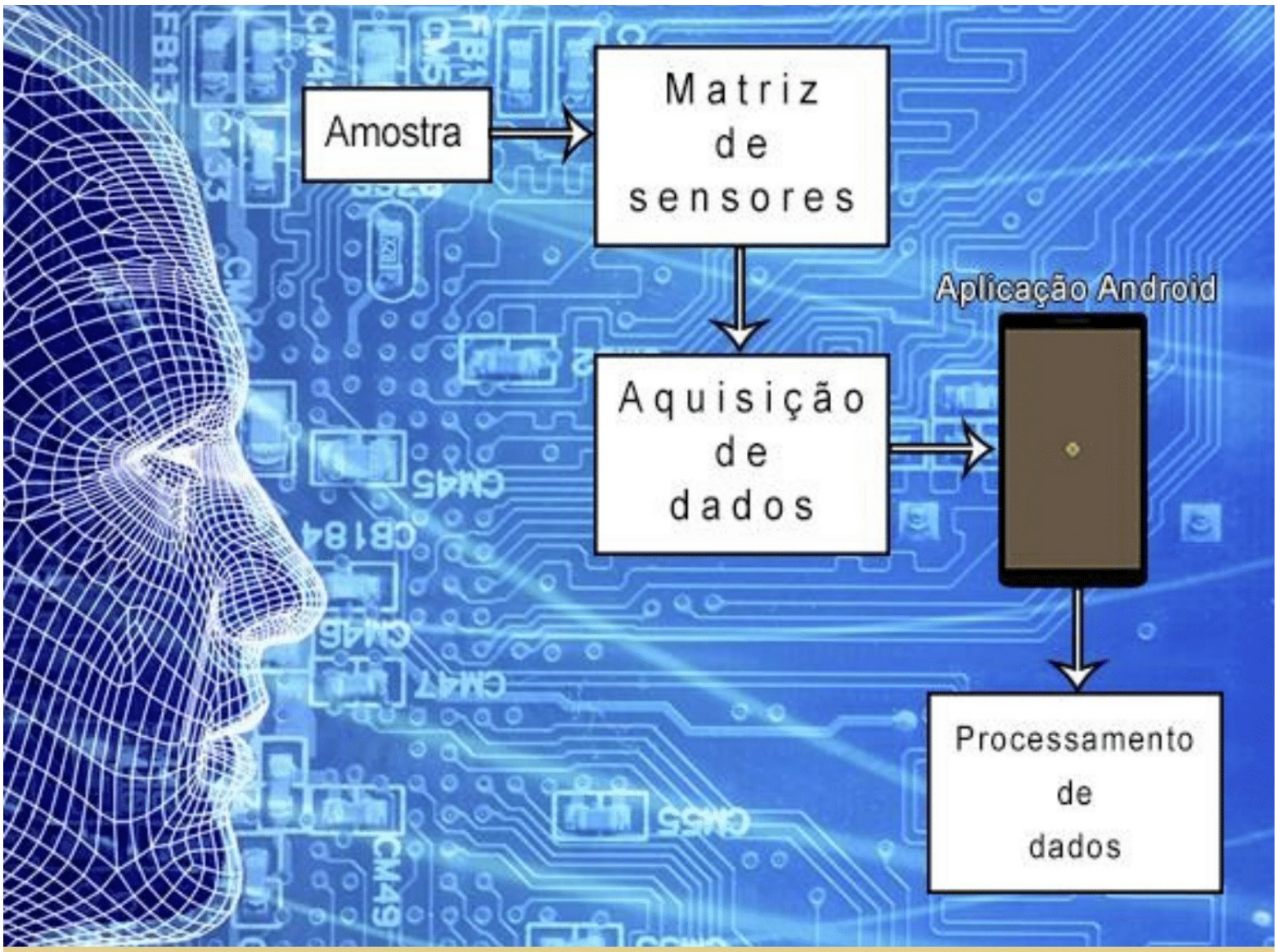


João Falcão de Sousa

Interface Gráfico para Nariz Eletrónico

Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores
09/2017

· U C ·

Universidade de Coimbra

FCTUC

UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Interface gráfico para nariz eletrónico

João Falcão de Sousa

Orientador:
Prof. Doutor Lino José Forte Marques

Júri:
Prof. Doutor António Paulo Mendes Breda Dias Coimbra
Prof. Doutor Gabriel Falcão Paiva Fernandes
Prof. Doutor Lino José Forte Marques

Coimbra, Setembro de 2017

Agradecimentos

Começo os meus agradecimentos pelos maiores responsáveis por ter chegado até aqui e finalizado esta etapa, os meus avós. A eles, ao meu tio e à táta, o meu obrigado pela educação e pelas oportunidades que me proporcionaram. Espero que esta etapa que agora termino, possa de alguma forma retribuir e compensar todo o carinho, apoio e dedicação que constantemente me oferecem. A eles, dedico todo este trabalho.

Agradeço, também, ao meu orientador, Lino Marques, pela oportunidade de trabalhar neste projeto, pela paciência e pela dedicação não só durante a realização desta dissertação, mas também ao longo das várias unidades curriculares.

À minha namorada, Cátia Almeida, agradeço do fundo do meu coração pelos conselhos dados ao longo de toda a dissertação mesmo que não entendesses quase nada do assunto; pelo apoio moral e emocional, estando sempre a meu lado nesta etapa da minha vida. Pela amizade e força que me dás e tens dado; obrigada por todas as deslocações de propósito só para podermos estar juntos. Por todo o apoio e amor expresso a minha gratidão.

A toda a equipa de Dr. Why "Já Fomos" que sempre me aturaram nos momentos difíceis, que me fizeram rir e estiveram sempre dispostos a fazer deslocações para que pudesse desanuviar a cabeça, o meu obrigado e já agora Portimão 2018 espera por nós.

A todos os amigos que já conhecia e aos que Coimbra trouxe o meu obrigado por todos os momentos de convivência. Em especial ao Tiago Andrade, agradeço por todos os anos de amizade e tardes passadas no café a conversar de nada e de tudo.

Muito obrigado a todos,
João Sousa

Abstract

The last few years brought a growth in mobile technologies, which caught the attention of the researchers for mobile phones as means of gathering data, replacing the traditional PCs altogether. With this evolution, mobile phones are evolving from a device that supported simple phone calls to devices with the ability to perform tasks that were previously carried out only by computers. Such tasks include reading e-mail, browsing the internet and even supervising and controlling remote automation processes. These new devices are called smartphones.

This dissertation developed a solution that allows researchers to use on the field portable solutions of electronic noses and collecting the data through a telephone. Throughout this work a technical solution used to reach a prototype of the mobile application is presented. The proposed prototype has the ability to connect to external devices through Bluetooth communications, receive data from those devices in real time and presenting to the user the visual data in form of text and graphics. Moreover, the application should still be able to perform a quick analysis of the data.

Since it is a prototype of an application, it has not been tested in the real world, but the data analysis and classification methods have been implemented and tested using files containing data acquired in other project. The proposed solution was carefully and systematically tested using several test steps to ensure the proper functioning of the prototype, to evaluate its qualities in terms of features, ease of use and performance.

Keywords:

Android; Software Engineering; Application; Electronic Nose; Framework; GUI; PCA; kNN

Resumo

Os últimos anos trouxeram um crescimento em todas as tecnologias, o que fez com que a evolução nas telecomunicações, em especial nos telefones, chamasse cada vez mais a atenção dos investigadores para a substituição do computador por telemóveis como meio de obtenção de dados. Com esta evolução na tecnologia, os telemóveis passaram de um simples equipamento para realizar chamadas telefónicas para um dispositivo com capacidade de realizar tarefas que até então apenas eram efetuadas num computador, tais como ler correio eletrónico (emails), navegar na internet e até controlar a casa remotamente. Estes novos dispositivos são denominados telefones inteligentes ou smartphones.

O foco desta dissertação passa por desenvolver uma solução que permita aos investigadores utilizarem no terreno soluções portáteis de narizes eletrónicos recolhendo os dados através de um telefone. Ao longo deste trabalho são apresentadas as soluções técnicas empregadas para alcançar um protótipo da aplicação móvel com a capacidade de conectar a dispositivos externos através de comunicação bluetooth. Para além disso a aplicação deverá ser capaz de receber dados em tempo real, apresentando-os ao utilizador em forma de texto e gráficos. A aplicação deverá também ser capaz de realizar uma análise e tratamento dos dados recebidos.

Visto tratar-se de um protótipo de aplicação, esta não foi testada em ambiente real mas os métodos de análise e classificação dos dados foram testados utilizando ficheiros de dados previamente adquiridos. A solução proposta foi cuidadosamente e sistematicamente testada utilizando várias etapas de teste para garantir o bom funcionamento do protótipo da aplicação, avaliar as suas qualidades em termos de recursos, a facilidade de uso e o seu desempenho.

Palavras chave:

Android; Engenharia de Software; Aplicação; Nariz Eletrónico; Arquitetura; Interface; PCA; kNN

Índice

Glossário

Aplicação Programa que corre sobre um sistema operativo e interage com o utilizador através de um GUI.

Barra de Ferramentas Conjunto de atalhos situado na parte superior da interface gráfica da aplicação onde se accionam todas as funcionalidades do software.

Clientes/Stakeholders Conjunto de pessoas que contrata developers para desenvolver software.

Developers Conjunto de pessoas responsáveis pelo desenvolvimento da aplicação.

Google Play Store É a loja online disponibilizada pela Google para distribuição de aplicações, jogos, filmes, música e livros para dispositivos com o sistema Android.

Menu Conjunto de opções ou ações.

Requisito Qualquer imposição feita pelos clientes/stakeholders na maneira como a aplicação deverá comportar-se, quer a nível de interface, quer a nível de funcionalidade.

Smartphone "Telefone Inteligente" é um telemóvel que combina os recursos de computadores pessoais, com funcionalidades avançadas estendidas por meio de aplicações executados pelo sistema operativo.

Software Qualquer programa ou aplicação.

Threads É uma forma de um processo se dividir em duas ou mais tarefas que podem ser executadas ao mesmo tempo.

Utilizador Pessoa que interage com a aplicação através do GUI. Partimos do princípio que este não tem qualquer conhecimento técnico de software ou hardware.

Acrónimos

APK *Android Package* ou Arquivo de Pacotes para Android.

FAQ *Frequently Asked Questions* ou Perguntas Mais Frequentes.

GUI *Graphical User Interface* ou Interface Gráfica do Utilizador.

GPS *Global Positioning System* ou Sistema de Posicionamento Global.

IDE *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado.

UML *Unified Modeling Language* ou Linguagem de Modelagem Unificada.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O sistema olfativo mamífero dá aos humanos o sentido de cheiro coletando odores provenientes do ambiente. Este é um sistema complexo, quando a membrana de perceção, localizada na cavidade do nariz, é confrontada com diferentes compostos, as células de deteção geram sinais que são transmitidos e reconhecidos pelo cérebro [11]. A funcionalidade do órgão de cheiro e o seu mecanismo conduziu os cientistas a desenvolver sistemas sensoriais que replicassem o sistema olfativo humano, denominada então nariz eletrónico.

1.1 Motivação

Das tecnologias inventadas nas décadas passadas, os *smartphones* têm ganho uma quota do mercado muito elevada em vários setores devido à sua usabilidade e ao seu preço. De acordo com a eMarketer [1], o número de utilizadores de telemóveis em 2017 é de 4.43 biliões e até 2020 irá chegar a 4.78 biliões, como se pode ver também pela figura 1.1. Pelo estudo feito em 2016, esta estima que em 2016, dos 4.30 biliões de pessoas que usam telemóveis, 49.7% usam smartphone pelo menos uma vez ao mês, no caso particular da Europa Ocidental esse valor corresponde a 71.7% da população e em 2020 deverá atingir os 82.7%.

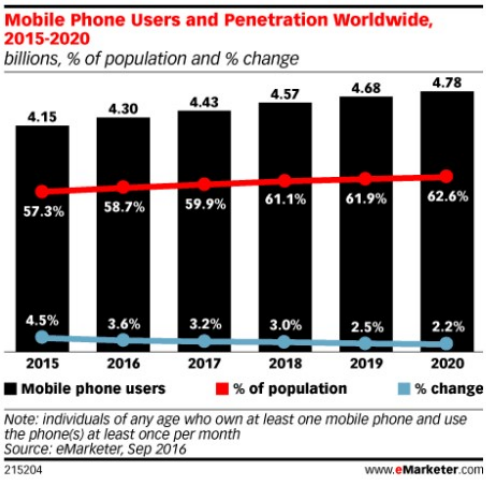


Figura 1.1 Número de utilizadores de telemóveis pela eMarketer [1].

Nos últimos anos os *smartphones* são cada vez mais usados como plataformas para deteção remota de dados. Muitas das soluções utilizam os sensores embutidos nos *smartphones*, como é o caso do giroscópio e medição de luz. Com isso muitas indústrias adotaram os *smartphones* para facilitar o seu trabalho, como é o caso da saúde onde num artigo publicado em 2011 [12] , foi feita uma pesquisa de aplicações desenvolvidas para telemóveis e que se destinassem a profissionais da área, estudantes ou pacientes. Ainda na saúde [13] propõem soluções utilizando smartphones na deteção e prevenção de quedas. Para além da saúde, também na educação tem existido avanços no uso do smartphone. Como é o caso do artigo [14] onde é feita uma revisão das várias publicações sobre o estado atual do uso de dispositivos móveis na educação. Para além disso, no artigo publicado por Milrad e Spikol [15] estes tiveram como objetivo estudar e identificar quais os conteúdos e serviços mais adequados para apoiar a aprendizagem e a comunicação no contexto universitário, utilizando smartphones.

Os *smartphones* são constituídos por vários circuitos eletrónicos embutidos, componentes acústicos e óticos que podem ser usados como sensores sozinhos ou como parte de outros sensores. Com isto os *smartphones* apresentam várias vantagens:

- É possível realizar modificação, o que é conveniente para integrar componentes externos.
- São computadores portáteis que permitem correr aplicações desenvolvidas para operar com os sensores, permitindo o processamento do sinal e facilita a interação com o utilizador.

- Geralmente são equipados com GPS, o que permite determinar o posicionamento do utilizador ou o mapeamento de uma área.
- Estão equipados com comunicação wireless, o que permite construir uma rede de sensores wireless.
- Têm um ambiente de fácil compreensão para os utilizadores, o que permite que para a sua utilização não seja necessário nenhum treino específico.

Cada vez mais as pessoas se preocupam com a qualidade do ar, não só fora mas também dentro de casa. A qualidade do ar está relacionada não só com a saúde das pessoas, mas também com o seu conforto, portanto é importante analisar regularmente a qualidade do ar para evitar situações de perigo. Com isso alguns investigadores já propuseram soluções móveis de narizes eletrónicos que permitem ao utilizador medir em tempo real a qualidade do ar, como é o caso de [16] que utiliza sensores de gás para monitorizar a qualidade do ar. Noutra investigação realizada por Dinko e Vedran [17] é proposto um sistema de partilha de informações da qualidade do ar utilizando dispositivos móveis que as pessoas possam transportar, com isso criasse uma rede de dados e obtêm-se uma maior monitorização da qualidade do ar ao longo de toda a cidade.

1.2 Objetivos

Esta dissertação teve como principal objetivo a elaboração de uma aplicação móvel que permita a comunicação com narizes eletrónicos através de *Bluetooth*, permitindo assim visualizar os dados recolhidos em tempo real. Para isso foi também necessário elaborar um sistema de comunicação, que será abordado no capítulo 3.

De forma geral, a aplicação realizada terá os seguintes objetivos:

1. Conexão a dispositivos equipados com *Bluetooth*;
2. Receção de dados em tempo real;
3. Realização de uma calibração dos dados;
4. Análise e tratamento dos dados recebidos;
5. Visualização dos dados através de texto e gráficos;
6. Guardar os dados recebidos num ficheiro de dados.

1.3 Objetivos alcançados

O resultado deste trabalho é uma aplicação móvel que permitirá aos investigadores a aquisição de dados através de dispositivos equipados com comunicação *Bluetooth*, neste caso em particular os narizes eletrónicos, sem a necessidade de levar consigo um computador para realizar a aquisição e tratamento dos dados. Com a solução proposta nesta dissertação e com o acesso a um smartphone ou a um tablet, será possível realizar uma calibração e um tratamento prévio dos dados; no terreno o investigador poderá, em tempo real, visualizar os dados obtidos.

Mais especificamente, foram atingidos os seguintes objetivos:

- Permitir o acesso a uma aplicação amigável e de fácil utilização.
- Possibilitar uma maior mobilidade no terreno utilizando apenas um smartphone, sem a necessidade de um computador para adquirir os dados.
- Permitir a realização de uma calibração dos dados adquiridos.
- Possibilitar uma análise automática dos dados utilizando algoritmos implementados para esse efeito.
- Permitir o acesso aos dados adquiridos após a sua análise, sendo possível enviá-los para o computador pessoal através de email.
- Ter uma aplicação que possa ser adaptada a outros dispositivos externos, sendo apenas necessário implementar o protocolo de comunicação no dispositivo pretendido, o protocolo de comunicação é abordado no capítulo 3.

1.4 Estrutura da tese

Este documento é composto por 5 capítulos que abordam o trabalho realizado no âmbito desta dissertação:

- No capítulo 2, são introduzidos conceitos necessários para a realização desta dissertação, como é o caso dos narizes eletrónicos, onde se analisa o modo de obtenção de dados, fala-se dos métodos utilizados ao longo desta dissertação para a análise dos dados e aborda-se os gráficos implementados Quanto ao android dá-se um breve olhar sobre o sistema e por fim fala-se de Engenharia de Software e sobre a sua importância no desenvolvimento de aplicações.

- O capítulo 3, aborda os passos utilizados na realização da aplicação proposta, desde a análise de requisitos até às metodologias utilizadas na sua implementação.
- De seguida, no capítulo 4, são apresentados os testes de software realizados para a validação da aplicação e os resultados obtidos realizando uma calibração dos dados e classificação de novas amostras.
- Por último, o capítulo 5 aborda as conclusões da dissertação e refere o trabalho futuro.

Capítulo 2

Enquadramento

2.1 Narizes eletrónicos

O conceito de nariz eletrónico como ferramenta para classificar odores composta por sensores foi introduzida por Persaud e Dodd em 1982 [18]. No decorrer das suas experiências os dois investigadores estabeleceram o objetivo de criar uma ferramenta capaz de imitar o sistema olfatório mamífero e assim reconhecer diferentes odores. Com esta ferramenta pode-se verificar que cada parte do sistema olfativo humano possui uma correspondência relativamente a um nariz eletrónico, como se pode ver pela figura 2.1. O nariz desenvolvido pelos investigadores continha:

- Uma matriz de sensores para simular os recetores do sistema olfativo humano;
- Uma unidade de processamento que realizaria a mesma função que o bulbo olfativo;
- Um sistema de reconhecimento de padrões que reconheça os padrões olfativos da substância a ser testada, nos humanos esta função é realizada pelo cérebro [19].

