e Tamanho**
Versão do ficheiro	Vetor de caracteres (9 bytes)
Valor do parâmetro PRF	Inteiro (32 bits)
Valor do parâmetro *Pre Amplifier Filter*	Inteiro (32 bits)
Valor do parâmetro *Post Amplifier Filter*	Inteiro (32 bits)
Valor do parâmetro *Pulse Type*	Inteiro (32 bits)
Valor do parâmetro *Pulse Clamp*	Inteiro (32 bits)
Valor do parâmetro *Pulse Train Iterations*	Inteiro (32 bits)
Valor do parâmetro *Pulse Width*	Inteiro (32 bits)
Valor do parâmetro *Low Noise Amplifier*	Inteiro (32 bits)
Valor do parâmetro *Range* em us	Float (32 bits)
Valor do parâmetro *Delay* em us	Float (32 bits)
Valor do parâmetro *Gain* em dB	Float (32 bits)
Número de *chunks* no ficheiro, *C*	Inteiro (32 bits)

O cabeçalho de cada *chunk* tem um tamanho de 4 bytes e é constituído pelos seguintes elementos:

Tabela C.2 - Constituição do cabeçalho de cada *chunk* do ficheiro onde os A-SCANs são guardados

Campo do cabeçalho	**Tipo e Tamanho**
Número magico que indica se esta secção contém A-SCANs. O seu valor deve ser igual a 666	Inteiro (32 bits)
O tamanho de um A-SCAN da secção, *N*	Inteiro (32 bits)
Número de A-SCANs na secção, *M*	Inteiro (32 bits)
Timestamp para posicionar a secção no tempo	Inteiro (32 bits)

As amostras de cada A-SCAN são escritas em shorts de 2 bytes.

Na figura C.1 é possível visualizar um diagrama da estrutura do ficheiro.

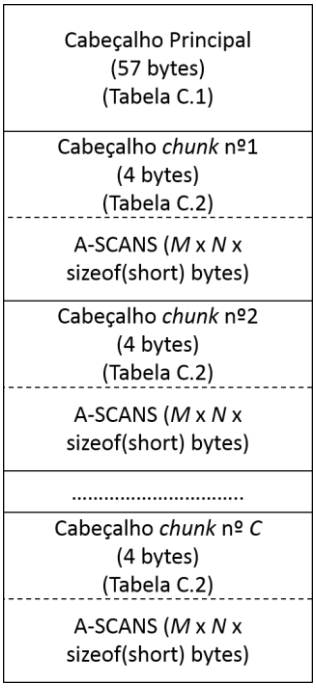

Figura C.1 - Diagrama da estrutura do ficheiro