Devido à capacidade do nariz eletrónico de discriminar e reconhecer uma variedade de gases e odores diferentes usando apenas um pequeno número de sensores e, devido aos primeiros resultados promissores obtidos nesta área, o interesse no assunto foi aumentando ao longo do tempo. Isso fez com que, o número de pesquisas focadas nas aplicações dos narizes eletrónicos em áreas diferentes aumentasse, alguns exemplos disso são por exemplo: no setor médico e diagnóstico como é o caso do artigo [20] onde é estudado o design de um protocolo de medição de algumas patologias como é o caso do cancro do pulmão, da esquizofrenia e da melanoma. Ainda nesta área, num artigo publicado em 2006 [21] foi abordado a aplicação dos narizes eletrónicos para deteção de micróbios em diferentes áreas, como é o caso da medicina e da indústria alimentar. Persaud em 2005 [22] fez uma

revisão de algumas aplicações clinicas onde os narizes eletrónicos possam ser aplicados, possibilitando uma monitorização não invasiva dos pacientes. Quantos aos alimentos, Peris e Escuder-Gilabert [23] examinam as características principais dos narizes eletrónicos e as suas aplicações no controlo alimentar. Nesse sentido também nos artigos [24] [25] são referenciadas as vantagens dos narizes eletrónicos no controlo alimentar, como por exemplo no controlo da qualidade, do processo, na avaliação da frescura do produto, entre outros.

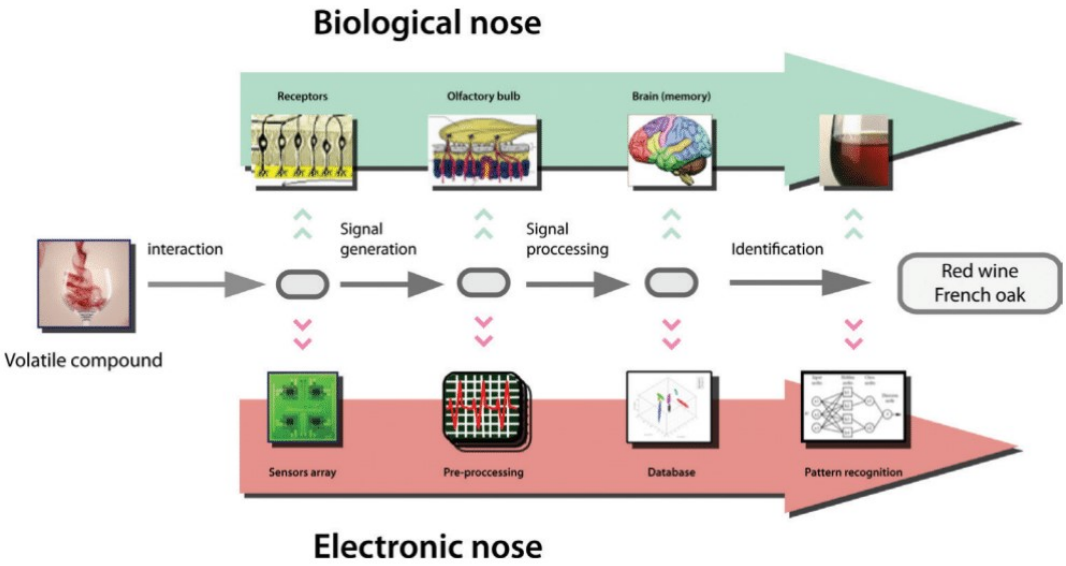


Figura 2.1 Semelhanças entre o nariz eletrónico e o sistema olfativo humano [2].

Aquisição de Dados

Antes de se poder analisar os dados, primeiro é necessário proceder à sua aquisição. Nesta fase os sensores que compõem o nariz eletrónico, são capazes de detetar o odor a ser testado. A matriz de sensores está então conectada a uma unidade de aquisição capaz de converter o sinal analógico dos sensores num digital. O sinal convertido é então processado por um dispositivo de processamento. A conversão do sinal analógico em digital muitas vezes causa dificuldades no que toca à classificação, uma vez que as características e limitações dos conversores podem limitar ou distorcer a informação. O resultado da aquisição de dados é um vetor de amostras com tamanho N*M, onde N é número de amostras e M o número de sensores que constitui o nariz eletrónico, este tem uma representação no espaço de amostras. Na figura 2.2 pode-se visualizar um exemplo dos dados obtidos por um matriz de quatro sensores para uma amostra de .

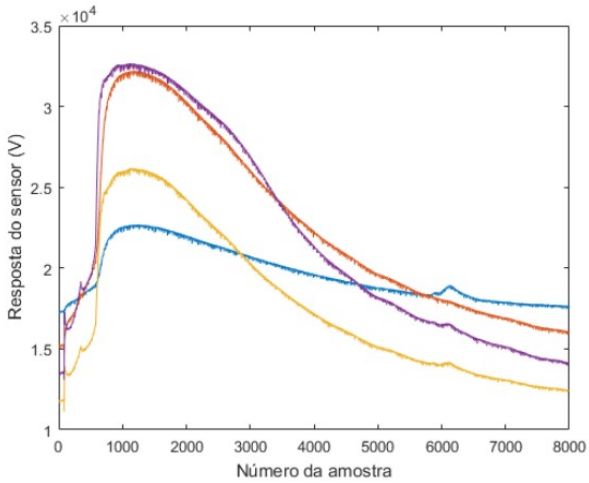


Figura 2.2 Gráfico das amostras obtidas.

Contudo, a concentração pode ter um efeito de escala nos padrões dos sensores, o que pode fazer com que a relação entre as amostras seja diferente da realidade. Para que se possa remover esse efeito deve-se realizar um pré-processamento dos dados, a normalização é a técnica mais utilizada. Aplicar uma normalização dos dados não só faz com que a resposta do vetor de amostras possua a mesma magnitude mas também permite reduzir o erro de cálculo nos métodos de classificação. A Tabela 2.1 mostra algumas técnicas padrão de normalização de dados [19] [26] [27]. Em [9] pode-se visualizar mais alguns métodos de normalização dos dados.

Tabela 2.1 Algumas técnicas de normalização dos dados [9].

Normalização	Equação
Escala $relativa_1$	$X_{ij} = \frac{X_{ij}}{\max(X)}$
Escala $relativa_2$	$X_{ij} = \frac{X_{ij}}{\max(x_j)}$
Escala $relativa_4$	$X_{ij} = \frac{X_{ij}}{\|\mathbf{x_i}\|}$
Escala de variaçãoo_1	$X_{ij} = \frac{X_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)}$
Escala de variaçãoo_2	$X_{ij} = \frac{2(X_{ij} - \min(x_j))}{\max(x_j) - \min(x_j)} - 1$

Da tabela acima tem-se, X é matriz contendo as amostras recebidas, tem tamanho N por M onde, N é o número de amostras e M o número de sensores. X_{ij} é a amostras i do j sensor, x_j contém todas as N respostas para o sensor j e x_i contém as M respostas para a amostras i.

Para além disso, a escala $relativa_1$ dá uma compreensão global dos dados, estes ficam com valor máximo de 1. A escala $relativa_2$ comprime os valores por sensor, e também faz com que o valor máximo seja 1. Quanto à escala $relativa_4$ esta corresponde à norma da distância euclidiana. Por fim tem-se a escala de variaçãoo_1 e variaçãoo_2 que define os limites como [0,1] e [-1,1] respetivamente.

Nas figuras 2.3 e 2.4 estão representados dois exemplos de diagramas referentes a dois projetos distintos que implementam narizes eletrónicos. Pela análise da figura pode-se analisar que normalmente os narizes eletrónicos são constituídos por uma matriz de sensores, por um sistema de aquisição de dados e, para além disso, os dois sistemas recebem os dados adquiridos num computador e é neste que os mesmos são tratados. O mesmo pode também ser visto em [28].

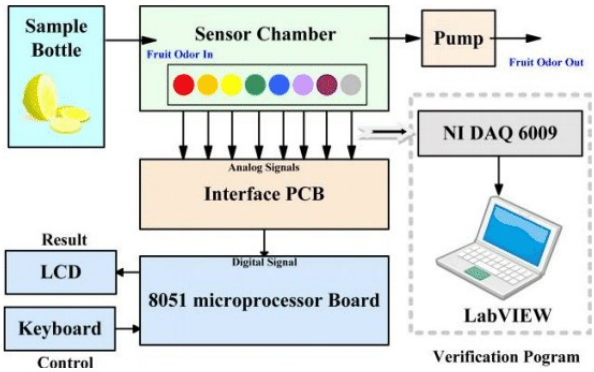


Figura 2.3 Arquitetura de um nariz eletrónico com aquisição e processamento local, por microcontrolador e visualização remota através do LabVIEW. [3].

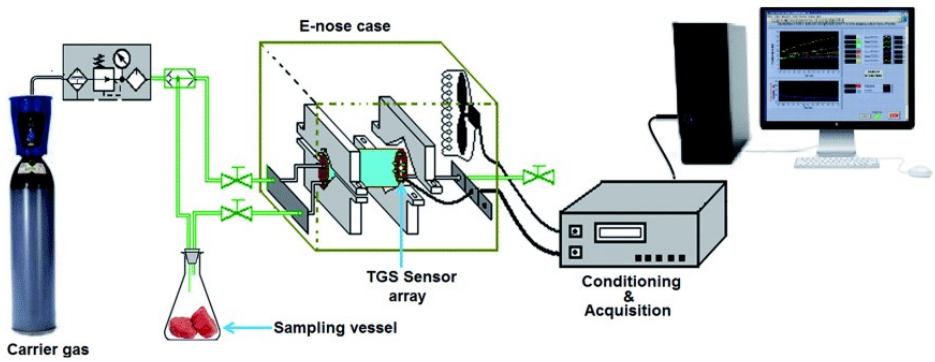


Figura 2.4 Arquitetura de um nariz eletrónico desenvolvido para análise dos padrões do cheiro de amostras de carne. A aquisição e processamento é realizado através do LabVIEW, para além disso o programa desenvolvido controla ainda a temperatura e a umidade do meio [4].

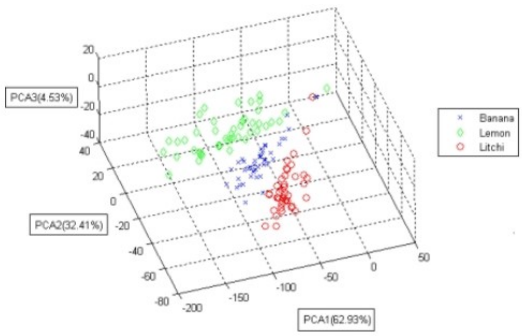
Seleção de Características

Após a obtenção dos dados e a realização da normalização dos mesmos, o próximo passo é a seleção das características, este é o processo de identificação das características mais importantes dos dados. Na investigação realizada por Blum e Langley [29] estes classificaram a técnica de selecionar características em três abordagens.

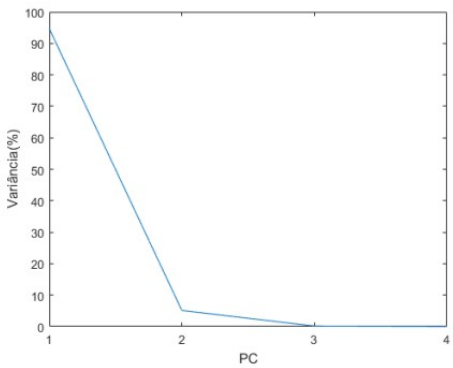
Na primeira, conhecida como abordagem incorporada, as características são adicionadas ou removidas em resposta a erros de previsão de um classificador embutido simples. O segundo são métodos de filtragem e trabalham de forma independente para remover características sem saber o efeito no algoritmo de classificação. Neste os métodos lineares utilizados são: análise de componentes principais (PCA) [30], análise discriminante linear (LDA) [31] e análise das componentes independentes (ICA) [32] [33]. O terceiro é o método *wrapper* e avalia o conjunto de características de um candidato usando um algoritmo de classificação nos dados de treino.

Análise de componentes principais (PCA)

Para a realização deste trabalho foi utilizada a análise de componentes principais. Este é um procedimento multivariável não supervisionado e é uma técnica de compressão de dados lineares e extração de características [9]. Os resultados produzidos por este método podem ser apresentados em duas ou três dimensões o que possibilita inspecionar os dados. No gráfico 2.5a, retirado de [3] pode-se visualizar o resultado obtido aplicando este método. O gráfico 2.5b apresenta um exemplo da percentagem da variância de cada PC obtido. Os dois gráficos não se referem ao mesmo exemplo, são meramente ilustrativos.



(a) Gráfico contendo as três componentes mais significativas da matriz PCA [3].



(b) Valores Próprios - percentagem.

Figura 2.5 Gráficos utilizados usando os dados obtidos através do PCA.

Com o PCA obtêm-se novas variáveis que são combinações lineares dos dados originais ordenadas pela variabilidade. O objetivo deste método é, essencialmente, reduzir a dimensão do conjunto de dados, mantendo a maior quantidade possível de informações. Eliminando assim as variáveis com classificação mais baixas. A escolha recaiu neste método por a ser um método simples e rápido, permitindo assim um tempo de computação baixo, o que faz com que a aplicação consiga realizar os cálculos pretendidos em tempo real e apresentá-los de imediato ao utilizador.

Para realizar a implementação deste método é necessário seguir os seguintes passos [30]:

1. Obter matriz de amostras;
2. Subtrair a média dos valores;
3. Calcular a matriz de covariância;
4. Calcular os valores e vetores próprios;
5. Formar o vetor de características.

O método de obtenção da matriz de amostras já foi falado anteriormente, por isso considera-se que já se tem todas as amostras adquiridas. O passo seguinte é subtrair a média de cada sensor à amostra correspondente, isto é, se a matriz de amostras for N por M, onde N é o número de linhas correspondente a cada amostra e M o número de colunas correspondente a cada sensor tem-se então

$$Novo\_x_{ij} = x_{ij} - \overline{x_j}, \tag{2.1}$$

onde x_{ij} é a amostra na linha i coluna j e $\overline{x_j}$ é a média de valores da coluna j.

No cálculo do PCA, a decomposição dos valores e vetores próprios é feita na matriz de covariância Σ, esta é do tamanho d por d onde cada elemento representa a covariância entre duas características e é calculada da seguinte forma [34]:

$$\sigma_{jk} = \frac{\sum_{i=1}^{n}(x_{ij} - \overline{x_j})(x_{ik} - \overline{x_k})}{(n-1)},$$

O cálculo da matriz de covariância é dado então por:

$$\Sigma = \frac{1}{n-1}\big((X - \overline{x})^T (X - \overline{x})\big),$$

onde n é o número de amostras e $\overline{x}$ é o um vetor com as médias de cada coluna

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{2.2}$$

Obtida a matriz de covariância procede-se ao cálculo dos valores próprios, estes podem ser calculados da seguinte forma [35]:

$$A.v = \lambda.v, \tag{2.3}$$

A é uma matriz n por n correspondente à matriz de covariância Σ, **v** é um vetor não nulo de tamanho n por 1 e λ é um escalar. Qualquer valor de λ para o qual a equação 2.3 tenha solução, é denominado valor próprio da matriz **A** e o vetor **v** é denominado vetor próprio.

$$\begin{aligned} A.v - \lambda.v &= 0, \\ A.v - \lambda.I.v &= 0, \\ (A - \lambda.I).v &= 0, \end{aligned}$$

se **v** for não não nulo, a equação só têm solução se:

$$|A - \lambda.I| = 0, \tag{2.4}$$

o resultado desta equação são os valores próprio de **A**. Pela equação 2.3 pode-se também obter

$$(A - \lambda).v = 0, \tag{2.5}$$

com os valores próprios obtidos pela equação 2.4 pode-se obter pela equação 2.5 os vetores próprios correspondentes.

De seguida forma-se o vetor de características, para isso deve-se ordenar os valores próprio do maior para o menor e reordenar os vetores próprio tendo em conta as alterações feitas nos valores próprios. Tendo então tudo ordenado, deve-se escolher o número de características a guardar [30]. Com esse número escolhido procede-se à construção do vetor de características, esse tem a seguinte forma

$$VetorCaractersticas = (eig_1, eig_2, eig_3, \ldots, eig_n),$$

neste vetor são guardados os **n** vetores próprios correspondentes ao número de características desejadas.

Por fim é necessário passar o vetor de dados recebidos do espaço de amostras para o espaço de características. Na equação 2.6 é apresentada a forma de calcular a matriz PCA.

$$PCA = VetorCaractersticas * MatrizAmostras \tag{2.6}$$

Classificador

Um sistema típico de classificação de características, conforme resumido na figura 2.6, pode ser dividido numa série de estágios, como referido em [9]:

1. Sistema de aquisição de dados;
2. Sistema de extração de características;
3. Classificador;
4. Estratégia de tomada de decisão.

Figura 2.6 Conceito de um classificador de amostras.

O limite conceitual entre um extrator de características e um classificador propriamente dito é quase arbitrário. Idealmente, o extrator de características deveria fazer uma representação dos dados o que reduziria o trabalho dos classificadores; ao contrário, um classificador universal não necessitaria de um extrator de características robusto [36].

A tarefa de um classificador é usar o vetor de características fornecido pelo extrator de características para atribuir o objeto que representa a uma categoria. O grau de dificuldade do problema de classificação depende da variabilidade nos valores das características para objetos na mesma categoria em relação à diferença entre os valores das características para objetos em diferentes categorias.

Espera-se que os narizes eletrónicos sejam utilizados em ambientes específicos e, que sejam pequenos, portáteis, dispositivos de baixa potência e fáceis de operar. Dadas essas características dos narizes, Shaffer, Rose-Pehrsson e McGill [37] propuseram 6 requisitos que um sistema de classificação ideal deveria possuir:

1. Alta precisão - deve haver o menor número de erros de classificações quanto possível.
2. Rápido - para análise em tempo real, o algoritmo deve ser capaz de produzir uma classificação com um mínimo atraso possível.
3. Simples de treinar - em muitas aplicações, a base de dados de treino das classificações é atualizada periodicamente e o classificador realiza de novo o treino, este procedimento deve ser rápido e simples de executar.

4. Requisitos mínimos de memória - para pequenos sistemas portáteis que podem ser usados como dispositivos portáteis, o classificador precisa de consumir poucos recursos.

5. Robusto para outliers - em ambientes descontrolados, o algoritmo deve ser capaz de reduzir o potencial de erros de classificação ao ser capaz de diferenciar entre uma amostra para o qual foi treinado para reconhecer e uma que não.

6. Produzir uma medida de incerteza - para muitas aplicações, o algoritmo precisa produzir uma medida do nível de correspondência da classificação, ou uma medida estatística sobre a certeza da classificação.

k–nearest neighbour

O algoritmo k-vizinho mais próximo (kNN) é um algoritmo supervisionado que tem um conceito e implementação simples. Para treinar este modelo, simplesmente é necessário armazenar os dados de treino após a normalização dos dados e a passagem pelo PCA.

A distância entre um vetor de amostras não classificado e todas as amostras do conjunto de treino é calculada. Para realizar o cálculo das distâncias usa-se então a distância euclidiana, esta é dada pela fórmula 2.7.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \quad (2.7)$$

Após esses cálculos o vetor de amostras não classificado é atribuído à classe dos k vizinhos com as distâncias mais próximas.

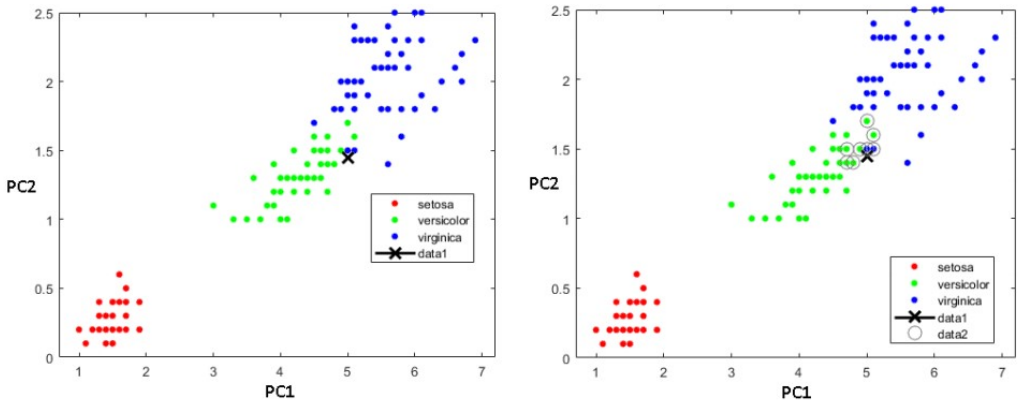
A escolha do valor de k é geralmente empírica, k = 1 é o valor mais utilizado e é conhecido como 1-NN ou vizinho mais próximo, só que 1-NN tem uma taxa de erro que é duas vezes maior do que o valor ótimo definido por Bayes [31]. Definindo maiores valores de k obtêm-se limites mais suaves e a complexidade temporal é dada por:

$$O(n^2),$$

pelo que se o número de observações duplicar, o tempo computacional, quadruplica.

Na figura 2.7 está representado o resultado do algoritmo kNN aplicado em Matlab [5]. Na figura 2.7a está uma representação dos dados no espaço do PCA e está ainda apresentado o ponto que se deseja classificar. Na figura 2.7b está o resultado da aplicação do algoritmo kNN, onde se pode ver que dado o ponto que se deseja classificar e um k = 8, obtêm-se os 8

vizinhos mais próximo do ponto. Por fim é feita uma ponderação entre os vizinhos e, é feita a decisão da classificação do ponto.

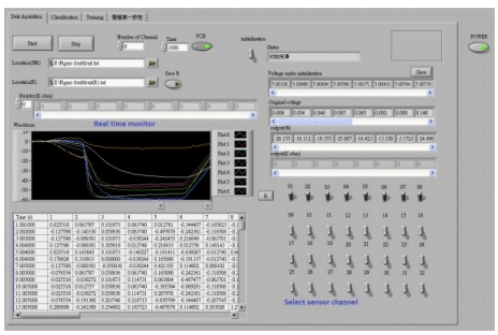


(a) (b)

Figura 2.7 Exemplo do algoritmo kNN retirado do site do Matlab [5].

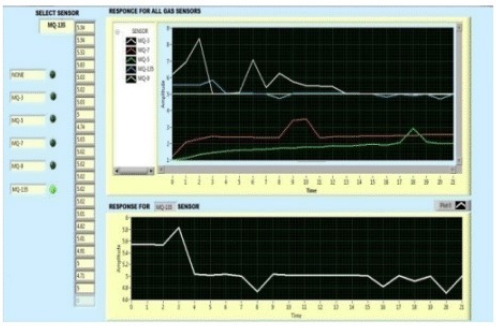
Ambientes Gráficos

Na figura 2.8, são apresentados alguns exemplos de ambientes gráficos desenvolvidos para apresentar ao utilizador os dados obtidos por vários narizes eletrónicos. Como se pode analisar pelas imagens dos seis exemplos mostrados, cinco são em LabVIEW [38] e apenas um utiliza um sistema Android como base para apresentar os dados ao utilizador.

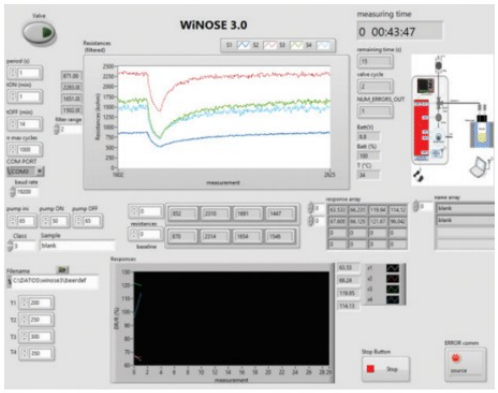


(a) [3]

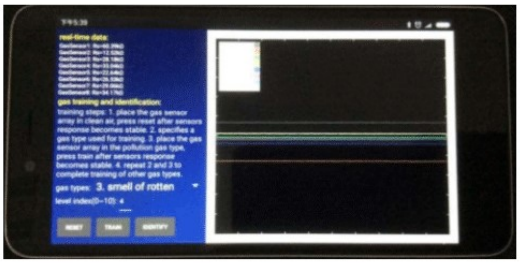
(b) [39]



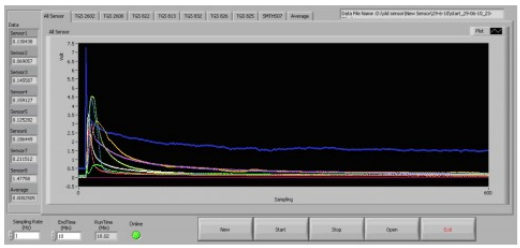
(c) [40]



(d) [2]



(e) [16]



(f) [41]

Figura 2.8 Exemplos ambientes gráficos narizes eletrónicos.

Pelas figuras 2.8a, 2.8b, 2.8c, 2.8f e 2.8d os gráficos apresentados ao utilizador têm no eixo dos x o número da amostra ou o tempo de aquisição desta, e no dos y a tensão obtida pelo sistema de aquisição. Em todos os exemplos é possível ver em tempo real os dados que estão a ser recebidos e é possível indicar o número de sensores que se deseja ler. Na figura 2.8e pode-se ver que a aplicação realiza uma calibração e consequentemente classificação das amostras e os passos para essa calibração são apresentados do lado esquerdo do ecrã.

Com estes exemplos pode-se retirar que, é importante o utilizador conseguir visualizar os dados em tempo real tanto através de um gráfico assim como através de texto. Para além disso, como se vê pelas figuras 2.8a e 2.8f o utilizador deve conseguir guardar os dados recebidos num ficheiro de dados. De notar ainda que o utilizador deverá ser capaz de inserir algumas configurações antes de iniciar a obtenção de amostras, como é o caso da frequência de amostragem, o número de sensores que compõem o nariz eletrónico e ainda um número máximo de amostras a retirar ou o tempo de execução.

2.2 Android

O android é uma pilha de software para dispositivos móveis que inclui um sistema operativo, Middleware, bibliotecas e aplicações que correm na framework,como se pode ver pela figura 2.9. Para além disso, o android é um sistema de código aberto baseado no Linux kernel e lançado pela Google [42]. Ao contrário dos sistemas operativos desenvolvidos para computadores, os sistemas operativos móveis são limitados pelo hardware, espaço de armazenamento, dissipação de energia e condições de mobilidade.

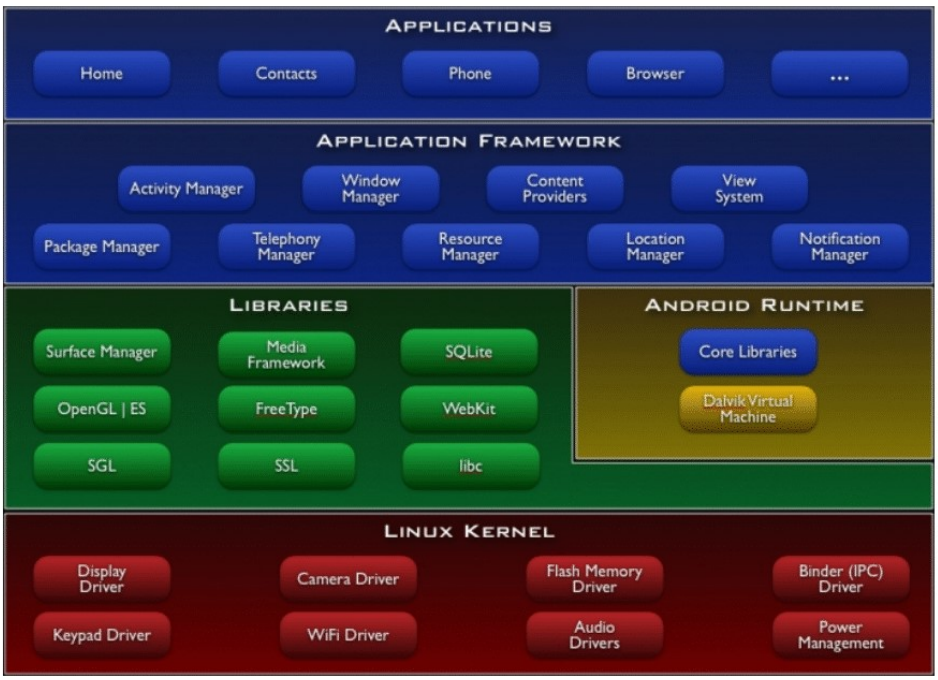


Figura 2.9 Arquitetura android [6].

Desde a sua apresentação oficial, o android captou o interesse das companhias, programadores e público em geral. Desde esse momento até agora, esta plataforma de software esta constantemente a melhorar em termos de características e suporte de hardware [42].

Como mencionado acima, o android corre no Linux kernel e, para além disso, as aplicações android são baseadas em java. Este fator implica o uso de uma máquina virtual, como se pode ver pela figura 2.10. Nesse sentido o android utiliza a sua própria máquina virtual denominada Dalvik que ao longo dos tempos, a Google tem redesenhado e otimizado para o novo hardware que os dispositivos apresentam [6]. Aplicações android são primeiramente escritas em java e compiladas em formato Dalvik executável (DEX), um código byte customizado. Cada aplicação executa o seu próprio processo, com a sua própria instância da máquina virtual Dalvik [43].

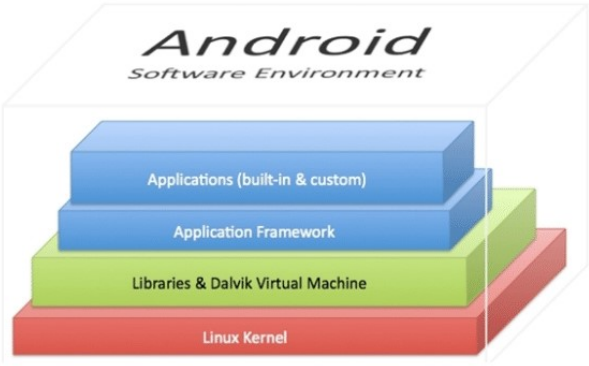


Figura 2.10 Máquina virtual Dalvik [6].

Quando uma aplicação é executada, apresenta ao utilizador uma nova atividade, que representa o ecrã que o utilizador visualiza. Para além disso, as atividades de uma aplicação trabalham em conjunto para que o utilizador tenha uma melhor experiência de utilização. Numa aplicação com várias atividades, normalmente, uma atividade é especificada como a atividade "principal", que é apresentada quando o utilizador inicia a aplicação pela primeira vez. Cada atividade pode então iniciar outra atividade para realizar diferentes ações. Sempre que uma nova atividade começa, a atividade anterior é interrompida, mas o sistema preserva a atividade numa pilha, para que possa ser chamada de novo e continuar a execução [6].

Na figura 2.11 é apresentado o ciclo de vida de uma atividade [6], como se pode ver pela imagem quando a atividade vai para background, fica na pilha até ser chamada de novo, ou até que a aplicação seja fechada. Caso uma aplicação principal do android necessite de memória, o processo fecha e será apresentado ao utilizador a atividade principal da aplicação.

Tabela 2.2 Dados coletados durante um período de 7 dias encerrado em 2017/8/8. [10]

Versão	Nome	API	Percentagem
2.3.3 - 2.3.7	Gingerbread	10	0.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.7%
4.1.x		16	2.7%
4.2.x	Jelly Bean	17	3.8%
4.3		18	1.1%
4.4	KitKat	19	16.0%
5.0	Lollipop	21	7.4%
5.1		22	21.8%
6.0	Marshmallow	23	32.3%
7.0	Nougat	24	12.3%
7.1		25	1.2%

Na tabela 2.2 é apresentado as versões de android mais utilizada atualmente. Analisando os dados da tabela, a aplicação a ser desenvolvida nesta dissertação terá compatibilidade até à versão 4.0.3 permitindo assim abranger um maior número de dispositivos.

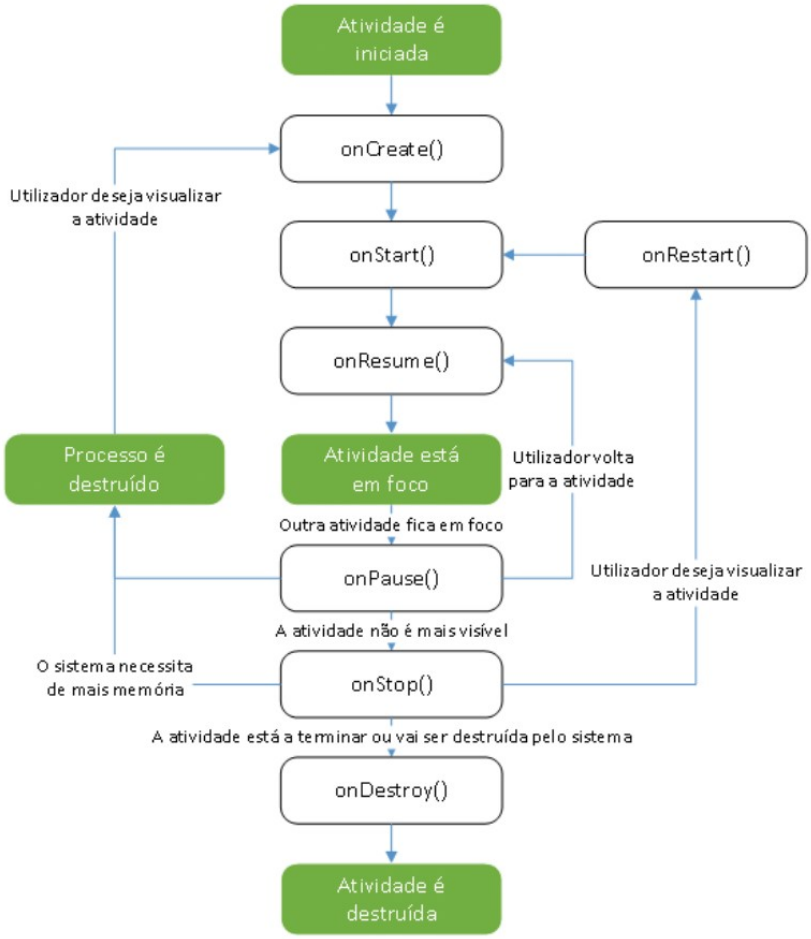


Figura 2.11 Ciclo de vida de uma atividade.

2.3 Engenharia de software

Os engenheiros de software estão habituados a utilizar o conhecimento de computadores e de informática para ajudar a resolver problemas. Frequentemente, o problema com que lidam relaciona-se com o computador ou com um sistema computacional existente, mas às vezes a dificuldade subjacente ao problema não se relaciona com computadores. Por isso, é essencial que primeiro se perceba qual é a natureza do problema. Uma componente chave do desenvolvimento de software é a comunicação entre os clientes e os programadores, para

que se possa entender o que o cliente quer e necessita antes de se iniciar a construção de um sistema [7].

O primeiro passo em qualquer processo de desenvolvimento é descobrir o que o cliente quer e documentar os requisitos. Uma vez que os requisitos são conhecidos e documentados, normalmente os analistas trabalham com os designers para gerar um modelo gráfico do sistema. De seguida os designers trabalham com os programadores para descrever o sistema de tal maneira que os programadores possam desenvolver um código que implemente o que os requisitos especificam.

Quando a equipa de desenvolvimento está satisfeita com a funcionalidade e qualidade do sistema, a atenção volta-se de novo para o cliente. A equipa de teste e o cliente trabalham juntos para verificar se o sistema desenvolvido é o que o cliente deseja. Para muitos sistemas de software, a aceitação pelo cliente não significa o fim do trabalho, uma vez que, muitas vezes os requisitos do cliente podem sofrer alterações. A este processo de desenvolvimento de um software é chamado de ciclo de vida, este descreve a vida de um produto desde a sua conceção até à sua implementação, entrega, uso e manutenção.

Sendo assim, o desenvolvimento de um software normalmente envolve as seguintes etapas:

- Análise e definição dos requisitos;
- Design do sistema;
- Implementação do sistema;
- Teste de software;
- Entrega do sistema;
- Manutenção.

Ao conjunto destas etapas e à interação entre elas é chamado modelo de desenvolvimento. Alguns modelos são representações da forma como o desenvolvimento de software deve progredir, e outros são descritivos da forma como o desenvolvimento de software é feito na realidade, mas todo o modelo de desenvolvimento de software inclui um sistema de requisitos como entrada e produtos entregues como saída.

Um dos primeiros modelos a serem propostos é o modelo em cascata, ilustrado na figura 2.12, onde as etapas progridem em forma de cascata [44]. Como a figura indica, o estágio de desenvolvimento deve ser concluído antes do próximo início. O modelo em cascata apresenta uma visão de alto nível sobre o que se passa durante o desenvolvimento, e sugere aos desenvolvedores a sequência dos eventos que eles devem esperar encontrar [7].

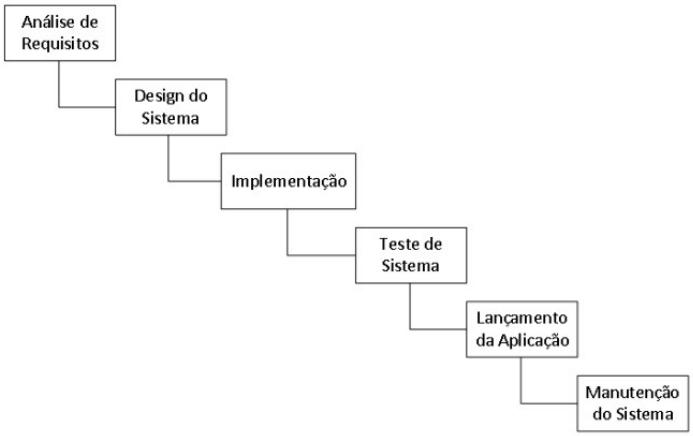


Figura 2.12 Modelo em Cascata.

Dentro deste modelo existe o modelo em cascata com prototipagem, a prototipagem é um subprocesso. Um protótipo é parcialmente um produto desenvolvido que permite aos clientes e programadores examinar alguns aspetos do sistema proposto e decidir a viabilidade do produto final. Por exemplo, os programadores são capazes de construir um sistema para implementar uma pequena porção de alguns requisitos chave para assegurar que os requisitos são consistentes, práticos e possíveis; caso contrário, as revisões são feitas na fase de requisitos e não na fase de testes, este modelo está descrito na figura 2.13.

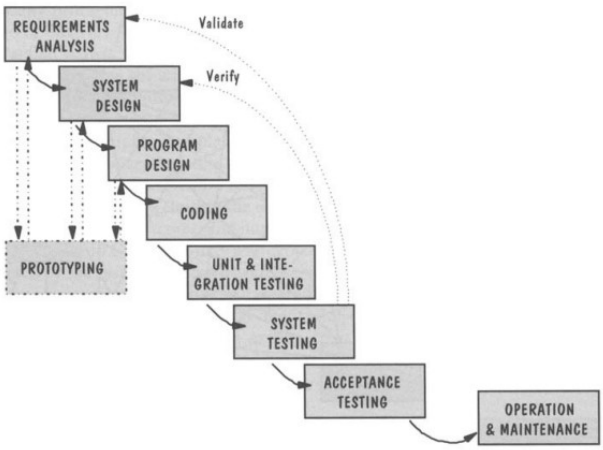


Figura 2.13 Modelo em Cascata com Prototipagem [7].

Visto que o desenvolvimento desta dissertação é realizado apenas por uma pessoa, este foi o modelo adotado pois, permite que as atividades sejam feitas de forma sequencial e apenas seguir para a próxima fase quando terminada a anterior. Para além disso utilizar o modelo com prototipagem é benéfico, pois é possível ir testando vários módulos da aplicação independentemente.

Capítulo 3

Implementação da Aplicação

Este capítulo descreve os passos utilizados para a realização do software (aplicação móvel) referida anteriormente. Como referido no capitulo 2.3 no desenvolvimento do software utiliza-se o modelo em cascata com prototipagem (*Waterfall Model with Prototyping*).

O primeiro passo no desenvolvimento de uma aplicação é o levantamento dos requisitos onde se responderá às seguintes questões:

- Quem são os interessados (*stakeholders*)?
- Quais são as suas necessidades?
- Quais serão as características do software?

Após feito o levantamento dos requisitos, passa-se para a análise dos mesmos, onde se deve responder às seguintes questões:

- Quais são os requisitos funcionais?
- Quais são os requisitos não funcionais?
- Quais são as restrições de design e de processo impostas ao projeto?

3.1 Levantamento de requisitos

Como referido no capítulo 2.3 os requisitos é uma parte importante no desenvolvimento de um software e, por isso, o levantamento dos mesmos deve ser feito com precisão e, para isso, o levantamento de requisitos é feito através de uma reunião com os interessados. No caso da aplicação desenvolvida nesta dissertação, a reunião foi realizada com o orientador da

mesma, dessa reunião surgiu a seguinte lista de requisitos sem nenhuma ordem em especifico de implementação.

Para uma leitura mais facilitada desta dissertação, a lista abaixo apresenta os requisitos base, levantados durante a reunião e os requisitos extra implementados ao longo do desenvolvimento.

1. A aplicação a desenvolver destina-se a investigadores e deve ser compatível com telemóveis e tablets. [1]

2. A aplicação será executada no sistema operativo Android e deverá ser compatível até à versão 4.0.3 (análise estatística no capítulo 2.2). [1]

3. A aplicação deverá ser compatível com telemóveis e tablets. [1]

4. O utilizador interage com o software através de uma interface gráfica intuitiva, com menus e barras de ferramentas. [2]

5. Toda a informação registada pelo utilizador na aplicação é guardada de forma persistente (na memória interna ou cartão de memória se este existir) e consistente, com um tempo de vida para além do tempo de execução da aplicação. [2]

6. A aplicação deverá permitir a conexão com dispositivos através do bluetooth. Para além de se conectar a um dispositivo já emparelhado, a aplicação deverá ser capaz de emparelhar a um novo dispositivo. [1]

7. A aplicação permite visualizar os dados recebidos através de um ambiente de texto e também através de gráficos. Os gráficos deverão possibilitar que o utilizador guarde uma imagem do mesmo. [1]

8. Para além dos dados recebidos a aplicação deve ainda apresentar o tempo a que os dados foram recebidos. [1]

9. Sempre que é feita uma nova tiragem de dados, o tempo deverá começar do zero. [1]

10. A aplicação deverá permitir ao utilizador configurar alguns aspetos da aplicação que sejam pertinentes para a mesma. [2]

11. A aplicação deverá apresentar algumas informações pertinentes ao utilizador. Deve ser apresentado um texto com Perguntas Mais Frequentes(FAQ) sobre a aplicação e um breve tutorial sobre o funcionamento da mesma. [2]

12. O utilizador deverá ter a capacidade de guardar históricos(logs) com os dados retirados e enviar os mesmos por email. A aplicação deverá também permitir carregar os históricos guardados. [2]

13. A aplicação deverá ser capaz de realizar uma calibração, onde deve receber um número de amostras definidas pelo utilizador. Com essas amostras deverá realizar uma análise dos dados e guardar essa análise num ficheiro para ser utilizado durante a receção de novos dados. [1]

14. A quando da receção de novos dados, a aplicação deverá ser capaz de realizar uma previsão da classe a que estes pertencem utilizando os dados obtidos na calibração. [1]

3.2 Análise de requisitos

3.2.1 Casos de uso

A partir dos requisitos acima apresentados, começa-se por realizar um caso de uso geral, que abrange as principais funções dos atores (utilizador). Com este diagrama, como mostra a figura 3.1, verifica-se que o utilizador poderá realizar quatro ações principais: conectar a um dispositivo, alterar as definições do software, visualizar informações da aplicação e carregar/guardar históricos. para além disso, as definições e os logs irão guardar as informações no dispositivo do utilizador para futura utilização.

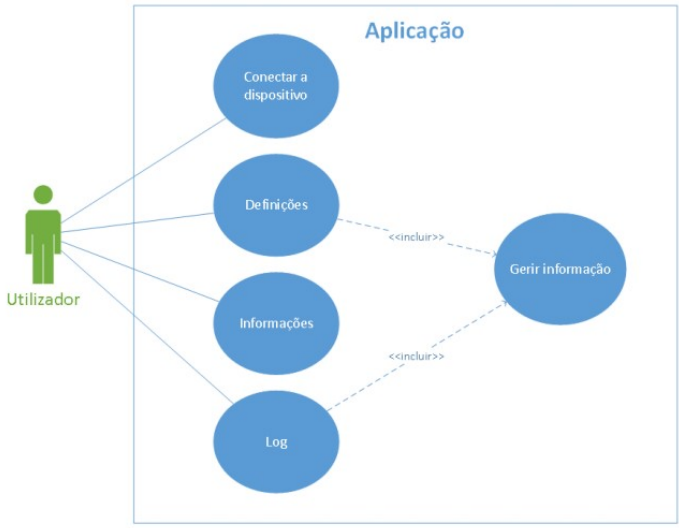


Figura 3.1 Diagrama caso de uso geral.

[1] Requisitos base.
[2] Implementação extra.

3.2.1.1 Requisitos funcionais

Neste ponto explora-se e organiza-se em forma de lista todos requisitos funcionais com prioridade de implementação.

Tabela 3.1 Lista de Requisitos Funcionais.

Referência	Descrição	Prioridade [3]
A	**Conectar a Dispositivo**	
A.1	Descobrir novos dispositivos	1
A.1.1	Emparelhar ao dispositivo	
A.2	Conectar a dispositivo já emparelhado	1
B	**Receber dados**	
B.1	Texto	2
B.1.1	Visualizar	
B.2	Gráficos	3
B.2.1	Visualizar	
B.2.2	Guardar imagem	
C	**Definições**	**5**
C.1	Visualizar	
C.2	Alterar	
D	**Informações**	**5**
D.1	Visualizar	
E	**Log**	
E.1	Carregar	4
E.1.1	Enviar por email	
E.2	Guardar	3
F	**Calibração**	
F.1	Analisar os dados	5
G	**Receber novos dados**	
G.1	Classificar amostras	5

Dependências

[3] O índice de prioridade usa uma escala de 1 a 5, onde 1 significa prioridade máxima e 5 prioridade mínima.

Em relação à tabela 3.1, temos:

- A - Não possui dependências.
- C - Não possui dependências.
- D - Não possui dependências.
- B - Depende de A.
- E - Depende de A e B.
- F - Depende de A e B.
- G - Depende de A, B e F.

Tendo em conta que C e D não têm dependentes, construiu-se um diagrama de dependências entre os casos de uso principais do software, demonstrando a funcionalidade prioritária da aplicação. Como se pode ver pela figura 3.2, para que o utilizador consiga classificar novas amostras, terá que primeiro realizar uma calibração. Só após a calibração concluída e guardada é que este pode adquirir novos dados e classificar os mesmos.

Figura 3.2 Ilustração de dependências e prioridades de funcionalidade.

3.2.1.2 Requisitos não funcionais

Os requisitos não funcionais ou requisitos de qualidade têm como objetivo descrever algumas características de qualidade que a aplicação deverá ter.

Desempenho

O desempenho da aplicação não é um fator crítico pois, o software não é destinado ao uso comercial mas, não quer dizer que este não seja um ponto a ter em consideração. Por isso mesmo, a nível dos recursos do telemóvel a aplicação deve ser o mais leve quanto possível. Para que, qualquer dispositivo que disponha de bluetooth, consiga executar a aplicação sem problemas.

Como o telefone tem como objetivo receber dados em tempo real, utiliza-se *threads* em paralelo com o *GUI* para que o utilizador possa navegar pela aplicação sem que esta fique bloqueada à espera de novos dados.

Usabilidade

De modo a que um utilizador comum com poucos conhecimentos de informática possa ser capaz de utilizar a aplicação sem necessitar de nenhum treino, deve-se ter como principal objetivo tornar a aplicação o mais intuitivo e simples possível, dando prioridade a menus de fácil acesso a todas as informações. Para além disso, a interface gráfica deve ser o mais apelativa e funcional possível. Para isso, na primeira execução da aplicação apresenta-se ao utilizador um pequeno tutorial mostrando as funcionalidades principais. Caso este tenha alguma dúvida durante o funcionamento da aplicação poderá esclarece-las através do ecrã das informações.

Segurança

Não se tratando de uma aplicação com dados importantes, o acesso aos dados não será controlado pela aplicação, pelo que não será necessário a implementação de um sistema de autenticação nem encriptação de ficheiros.

Disponibilidade

Neste ponto deve-se ter em consideração as falhas que poderão ocorrer devido ao uso indevido do utilizador e todas as falhas que poderão ocorrer após o lançamento da aplicação, para isso, todos os erros ocorridos serão enviados automaticamente para um servidor onde posteriormente se poderá detetar e corrigir o erro. Quando ocorrer alguma falha prevista pelo software, o utilizador receberá logo uma mensagem e a aplicação reverterá para o estado anterior, onde este poderá tentar de novo a ação que estava a realizar.

Manutenção

Para que a manutenção do software seja fácil, tanto na correção de erros como na implementação de novas funcionalidades, deve-se para isso realizar uma programação o mais estruturada e de fácil interpretação quanto possível, tanto a nível de código bem como a nível de comentários ao longo do código.

Este requisito é de extrema importância, pois facilita numa primeira abordagem um melhor *debug* da aplicação e, quando este estiver terminado e ao serviço do cliente, uma melhor e fácil manutenção.

3.2.1.3 Restrição de design

Esta restrição recai sobre as escolhas de plataforma, de interface de desenvolvimento, linguagem, etc.

Neste caso o *stakeholder* decidiu que a aplicação fosse desenvolvida para o sistema operativo Android, pelo que no fim será gerado automaticamente um APK que será então disponibilizado a todos os interessados, se for caso disso poderá também ser colocado online num servidor para download ou então na Google Play Store.

Escolhida a plataforma de desenvolvimento foi necessário decidir qual a linguagem a utilizar e, qual o IDE que seria utilizado para o desenvolvimento da aplicação. A escolha recaiu no Android Studio na linguagem Java, visto que atualmente este é o IDE oficial do Android. Para além disso, o site de apoio ao IDE inclui para além de explicações sobre vários aspetos da arquitetura do sistema, uma lista extensa de tutoriais de iniciação/avançados de Android.

3.2.1.4 Restrição de processo

As restrições de processo recaem sobre restrições que existam nos métodos, técnicas ou recursos a utilizar na construção do sistema. Neste caso, as únicas restrições de processo que existem recaem sobre as limitações que a linguagem Java possa ter.

3.2.2 Lista de requisitos e restrições

A tabela abaixo não apresenta nenhuma ordem em específico de restrições, foi realizada apenas tendo em conta o levantamento de requisitos.

Número	Descrição	Tipo [4]
1	A aplicação destina-se a investigadores usando o telefone pessoal	RNF
1.1	A aplicação deverá ser compatível com telemóvel e tablets	RD
2	A aplicação deve ser desenhada para ser executada em Android	RD
3	A aplicação deve ter uma Interface Gráfica intuitiva, com recurso a menus e barras de ferramentas	RD
4	Toda a informação deverá ser guardada de forma persistente e consistente, ficando guardada na memória interna do telefone ou no catão de memória	RF
5	A linguagem de programação utilizada deverá ser Java	RD
6	A aplicação deverá permitir conexão/emparelhamento a dispositivos através do bluetooth	RF

7	A aplicação permite receber e visualizar os dados através de texto e gráficos	RF
7.1	Para além dos dados recebidos, a aplicação deverá ainda apresentar o tempo em que estes foram recebidos	RF
7.2	Em cada tiragem o tempo deverá iniciar sempre do zero	RF
8	A aplicação deverá permitir a configuração de alguns aspetos da mesma	RF
9	A aplicação deverá apresentar um pequeno FAQ sobre a mesma	RF
10	A aplicação deverá permitir guardar e carregar logs com os dados recebidos	RF
10.1	Os logs guardados deverão ter a possibilidade de serem enviados para um email à escolha do utilizador	RF
11	A aplicação deverá ser capaz de analisar os dados recebidos através de algoritmos apropriados	RF
11.1	A calibração feita deverá ser guardada num ficheiro para uso futuro.	RF
11.2	Os novos dados adquiridos após a calibração deverão ser tratados de acordo com a mesma.	RF

Tabela 3.2 Lista de Requisitos e Restrições.

3.3 Especificação UML

Após realizada a análise de requisitos, procede-se então à especificação dos mesmos onde serão apresentados diagramas UML para analisar mais pormenorizadamente os requisitos referidos anteriormente. Nesta secção começa-se por apresentar diagramas de caso de uso para subdividir o diagrama apresentado na figura 3.1.

3.3.1 Subsistema "Conectar a dispositivo"

O primeiro subsistema que se aborda é referente ao menu "Conectar a dispositivo" e como se pode ver pela figura 3.3, neste diagrama existem duas opções: Descobrir novos dispositivos e Conectar a um dispositivo já emparelhado.

[4]RF: Requisito Funcional RNF: Requisito Não Funcional RD: Restrição de Design

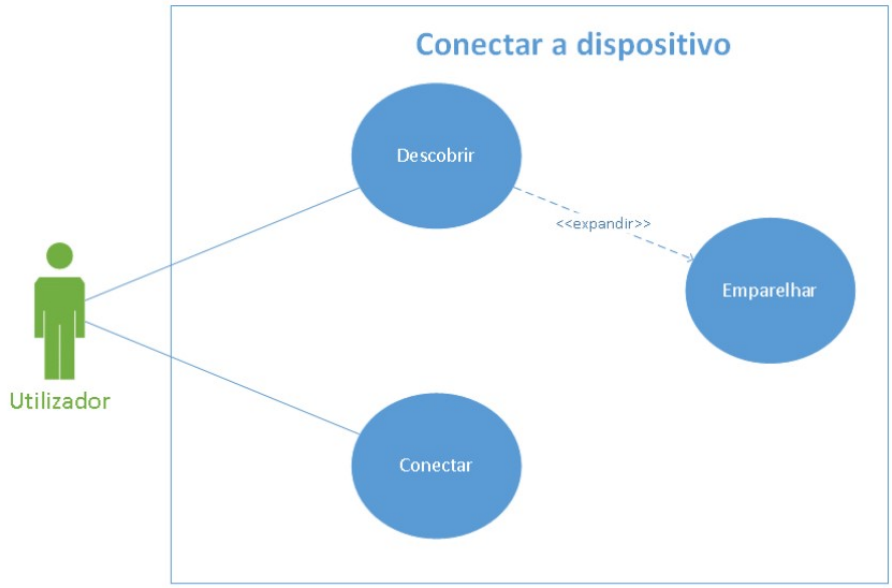


Figura 3.3 Diagrama caso de uso referência **A** da tabela 3.1.

Se um novo dispositivo for descoberto pela aplicação, o utilizador terá ainda a opção de emparelhar ao mesmo e a conexão a este será feita sem intervenção do utilizador.

3.3.2 Subsistema "Receber dados"

O próximo subsistema é referente à receção de dados, o utilizador só terá acesso a esta opção após se conectar a um dispositivo, mas como se trata de um requisito do sistema decidimos abordar o mesmo nesta secção.

No momento em que uma conexão seja feita corretamente, automaticamente a aplicação deverá mostrar ao utilizar uma caixa de texto onde serão visualizados os dados. Para além dos dados na forma de texto, o utilizador terá ainda a possibilidade, através do menu auxiliar, de visualizar os dados recebidos através de gráficos, como se pode visualizar na figura 3.4.

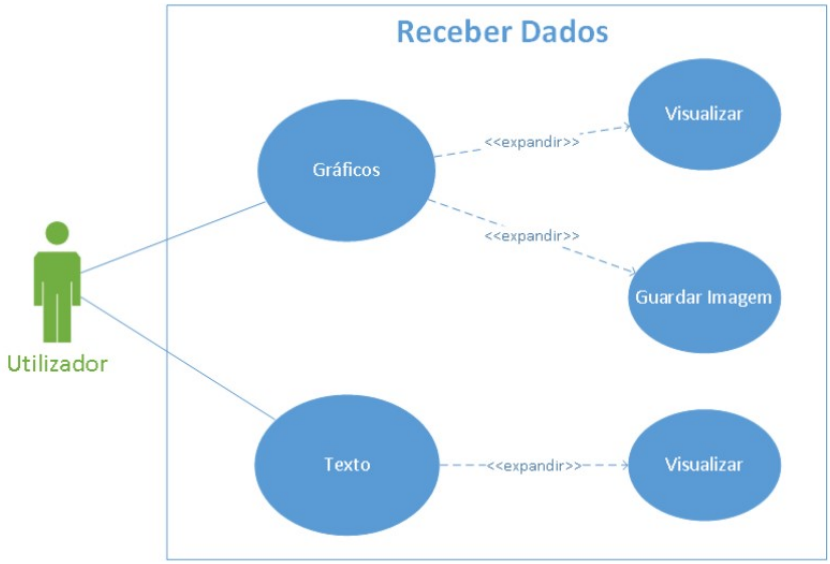


Figura 3.4 Diagrama caso de uso referência **B** da tabela 3.1.

3.3.3 Subsistema "Definições"

O terceiro subsistema é referente ao menu "Definições". Neste, o utilizador pode visualizar algumas opções, estas vêm por predefinição com as definições mais pertinentes para o uso da aplicação, mas em qualquer altura o utilizador poderá alterar as mesmas, como se visualiza na figura 3.5. Nas figuras 3.6a e 3.6b mostra-se mais em pormenor as definições implementadas.

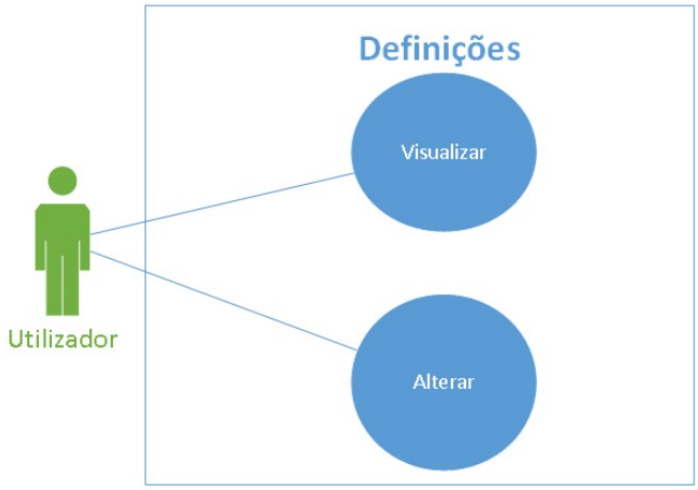
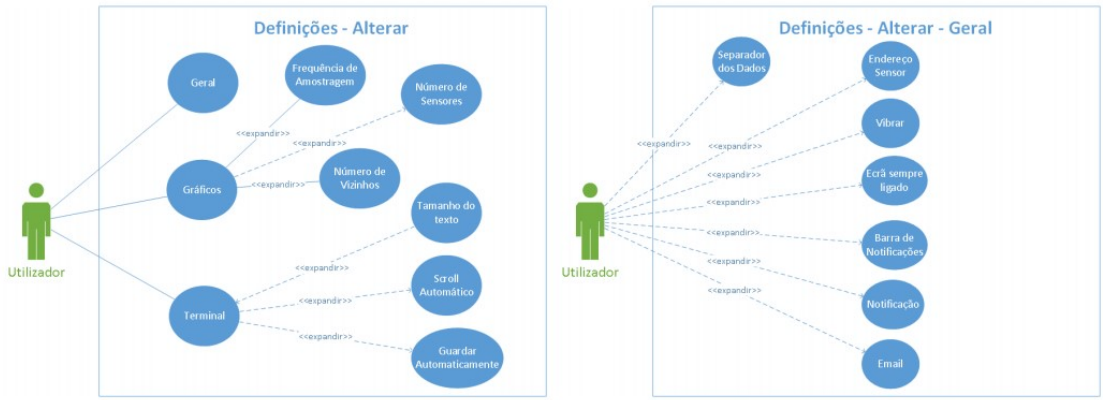


Figura 3.5 Diagrama caso de uso referência **C** da tabela 3.1.

Como se viu na figura 3.5 o utilizador pode alterar as definições apresentadas. Pode-se então ver pela figura 3.6a que existem três categorias: Geral, Gráficos e Terminal. Dentro de cada uma dessas categorias existem várias opções que podem ser alteradas pelo utilizador, como mostram as figuras 3.6a e 3.6b.

Algumas opções ao serem selecionadas mostram uma pequena demonstração da sua funcionalidade, como é o caso da opção "Vibrar" que ao ser selecionada faz com que o telefone vibre durante breves momentos e ainda a opção "Notificação" que apresenta ao utilizador por breves instantes (5 segundos) uma notificação de teste.

No caso da opção "Barra de Notificações", ao estar selecionada impossibilita ao utilizador de escolher mutuamente a opção "Notificação". Se o utilizador tiver a barra de notificações oculta, não irá conseguir ver a notificação no seu telefone. Assim consegue-se fazer com que a aplicação não consuma desnecessariamente recursos do telefone.



(a) Definições - Alterar. (b) Definições - Alterar - Geral.

Figura 3.6 Diagrama caso de uso referência **C.2** da tabela 3.1.

3.3.4 Subsistema "Informações"

O quarto subsistema é referente ao menu das informações. Neste menu, o utilizador poderá visualizar algumas informações sobre a aplicação bem como um pequeno *FAQ* sobre a mesma. Na figura 3.7 é apresentado o diagrama referente a este subsistema.

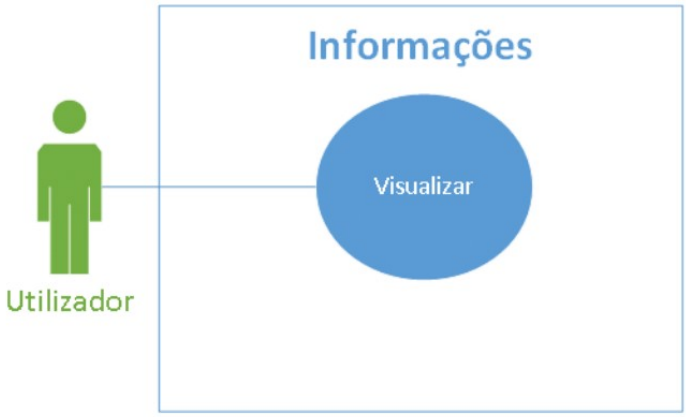


Figura 3.7 Diagrama caso de uso referência **D** da tabela 3.1.

3.3.5 Subsistema "Logs"

O próximo subsistema é referente aos logs e neste o utilizador poderá guardar os dados recentemente adquiridos pela aplicação. Caso o utilizador já tenha algum ficheiro guardado no seu telefone poderá então aceder ao menu de carregamento e visualizar todos os ficheiros guardados. Caso assim o deseje pode ainda enviar o log via email. De notar que, para isso, deve definir um email no menu das Definições.

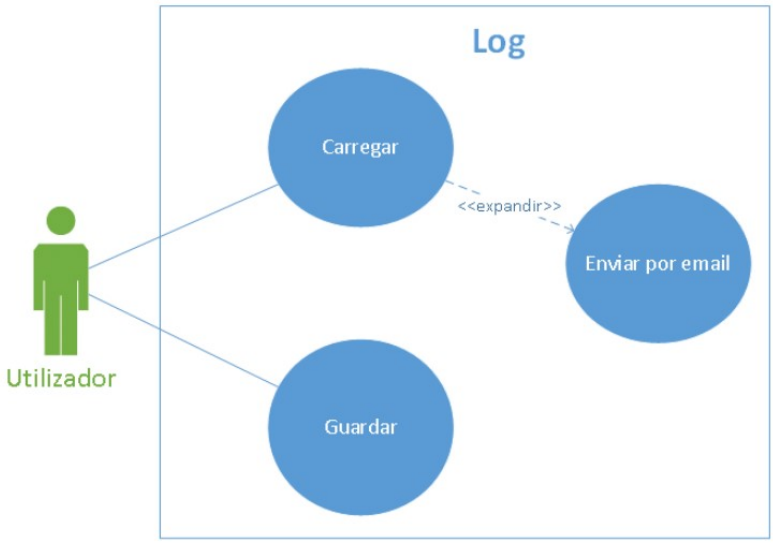


Figura 3.8 Diagrama caso de uso referência **E** da tabela 3.1.

3.3.6 Subsistema "Calibração"

O penúltimo subsistema refere-se à calibração, este pode ser descrito pela figura 3.9. Neste, o utilizador deverá introduzir algumas definições iniciais, como o número de experiências que deseja realizar bem como o número de amostras por experiência, por exemplo: 3 experiências, 30 amostras. Após desejar começar a calibração, será apresentado o ecrã de conexão a um dispositivo, como referido no capítulo 3.3.1, assim que a conexão seja feita com sucesso, é mostrado um ecrã onde o utilizador deve inserir o nome da experiência e a sua concentração (em ppm) por exemplo: Vinagre, 1000. De seguida a aplicação inicia a receção dos dados, como descrito no capítulo 3.3.2. No exemplo dado, seriam adquiridas 30 amostras, no fim de cada experiência é apresentado de novo o ecrã para o utilizador inserir os dados da próxima experiência. A aplicação repetirá este processo até concluir o processo de obtenção de dados, que no exemplo dado seriam três experiências, logo, pediria ao utilizador que inserisse os dados de cada experiência três vezes e retirava 30 amostras para cada, o que resultaria num vetor de amostras de tamanho 90 * 4 no caso de um nariz eletrónico constituído por 4 sensor.

Após concluída a obtenção de dados, a aplicação irá aplicar o algoritmo para obter a matriz PCA, como descrito no capítulo 2.1. Desta guarda-se o PC1 e PC2 (duas componentes mais significativas dos dados obtidos) e calcula-se o centro de cada grupo de amostras, esse é obtido calculando a média de valores correspondentes a cada um dos grupos.

Tendo já a matriz PCA, a aplicação apresenta ao utilizador um novo gráfico Este é denominado gráfico de dispersão e mostra os pares de valores (PC1, PC2) possibilitando assim visualizar o clustering realizado.

Quando o utilizador desejar pode fechar esta janela e os dados serão guardados num ficheiro para serem utilizados na obtenção de novos dados e, assim realizar a classificação dos mesmos, utilizando o método kNN referido no capítulo 2.1.

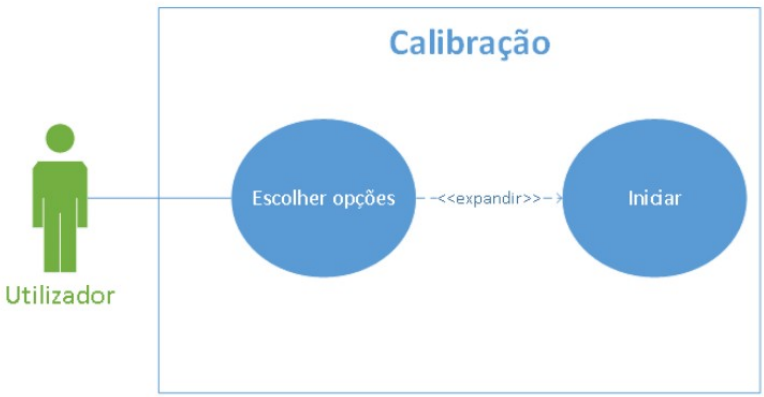


Figura 3.9 Diagrama caso de uso referência **F** da tabela 3.1.

3.3.7 Subsistema "Receber novas amostras"

O último subsistema abordado refere-se à receção de novas amostras. Pela figura 3.10, pode-se ver que este subsistema tem como objetivo classificar as amostras recebidas, para isso a aplicação deverá apresentar o ecrã de conexão a um dispositivo, como referido no capítulo 3.3.1. Após feita uma conexão válida, a aplicação carrega o ficheiro de calibração obtido no subsistema anterior. De seguida, procede-se à aquisição de dados como apresentado no capítulo 3.3.2. Cada vetor de amostras adquirido é então classificado utilizando o método kNN referido no capítulo 2.1.

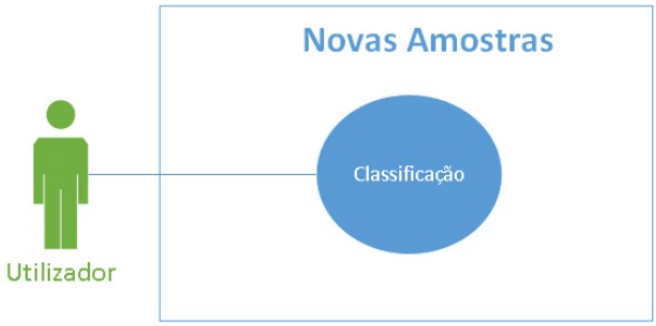


Figura 3.10 Diagrama caso de uso referência **G** da tabela 3.1.

3.4 Desenho da Arquitetura em UML

Após feita uma análise e especificação dos requisitos em UML, esta secção terá como objetivo projetar em UML uma solução técnica para o software tendo em conta o referido nas secções anteriores. Esta secção divide-se em duas fases: *design* conceptual e *design* técnico.

3.4.1 Desenho conceptual

Nesta secção será feita uma apresentação dos *mokups* feitos antes da elaboração do software, que têm como objetivo a criação da interface do utilizador em papel ou utilizando aplicações específicas, para que os *stakeholders* possam ter uma ideia de como será o software. Assim a equipa de desenvolvimento terá já uma ideia de como os interessados visualizam o produto final [45].

Para além dos *mokups*, será ainda feita uma abordagem à visão geral da arquitetura da aplicação, onde se falará dos componentes que irão constituir a aplicação e a dependência dos pacotes da mesma.

3.4.1.1 Interface com o utilizador

A figura B.1 apresentada no anexo B mostra os *mokups* construídos a partir dos requisitos apresentados anteriormente. Estes foram desenhados utilizando o software Balsamiq Mockups 3. Visto que a aplicação terá que carregar definições definidas pelo utilizador decidiu-se iniciar a aplicação com uma tela de abertura como mostra a figura B.1a.

Após a aplicação carregar completamente será apresentado ao utilizador a figura B.1b, onde será apresentado um pequeno texto de como a aplicação funciona.

Como referido nos requisitos, o utilizador deverá interagir com a aplicação maioritariamente através de menus, pelo que se implementa um menu de navegação, como existem nas aplicação da Google (ex. Google Play Store), como se pode ver pela figura B.1c, os campos do menu serão implementados de acordo com a tabela 3.1.

De seguida, apresenta-se a interação do utilizador com as várias categorias do menu. A imagem B.1d é referente à tabela 3.1 referência **A**, onde o utilizador terá uma caixa *pop-up* com os dispositivos emparelhados ao telefone em primeiro plano, e logo abaixo a opção de descobrir novos dispositivos. A imagem B.1e refere-se à tabela 3.1 referência **C** onde são apresentadas as definições, como se mostrou nas figuras 3.6a e 3.6b. Como já referido as informações serão constituídas por um texto explicativo da aplicação, como se pode ver pela imagem B.1f, esta é referente à tabela 3.1 referência **D**.

Quanto aos logs, estes podem ser guardados através do menu principal, para além disso, o utilizador poderá carregar um log previamente guardado, para isso será apresentado uma caixa *pop-up* com os logs guardados no telemóvel, como se pode ver pela figura B.1g. As últimas duas figuras, B.1h e B.1i, são referentes à tabela 3.1 referência **B** e como já referido na secção 3.3 o utilizador terá uma caixa de texto poderá visualizar os dados recebidos. Para além disso, pode-se também analisar os dados através dos gráficos implementados, como se pode ver pela figura B.1i.

3.4.1.2 Visão geral da arquitetura

Arquitetura Alto Nível

Outro aspeto importante no *design* conceptual são os componentes principais que constituirão a aplicação. O sistema será composto por 3 componentes principais:

- Uma interface gráfica (GUI) com a qual o utilizador interage.
- Um conjunto de bibliotecas e funções que são executadas tanto em paralelo com GUI, no caso de *threads*, bem como quando o utilizador interage com o sistema através do GUI.

- Os ficheiros serão onde se guardará os vários logs guardados pelo utilizador e, também onde se armazena as preferências da aplicação.

Estes componentes interagem entre si, como é apresentado no diagrama abaixo.

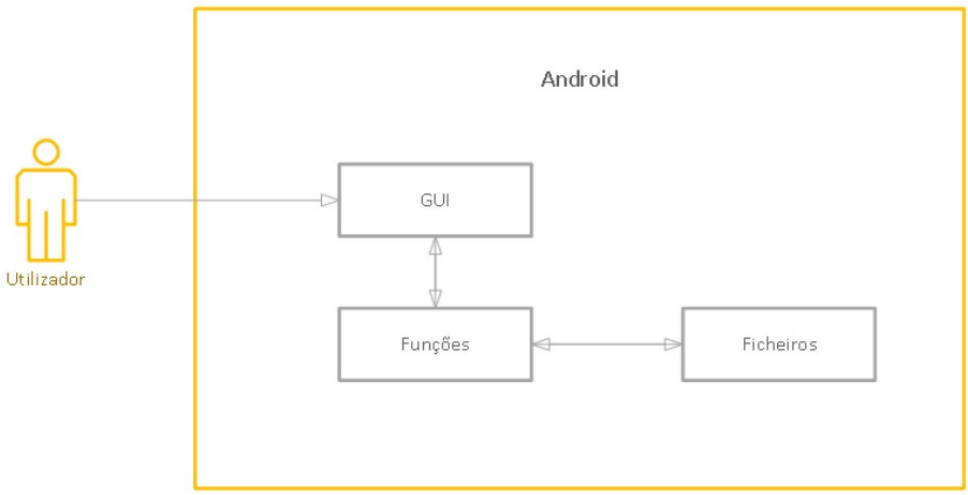


Figura 3.11 Componentes Principais e sua Interação.

Dependência entre pacotes

Como se pode ver pela figura 3.12, a aplicação divide-se em três grandes pacotes: java, res e manifests. No pacote "java", é onde se situara o código referente a cada atividade e terá como objetivo executar ações referentes à interação do utilizador com o GUI, é também neste pacote que se interage com os ficheiros (logs e preferências do utilizador) criados ao longo da execução.

No pacote "res", é onde estão os ficheiros referentes ao layout do GUI, é neste onde se realiza o desenho do ambiente gráfico e dos menus a que o utilizador terá acesso. Como se pode ver na figura 3.12 existe um pacote denominado "values", neste cria-se um ficheiro strings.xml onde são guardadas todas as strings usadas ao longo do programa, o que fará com que a aplicação possa ser implementada em várias línguas, neste caso a aplicação é implementada em Inglês como predefinição e também em Português.

Por fim o pacote "manifests", tem apenas um ficheiro denominado AndroidManifest.xml, que contém a informação do comportamento das várias atividades da aplicação, bem como, qual será a atividade que iniciará a aplicação.

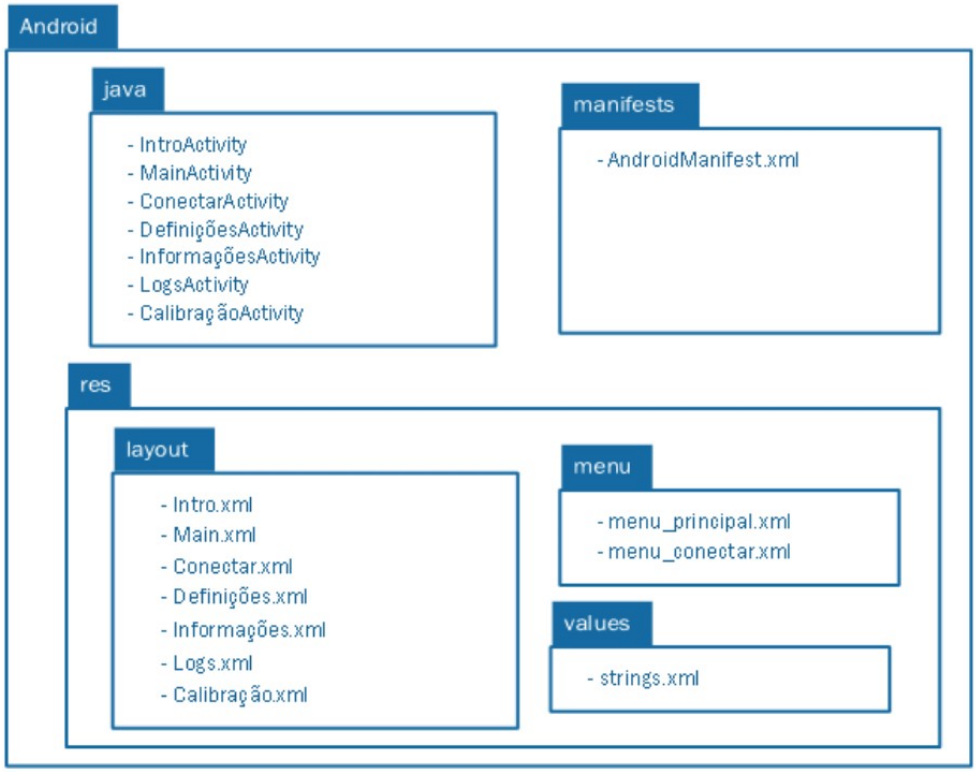


Figura 3.12 Diagrama de Pacotes.

3.4.1.3 Design gráfico

A maior preocupação quanto ao design gráfico é que a aplicação seja o mais funcional e intuitivo possível para o utilizador. Nunca uma interface gráfica deve ser de tal forma complexa que torne a aplicação mais difícil de utilizar, mesmo que isso a torne visualmente mais apelativa.

Para que o utilizador consiga navegar pela aplicação e tirar partido das principais funções da aplicação sem ser necessário a leitura do manual do utilizador, a aplicação terá uma interface simples no que diz respeito à apresentação de dados. Como todos os telefones que corram o sistema operativo android têm instalado o Google Play Store, utiliza-se muitas das funcionalidades implementadas nessa aplicação para que o utilizador se sinta familiarizado com o ambiente gráfico.

Tendo em atenção isso, implementa-se um menu lateral igual ao Google Play Store, onde os menus serão seguidos de imagens ilustrativas da função para uma maior facilidade de compreensão. Para além do menu apresenta-se a maioria dos dados no ecrã principal, com exceção de alguns *pop-ups* que poderão ser implementados.

Caso seja implementada alguma atividade que sobreponha o ecrã principal, na barra de ações será implementado uma seta para que o utilizador possa recuar para o ecrã anterior.

No que toca aos gráficos, como serão implementados mais que um gráfico, estes serão apresentados como na aplicação da Google Play Store quando se pretende trocar de categoria. Para isso, o utilizador tem apenas que deslizar o dedo ao longo do ecrã no sentido que pretende trocar de ecrã.

3.4.2 Desenho Técnico

Após feita uma abordagem conceptual do software, nesta secção fala-se concretamente do software. Primeiramente são abordados os diagrama de classes da aplicação, de seguida fala-se do comportamento do software onde se descreve os dois casos mais complexos, após isso aborda-se os diagramas de atividades referentes à tabela 3.1. De seguida apresenta-se os diagramas de componentes e de implementação referentes à aplicação, onde se dá uma perspetiva de todas as bibliotecas utilizadas no projeto e como o telefone irá interagir com a aplicação, bluetooth e memória interna.

Por fim, é apresentado o diagrama de ficheiros onde são apresentados os parâmetros guardados em cada ficheiro.

3.4.2.1 Estrutura do sistema

Como se pode ver pela figura A.1 no anexo A, a aplicação tem como classe principal a MainActivity que é onde a maioria das funções são implementadas. Wssa classe implementada 5 *AssyncTask* responsáveis por ligar o bluetooth do telefone, conectar e desconectar a dispositivos, receber dados dos dispositivos conectados ao aparelho e por fim ler os logs guardados na memória do telefone.

Como os dados são recebidos na classe MainActivity, esta comunica com o TerminalFragment para que este possa atualizar as informações recebidas e, assim alterar os valores no terminal para que o utilizador os consiga visualizar em tempo real.

Da mesma forma que a classe comunica com o TerminalFragment, interage também com os vários gráficos implementados, neste caso Chart1Fragment e Chart2Fragment. Para que sejam apresentados os gráficos ao utilizador é necessário recorrer à biblioteca MPAndroidChart [46], que implementa um vasto leque de gráficos para android.

Quanto ao MainActivity.pagerAdapter, este corresponde à função responsável por implementar os fragmentos dos gráficos e, permitir que o utilizador consiga interagir com estes e alterar qual deseja visualizar.

Por fim o InitialFragment é responsável por mostrar ao utilizador a tela inicial da aplicação, onde são apresentadas algumas informações sobre a mesma.

De seguida tem-se a figura 3.13, onde estão presentes mais algumas atividades, como é o caso da LoadActivity, esta tem como finalidade apresentar ao utilizador todos os ficheiros guardados no telefone, caso o utilizador deseje pode ainda carregar o ficheiro ou enviar o mesmo por email. No caso de o utilizador pretender carregar o ficheiro, a informação do ficheiro é passada para a MainActivity e aí a AssyncTask loadingFile tratará de carregar o ficheiro e apresentar os dados ao utilizador. Caso tenha selecionado enviar o log por email, é apresentado uma opção onde o utilizador pode selecionar qual o método de partilha que deseja, dependendo das aplicações que estejam instaladas no telefone, os métodos de partilha podem variar.

A InformationsActivity apresenta ao utilizador várias informações sobre a aplicação, como por exemplo um pequeno FAQ sobre o funcionamento da mesma.

De seguida tem-se a SettingsActivity, esta tem a responsabilidade de apresentar todas as definições implementadas no sistema e, deve após o utilizador sair das mesmas guardar em memória todas as alterações feitas.

Quanto a SplahsScreenActivity, esta mostra ao utilizador um breve carregamento da aplicação, para que a aplicação possa carregar todas as informações e definições sem que o utilizador note.

Por fim, a MyApplication é onde se inicia a biblioteca ACRA (Application Crash Reports for Android) [47], esta biblioteca permite que caso o utilizador tenha algum erro/falha na aplicação durante a sua utilização, caso exista ligação à internet esse erro será enviado automaticamente para um servidor previamente configurado. Com esta implementação é possível atingir um dos objetivos referidos na secção 3.2.1.2 no parágrafo Manutenção.

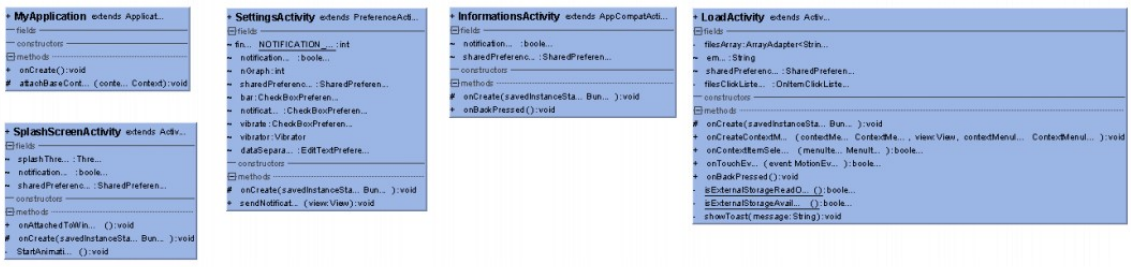


Figura 3.13 Diagrama de Classes - Atividades

Na figura 3.14, temos a IntroActivity, esta é responsável pela implementação da biblioteca AppIntro [48], o que esta biblioteca faz é apresentar ao utilizador, na primeira execução da aplicação, um pequeno tutorial de como a aplicação funciona, bem como pedir as permissões necessárias ao bom funcionamento da mesma.

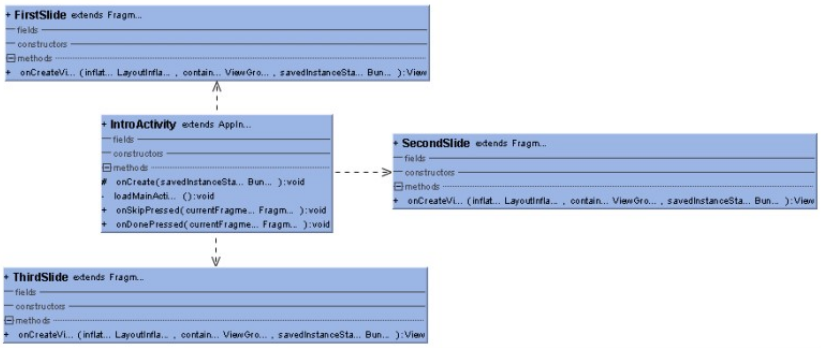


Figura 3.14 Diagrama de Classes - IntroActivity

Por fim, na figura 3.15, tem-se a DeviceListActivity que, como mostra a figura, tem uma AssyncTask denominada searchingTask. Esta é a classe responsável por apresentar ao utilizador os dispositivos *bluetooth* já emparelhados ao telefone e ainda permite ao utilizador procurar por novos dispositivos que estejam perto de si.

Quanto aos dispositivos já emparelhados, o utilizador tem duas opções, desemparelhar caso deseje ou conectar ao dispositivo, caso o dispositivo não se encontre perto do utilizador ou esteja indisponível, a aplicação apresenta uma mensagem de erro.

No caso de novos dispositivos, o utilizador tem apenas a opção de se conectar a estes e, automaticamente uma nova conexão é feita.

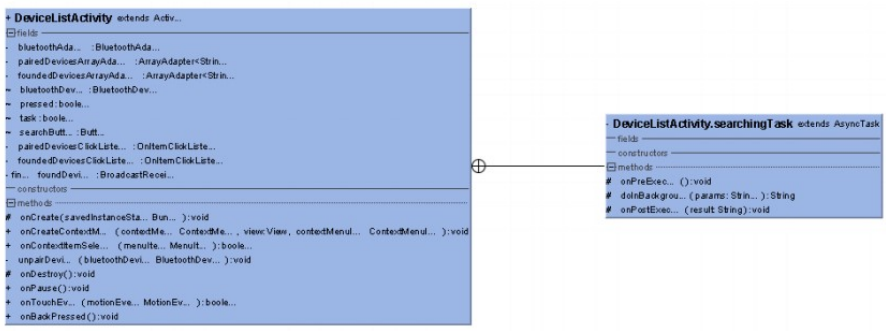


Figura 3.15 Diagrama de Classes - DeviceListActivity

3.4.2.2 Arquitetura de comportamento

Neste ponto, apresenta-se os diagramas de sequência referentes ao caso mais complexo da aplicação. Este descreve o procedimento que o utilizador deve realizar se pretender carregar um log na primeira utilização ou numa situação em que ainda não tenha guardado nenhum log. O caso descrito pode-se dividir em duas partes, a primeira, descrita na figura 3.16 refere-se a um utilizador que utilize a aplicação pela primeira vez ou que não tenha

ainda nenhum dispositivo emparelhado. Para isso, quando o utilizador desejar conetar a um dispositivo deverá primeiro ligar o seu bluetooth. Após este se ter ligado corretamente, este deverá procurar por dispositivos e esperar que o dispositivo desejado apareça.

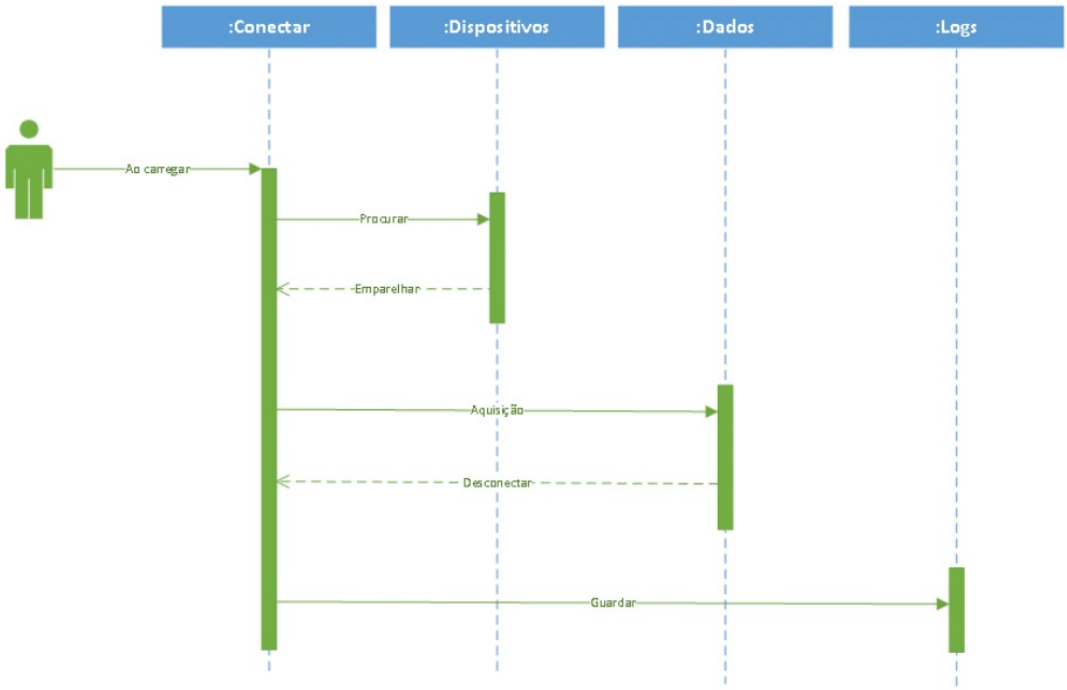


Figura 3.16 Diagrama de sequência - Receber Dados.

Quando o dispositivo aparecer na lista de dispositivos descobertos, o utilizador deve então proceder ao emparelhamento com o dispositivo. Automaticamente a aplicação irá iniciar a conexão com o dispositivo e assim inicia a receção de dados. Quando o utilizador entender que não pretende receber mais dados, deve então desconectar-se do dispositivo e guardar os dados em memória para que possam ser analisados mais tarde.

Após o utilizador ter guardado um ficheiro no seu telefone, pode então avançar para o carregamento do mesmo e assim analisar os dados. Para isso, deve seguir a figura 3.17.

O utilizador deve então no menu lateral carregar em Lista de logs(na aplicação aparece como "Carregar/Load") e é-lhe apresentada uma lista com todos os ficheiros guardados na memória do seu telefone. Após escolher qual o ficheiro que deseja carregar, os dados serão carregados. Quando já não pretender analisar os dados, pode então fechar o ficheiro e voltar ao ecrã inicial.

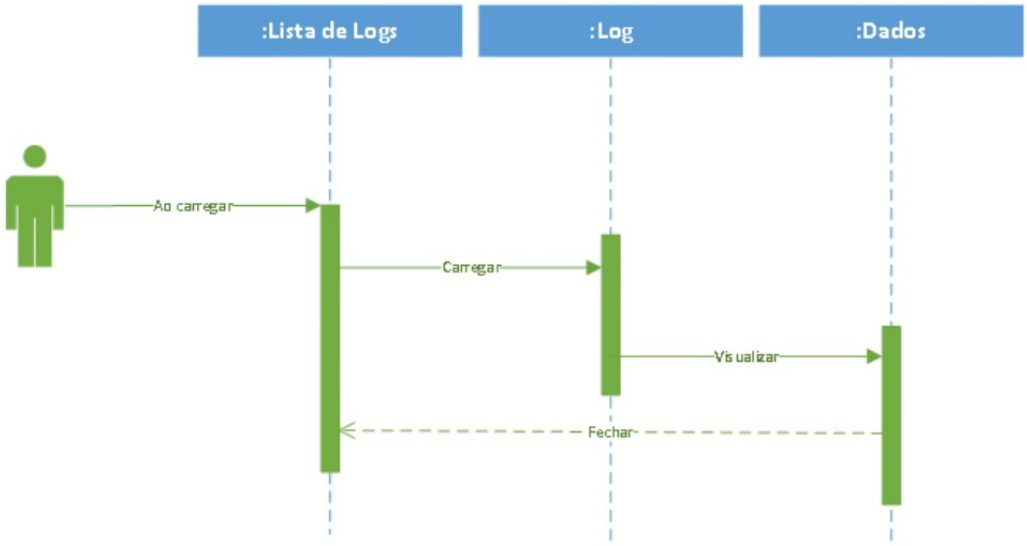


Figura 3.17 Diagrama de sequência - Logs.

3.4.2.3 Fluxo de atividades do software

Neste ponto, apresenta-se os diagramas de atividades referentes ao software desenvolvido. Com estes diagramas é possível ver o fluxo de acontecimentos dependentes da decisão do utilizador. Serão expostos sete casos particulares Conectar a Dispositivo, Definições, Informações, Logs, Receber Dados, Calibração e Classificação de novas amostras.

Antes de se falar individualmente em cada um dos casos, primeiro é necessário explicar os dois modos de funcionamento da aplicação. Como se pode ver pela figura 3.18 quando a aplicação inicia o utilizador depara-se com duas opções. Se a aplicação já tenha realizado alguma calibração previamente, o utilizador poderá iniciar o modo de obtenção e classificação de novas amostras. Caso contrário, deverá iniciar o modo de calibração onde deverá inserir as definições que deseja e aguardar o fim da aquisição.

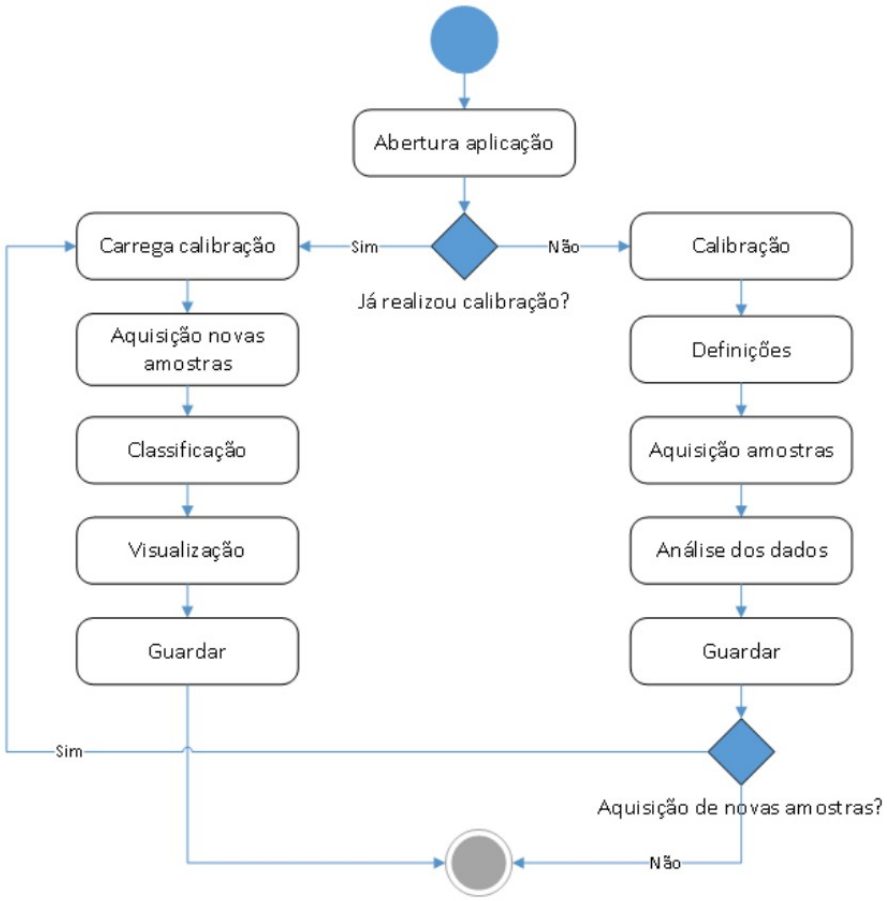


Figura 3.18 Fluxograma referente aos dois modos de funcionamento da aplicação.

O primeiro diagrama, figura 3.19, é referente à atividade de conectar a um dispositivo. Primeiro, o utilizador depara-se com uma tela de abertura, após a aplicação carregar será apresentado o ecrã inicial, após isso, o utilizador deverá abrir o menu lateral e selecionar a opção Conectar, aqui pressupõem-se que o utilizador ativa o bluetooth. Após o utilizador selecionar a opção Conectar, será apresentado o ecrã de conectar a dispositivos, onde o utilizador terá a opção de procurar novos dispositivos, conectar/desemparelhar a um dispositivo já emparelhado.

Se o utilizador desejar emparelhar a um novo dispositivo a aplicação irá proceder ao emparelhamento e conclui a ação conectando-se a esse dispositivo; se o utilizador desejar conectar-se com um dispositivo já emparelhado a aplicação irá tentar realizar essa conexão.

Caso o utilizador não selecione nenhuma das opções referidas acima ou tenha conectado a um dispositivo ou tenha selecionado a opção de desemparelhar o dispositivo, a aplicação irá atualizar o GUI e voltará ao ecrã principal.

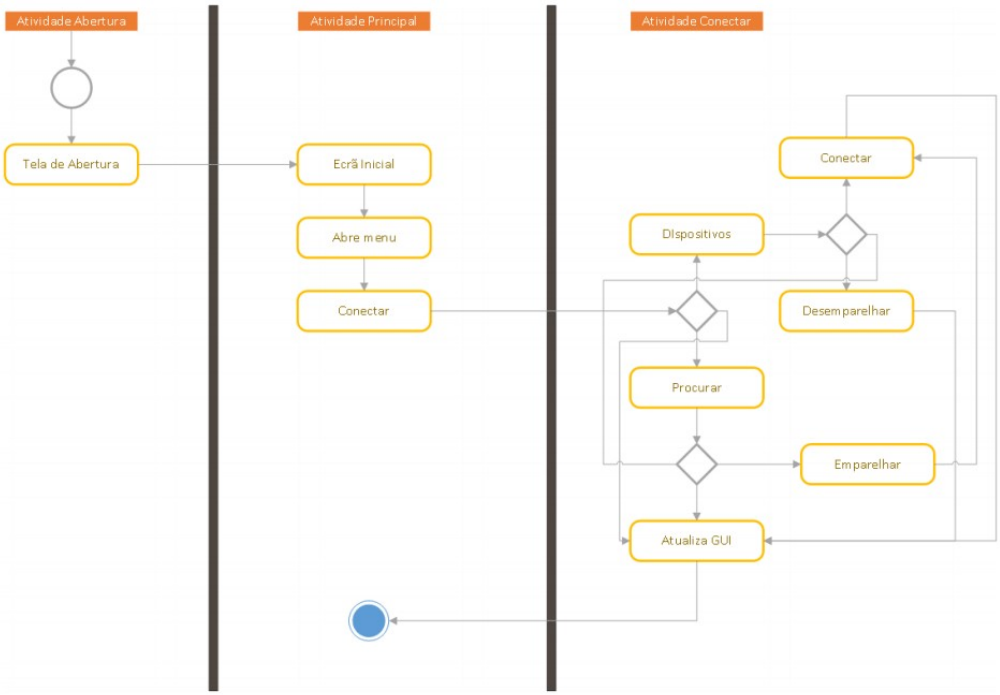


Figura 3.19 Diagrama de Atividades - Conectar a Dispositivo.

Como mostra a figura 3.20, esta inicia da mesma forma do diagrama anterior. Quando o utilizador abrir o menu lateral, terá que selecionar a opção Definições. Após isso, o ecrã das definições será aberto e aí o utilizador poderá visualizar e alterar as definições implementadas na aplicação. Quando desejar sair, a aplicação irá atualizar o GUI e mudará para o ecrã principal.

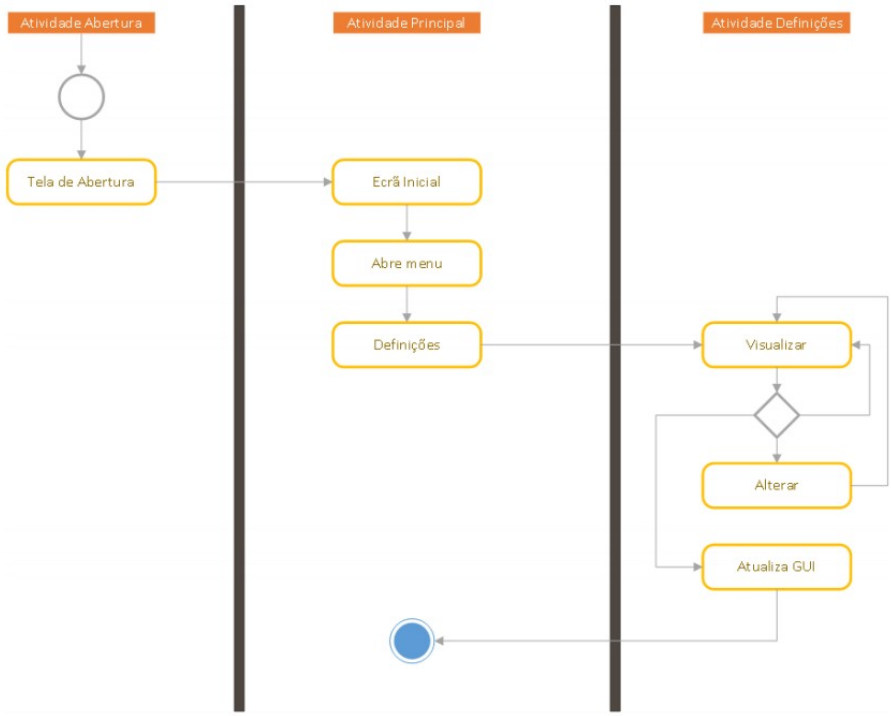


Figura 3.20 Diagrama de Atividades - Definições.

Quanto às informações, esta é muito semelhante ao diagrama anterior mas, nesta, o utilizador apenas terá a opção de visualizar o texto apresentado, como se pode ver pela figura 3.21.

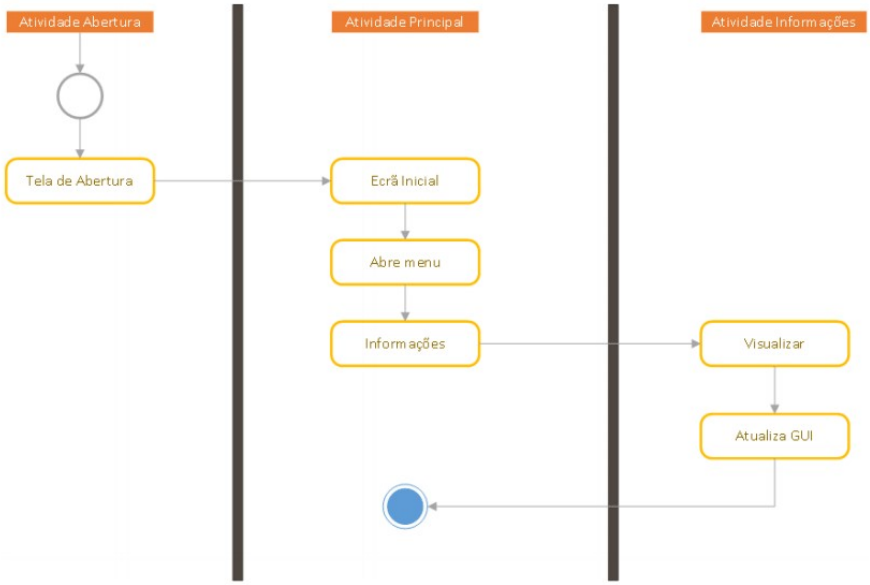


Figura 3.21 Diagrama de Atividades - Informações.

De seguida, apresenta-se a figura 3.22 referente à atividade de receber dados. Para puder receber dados, o utilizador deverá primeiro conectar a um dispositivo, como mostrado na figura 3.4.

Após o utilizador se conectar corretamente a um dispositivo, no menu lateral este terá a opção de visualizar os dados através de um terminal ou através de gráficos.

No caso do terminal, o utilizador poderá visualizar os dados recebidos através de uma caixa de texto. Relativamente aos gráficos, este terá a opção de visualizar os gráficos e se pretender poderá ainda escolher outro. Em cada um o utilizador tem a opção de guardar uma imagem do gráfico apresentado.

Quando o utilizador não pretender receber mais dados, pode desconectar-se do dispositivo e após isso poderá ainda guardar os dados num ficheiro.

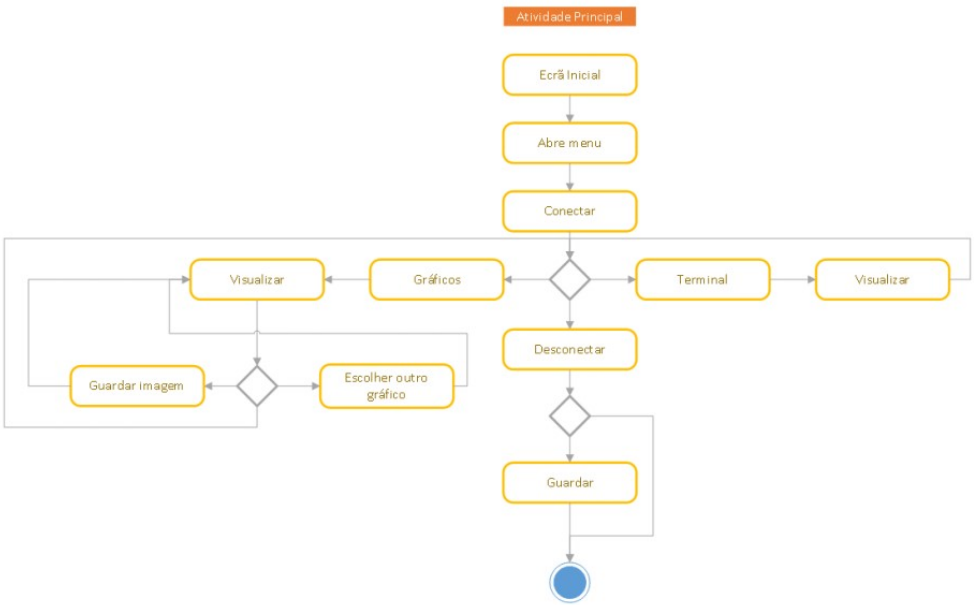


Figura 3.22 Diagrama de Atividades - Receber Dados.

Na figura 3.23, apresenta-se o diagrama de atividades dos logs, neste pressupõe-se que o utilizador já definiu o email para onde deseja enviar os logs. Após o utilizador selecionar a opção carregar, será apresentado o ecrã referente a essa opção, onde este apresentará ao utilizador todos os logs presentes no telefone. Com esses logs, o utilizador poderá carregar o ficheiro para a aplicação ou enviar o mesmo por email. Após ser selecionada a opção de carregar um ficheiro ou o utilizador pretender sair sem selecionar nenhum ficheiro, a aplicação irá atualizar o GUI e por fim apresentar o ecrã principal. No ecrã principal, o utilizador poderá visualizar os dados como explicado na figura 3.22.

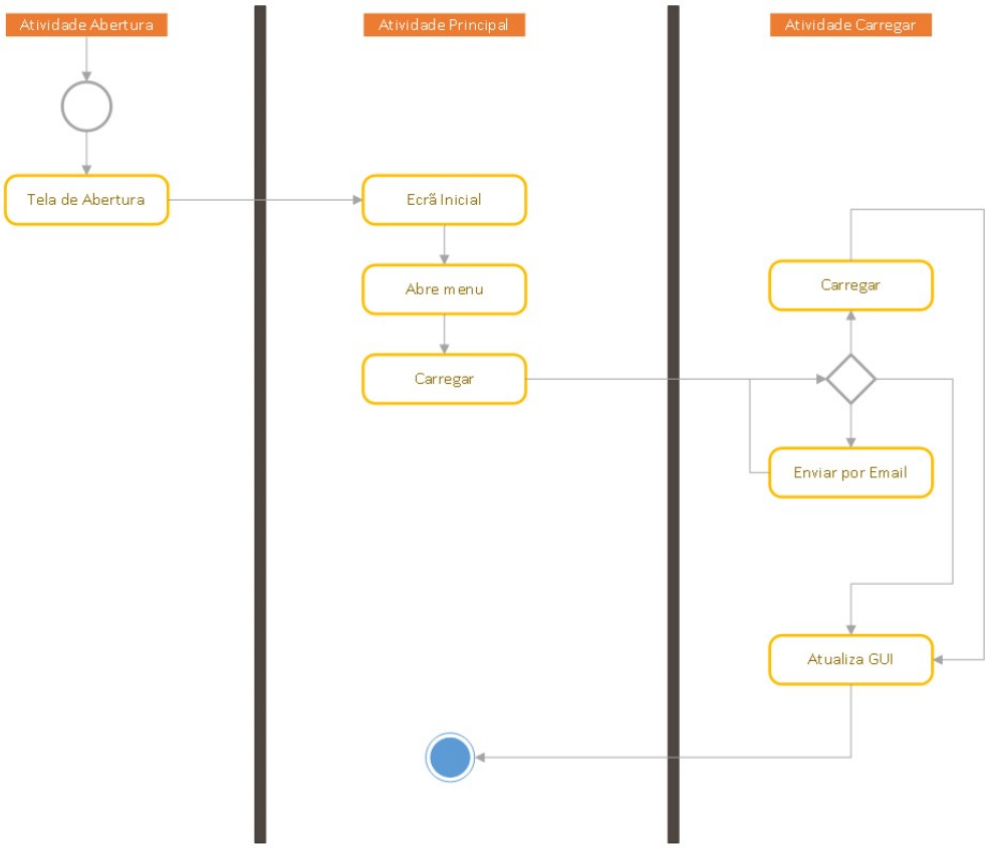


Figura 3.23 Diagrama de Atividades - Logs.

De seguida tem-se na figura 3.24, o diagrama de atividades da calibração. Após a aplicação iniciar, o utilizador seleciona a opção calibração e deve inserir as configurações necessárias para iniciar a calibração, caso não pretenda realizar uma nova calibração poderá sair e voltar ao ecrã inicial. Após inserir corretamente todas as configurações, a aplicação mostrará o ecrã de definições de uma nova experiência. Tendo concluído o processo de configuração da calibração, a aplicação irá pedir ao utilizador que se conecte a um dispositivo. Após existir uma conexão válida, inicia-se o processo de aquisição de dados, no fim desse processo calcula-se a matriz PCA utilizando o método referido no capítulo 2.1. Após a aplicação concluir os cálculos necessários guarda essa informação num ficheiro de calibração para que possa ser utilizado na classificação de novas amostras.

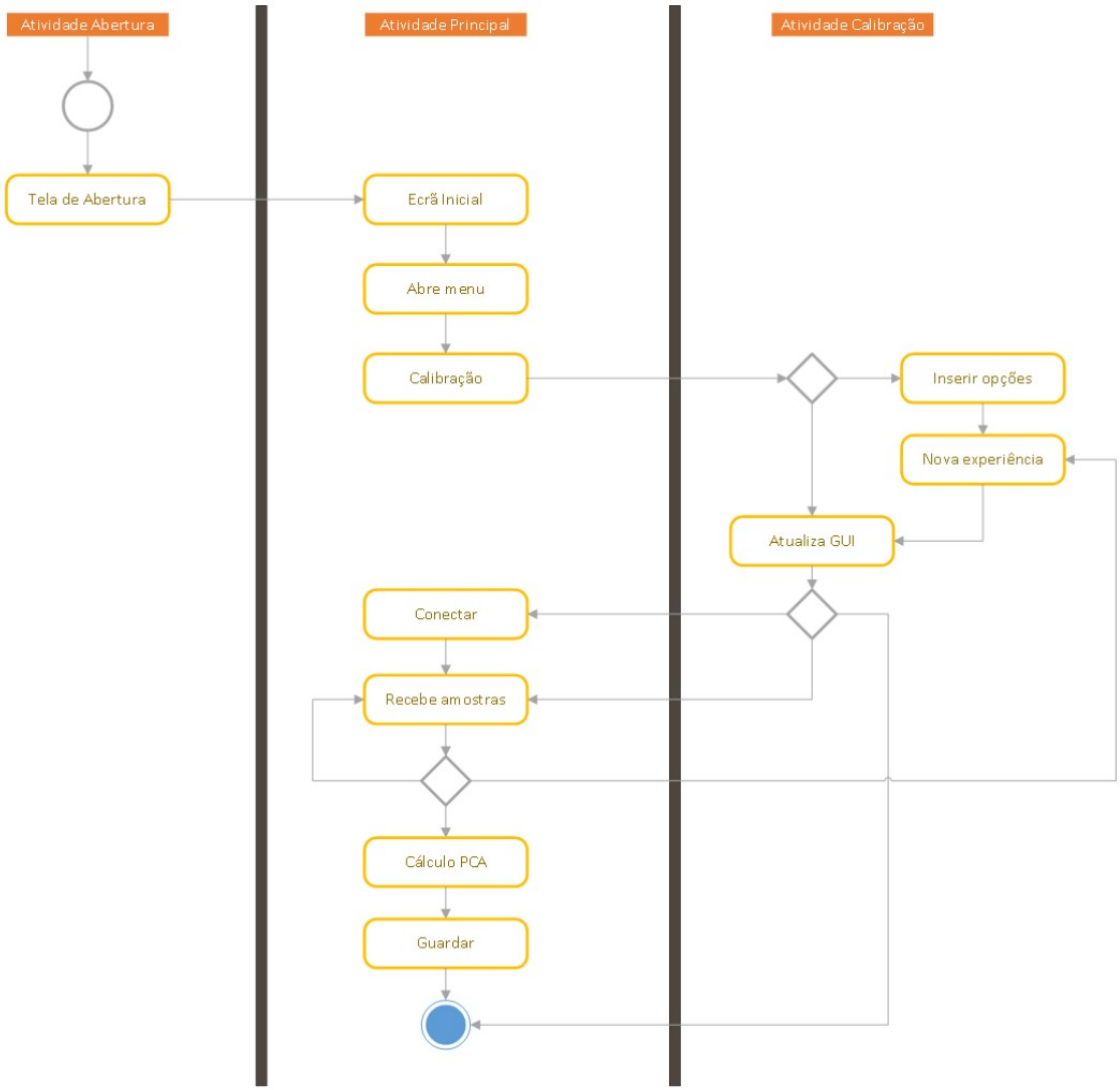


Figura 3.24 Diagrama de Atividades - Calibração.

Por último tem-se a figura 3.25, esta refere-se ao diagrama de atividades da classificação de novas amostras. O utilizador apenas terá acesso a esta atividade após realizar corretamente uma calibração. Caso a aplicação detete que existe um ficheiro válido, o utilizador pode assim conectar-se a um dispositivo. Após existir uma conexão válida, a aplicação irá carregar em memória o ficheiro contendo os dados da calibração, para assim aplicar o método kNN referido no capítulo 2.1, para que possa assim classificar as novas amostras. Quando o utilizador não desejar receber novos valores poderá então desconectar-se do dispositivo e a aplicação voltará ao ecrã inicial.

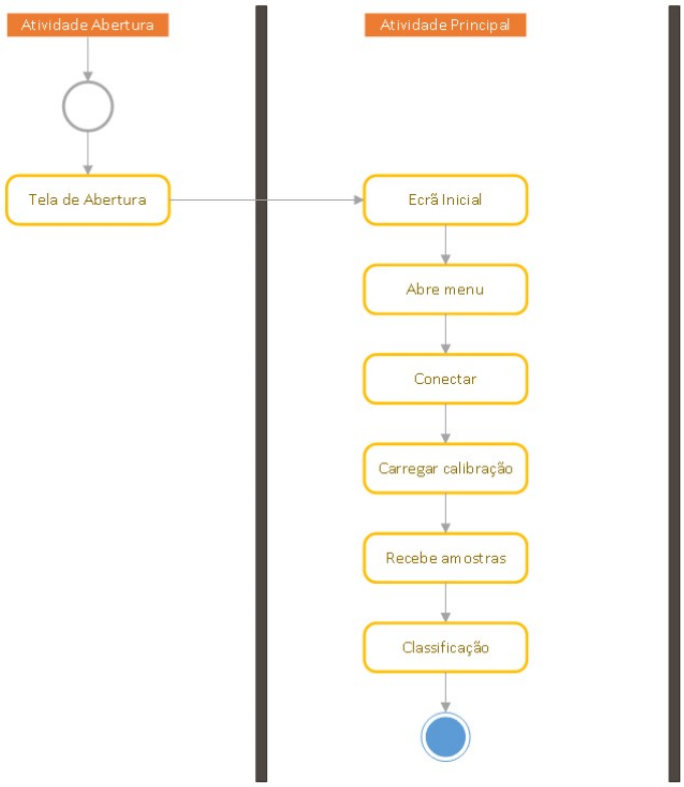


Figura 3.25 Diagrama de Atividades - Classificação novas amostras.

3.4.2.4 Organização e dependências de componentes

No diagrama da figura 3.26 são apresentados os componentes que constituem a aplicação. No lado esquerdo da figura são apresentadas todas as bibliotecas externas ao android que são utilizadas na conceção desta aplicação. Estas bibliotecas foram escolhidas pela necessidade de implementar algumas funcionalidades que não são diretamente implementadas no android, para isso utilizou-se o site Android Arsenal para pesquisar todas as bibliotecas existentes para android.

As bibliotecas ACRA, AppIntro e MPAndroidChart já foram faladas acima, pelo que não se irá repetir a sua funcionalidade neste ponto. Quanto à biblioteca LeakCanary [49], esta é utilizada ao longo da implementação da aplicação para detetar fugas de memória durante a execução da aplicação para assim se conseguir, sempre que possível, modificar a aplicação e assim corrigir esse erro. Com esta biblioteca também é possível, por exemplo detetar uma fuga de memória quando a aplicação não tem permissões para executar alguma tarefa, como é o caso de ler/guardar ficheiros. No fim da implementação da aplicação esta biblioteca não fará parte do sistema.

Por fim tem-se a biblioteca MaterialSeekBarPreference [50], esta será utilizada para implementar uma barra de progresso na atividade definições, visto que, como a aplicação é implementada com a finalidade de funcionar com várias versões android, nalgumas versões anteriores não era possível utilizar esta funcionalidade, pelo que foi necessário recorrer a uma biblioteca externa para contornar esta situação e assim garantir compatibilidade entre todas as versões.

Do lado direito da figura baixo, são apresentadas todas as atividades que compõem a aplicação.

Figura 3.26 Diagrama de Componentes da aplicação desenvolvida.

3.4.2.5 Desenho do sistema

Como se pode ver pela figura 3.27, no fim do projeto é gerado um ficheiro apk como está representado do lado esquerdo da figura, esse apk é instalado no sistema android que corre num *smartphone*, este terá acesso à memória externa do telefone e ao seu bluetooth para se conectar aos dispositivos. Para além disso será ainda necessário o acesso à internet para que a aplicação consiga enviar os erros/falhas para o servidor e para que o utilizador consiga partilhar os ficheiros guardados.

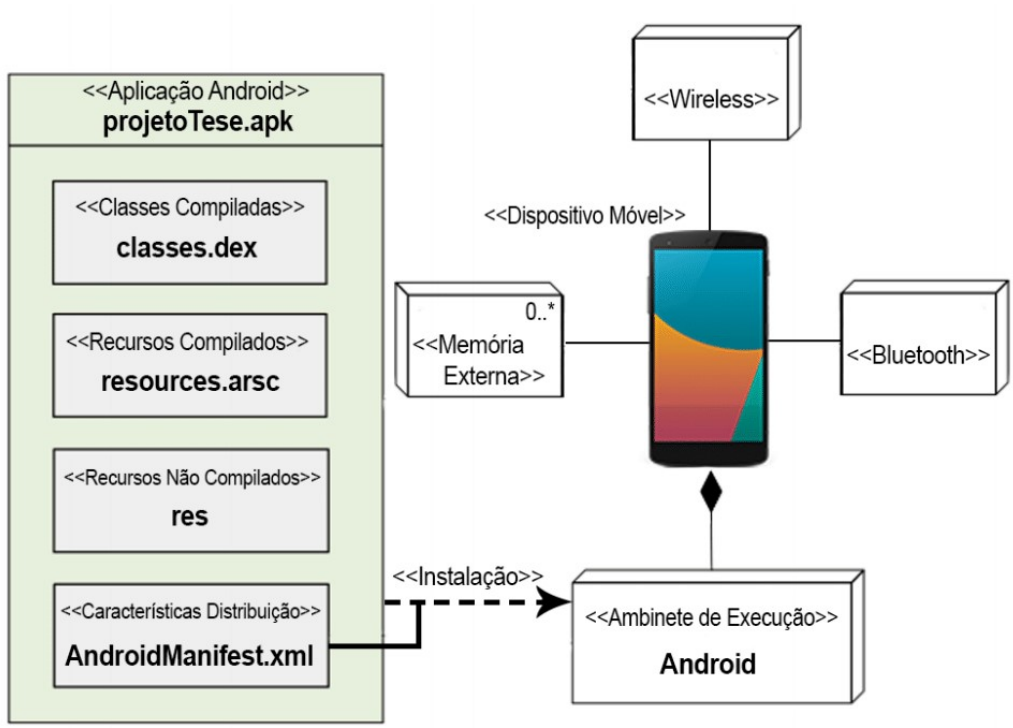


Figura 3.27 Diagrama de Implementação.

3.4.2.6 Desenho da arquitetura de ficheiros

Pela figura 3.28, pode-se visualizar a arquitetura de ficheiros da aplicação, estes são os ficheiros que serão guardados ao longo da execução da mesma.

O ficheiro Preferências, guardará todas as definições do utilizador e tem como campos os definidos nas figuras 3.6a e 3.6b.

O ficheiro Logs, é guardado na extensão .txt, tem como campos o tempo a que a amostra foi retirada (a começar de zero) e o valor da amostra. Para além disso, é ainda guardado o número de amostras adquiridas e o número de sensores que compõem o nariz eletrónico. O nome do ficheiro será da seguinte forma "dia-mês-ano_hora-minuto.txt".

Quanto ao ficheiro Calibração, este guardará os dados obtidos após a análise dos dados e tem como campos os valores do PC1 e PC2 (componentes principais 1 e 2 obtidas através do método PCA), a que cluster pertence a cada amostra, o nome e as coordenadas dos centros dos clusters. A primeira linha do ficheiro terá o número de experiências, o número de amostras e número de sensores.

O último ficheiro refere-se ao dados recebidos durante a calibração, estes são guardados num ficheiro denominado "data_calib.txt" e tem na primeira linha o número de experiências realizadas, o número de amostras por experiência, o número de sensores que compõem o nariz eletrónico. De seguida são guardados todos os dados obtidos, nas colunas são apresentados os dados obtidos por cada sensor e o nome de cada experiência e nas linhas cada amostra obtida.

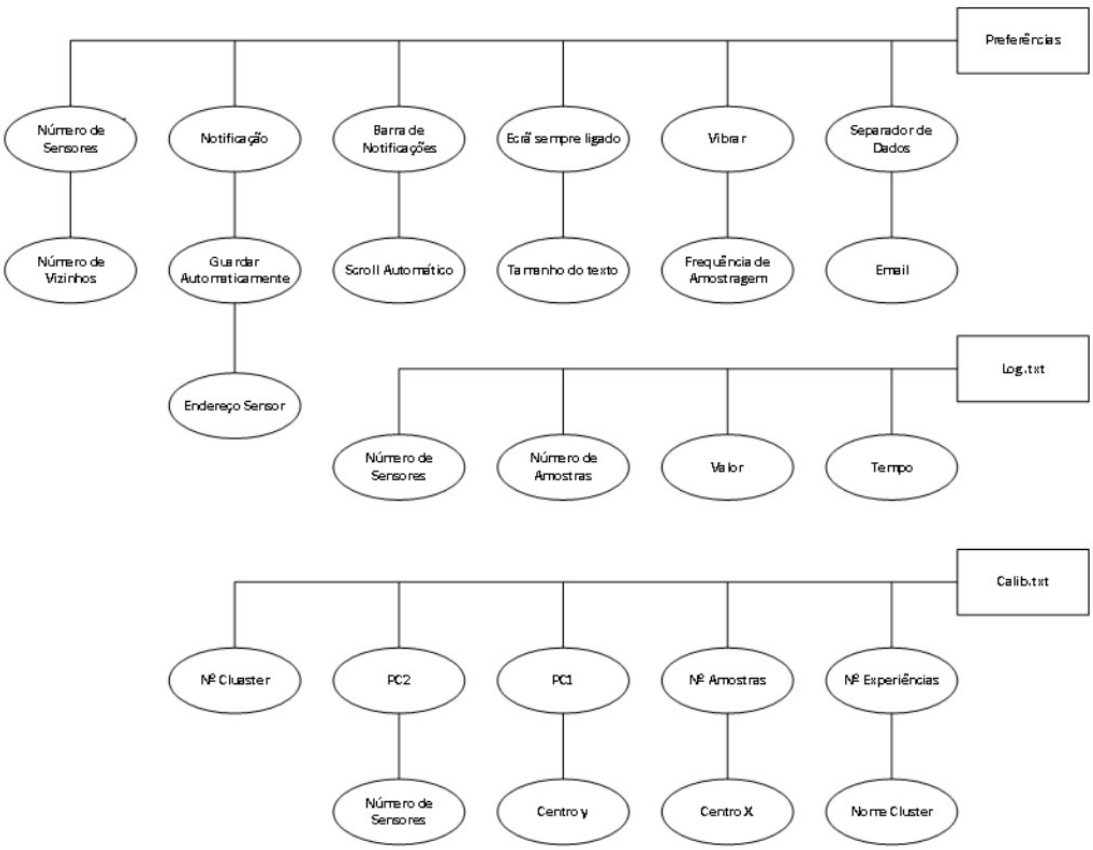


Figura 3.28 Diagrama de Ficheiros.

3.5 Implementação

3.5.1 Padrão de programação

No que toca ao padrão de programação foi utilizada uma estratégia que simplificasse ao máximo a análise do código e da Interface gráfica. Para isso foi utilizado o guia feito pela google, e disponibilizado na página do github, Google Java Style Guide [51].

Em baixo é dado alguns dos exemplos do padrão adotado.

- Nomes das classes - UpperCamelCase (Primeira letra de cada palavra escrita em maiúsculas).
- Nomes dos métodos - LowerCamelCase (Primeira palavra em minúscula, e a primeira letra das outras em maiúscula).
- Constantes - TUDO_EM_MAIÚSCULAS.
- Parâmetros - LowerCamelCase (Primeira palavra em minúscula, e a primeira letra das outras em maiúscula).
- Variáveis locais - LowerCamelCase (Primeira palavra em minúscula, e a primeira letra das outras em maiúscula).
- Variáveis - LowerCamelCase (Primeira palavra em minúscula, e a primeira letra das outras em maiúscula).

Outra parte importante quando se desenvolve um software é os comentários que são colocados ao longo do desenvolvimento e, que no fim permitem gerar um manual de referência da aplicação. No caso do Java, ao manual de referência é dado o nome de Javadoc e existem regras que são necessárias cumprir, para que no fim o manual seja gerado corretamente.

Existem várias formas de apresentar os comentários, em baixo são apresentados alguns casos.

- Várias linhas:

```
/**
* Pode-se escrever várias linhas,
* que depois serão apresentadas seguidas
*/
public int metodo(String p1) { ... }
```

- Só uma linha:
  /** Pode-se também escrever assim */
  public int metodo(String p1) { ...}

Para além de texto a explicar o método implementado, é também possível enumerar e explicar as variáveis que o método recebe e envia, como por exemplo:

```
/**
* Metodo de transformar texto em numero.
*
* @param p1 texto de entrada
* @return o inteiro transformado
*/
public int metodo(String p1) { ... }
```

Código 3.1 Padrão de programação

3.5.2 Funcionalidades implementadas

O último ponto desta secção irá abordar as funcionalidades implementadas e são complementadas com excertos de código. Como referido anteriormente, foram implementadas algumas *AsyncTask* para realizar algum processamento em paralelo com a *task/thread* principal. Nesse sentido, primeiramente fala-se que trabalho essas *tasks* vão realizar. Na secção 3.4.2.1 se referiu que as *tasks* eram implementadas nos ficheiros MainActivity e DeviceListActivity.

Em Java uma *AsyncTask* [52] pode ter vários métodos implementados, mas pelo menos o método doInBackground(Params...) deve estar presente. O código 3.2 apresenta um pequeno exemplo de como se pode implementar uma *AsyncTask* retirado do site do Android Developer.

```
private class DownloadFilesTask extends AsyncTask<URL, Integer,
 Long> {
  protected Long doInBackground(URL... urls) {
    int count = urls.length;
    long totalSize = 0;
    for (int i = 0; i < count; i++) {
      totalSize += Downloader.downloadFile(urls[i]);
      publishProgress((int) ((i / (float) count) * 100));
      if (isCancelled()) break;
    }
    return totalSize;
```

```
    }

    protected void onProgressUpdate(Integer... progress) {
      setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
      showDialog("Downloaded " + result + " bytes");
    }
  }
```

Código 3.2 Exemplo *AsyncTask*

No ficheiro DeviceListActivity implementa-se a searchingTask responsável por descobrir novos dispositivos, de notar que apenas se irá falar das partes mais importante do código, no anexo C é apresentado excertos do código das tasks.

No código apresentado em C.1 mostra-se como é feita a descoberta de dispositivos, o onPreExecute é responsável por realizar tarefas ainda na *thread* principal. Neste caso, é feito o cancelamento da descoberta de dispositivos para que quando se inicia a descoberta de dispositivos essa ser feita do início. No doInBackground inicia-se a descoberta de novos dispositivos, após isso, no onPostExecute executa-se um *Handler* [53] que executa ao fim de 10 segundos, devido ao postDelayed(), um Runnable [54] que faz com que a *AssyncTask* execute ações na *thread* principal, *thread* essa que está associada ao ambiente gráfico.

Quando é executado o startDiscovery() o método *BroadcastReceiver*, apresentado em C.2, sempre que encontra um dispositivo novo vai adicioná-lo à lista de dispositivos encontrados e apresenta automaticamente essa informação ao utilizador.

Quanto ao ficheiro MainActivity são implementadas 7 *AssyncTasks*, o primeiro a ser falado é o turnBluetoothOn que tem como objetivo ligar o bluetooth do telefone.

Como se pode ver pelo código apresentado em C.3, no doInBackground é feita a ativação do bluetooth do telefone, após isso a *task* passa logo para o onPostExecute onde lança um *Handler* que espera 5 segundos(tempo médio para o bluetooth ligar corretamente) e lança a DeviceListActivity.

De seguida tem-se a *task* loadingFile responsável por carregar os históricos para a aplicação. No onPreExecute é feita a abertura do ficheiro, para que este possa ser lido no doInBackground, o texto é lido linha a linha e colocado numa *class*, para que os dados possam ser analisados posteriormente. Enquanto os dados são lidos do ficheiro, são automaticamente apresentados ao utilizador e os gráficos são atualizados, o código referente a esta task pode ser visto em C.4.

A próxima *task* implementada refere-se à conexão a um dispositivo bluetooth, e é iniciada quando o utilizador se deseja ligar a um dispositivo através da DeviceListActivity. Esta *task* é denominada connectBluetooth, código apresentado em C.5 e nesta é criada um *socket* de comunicação.

No doInBackground a aplicação verifica se não existe nenhuma comunicação ativa nesse momento. Caso não exista, procede-se à criação de um *socket* e conecta-se ao dispositivo. No onPostExecute caso a aplicação não se tenha conectado corretamente ao dispositivo, o utilizador receberá um alerta, caso contrário a aplicação irá iniciar a receção de dados.

A próxima *task* é a mais extensa da aplicação pois, é nesta que é feita a receção dos dados, mas antes de os dados começarem a ser enviados pelo dispositivo, é necessário o envio de comandos para que estes sejam enviados corretamente. Primeiro será feita a análise do código e no fim disso, será apresentado a lista com os códigos disponíveis para o utilizador, o código desta task pode ser visto em C.6.

Primeiro no onPreExecute são enviados comandos ao dispositivo para que ele altere a comunicação para bluetooth, de seguida é enviado o comando para que este altere o modo de funcionamento para manual, e por fim é enviado a frequência de amostragem. Após o dispositivo estar configurado com as definições do utilizador, no doInBackground é feita a receção dos dados e atualiza-se os valores no terminal e nos gráficos, como a *AsyncTask* não corre na *thread* de desenho dos dados, é necessário atualizar os valores através da runOnUiThread, esta faz o mesmo que o *Handler* mas está associada à *thread* principal e não à que o criou.

Quando ocorrer algum erro na receção dos dados ou o utilizador pretender desconectar-se, a *AsyncTask* passa para o onPostExecute onde envia ao dispositivo o comando para parar de enviar dados e inicia a *task* disconnectBluetooth que se falará a seguir.

As duas tabelas abaixo são retiradas da dissertação desenvolvida por João Figueiredo [55], uma vez que a aplicação desenvolvida pretende ser compatível com o protótipo realizado nessa dissertação. Foi implementado o mesmo método de comunicação, nem todos os comandos possíveis são mostrados na tabela, apenas são abordados os usados na aplicação.

O protocolo de comunicação tem a seguinte forma:

$$< Flag >< Endereo >< Tamanho >< Comando >< DadosAdicionais >$$

A flag é um byte com o valor hexadecimal 0xFF, o endereço por predefinição é de 0x10 e este pode ser alterado nas definições da aplicação. O tamanho contém o tamanho total da mensagem em bytes, correspondente à soma do comando e dos dados adicionais em bytes. No campo do comando, são utilizados os códigos mostrados na tabela 3.3, e os dados adicionais estão na tabela 3.4.

Tabela 3.3 Comandos disponíveis

Código	Ação
0x13	Altera o modo de comunicação: Bluetooth(uma vez)
0x23	Altera o modo de operação: Manual(uma vez)
0x30	Adquire dado e envia para o Master (Apenas para manual)
0x31	Para a aquisição de dados (Apenas para manual)

Tabela 3.4 Dados adicionais disponíveis

Tamanho(máximo)	Código	Ação
0x01	0x13	NULL
0x01	0x23	NULL
0x03	0x30	Frequência de Amostragem[amostras/segundo] [0x00..0xFF]

Como referido na *task* acima, assim que o utilizador desconectar o dispositivo ou ocorrer algum erro de leitura, é iniciado a *d*isconnectBluetooth. Esta, através de doInBackground fecha a socket de leitura e escrita, após isso, coloca essa variável a null para que a aplicação não tenha mais conhecimento do dispositivo conectado, o código da *task* pode ser visto no anexo C.7.

Por último, tem-se a *task* savingFile que é executada quando o utilizador pretende guardar o ficheiro com o histórico dos dados recebidos. Antes do onPreExecute pode-se ver pelo código C.8 como é feita a inicialização do nome do ficheiro, como já referido na secção 3.4.2.6. Após a inicialização das variáveis no onPreExecute é feita a criação do ficheiro e a sua abertura. No doInBackground é realizado a escrita do ficheiro, essa é feita linha a linha, no fim de cada linha é inserido o caracter "\n". Por fim no onPostExecute é fechado o ficheiro.

Capítulo 4

Testes de Software

No desenvolvimento de um sistema complexo normalmente os testes envolvem várias fases. O primeiro teste é o teste unitário, onde cada módulo da aplicação é testado independentemente dos outros. Este teste tem como objetivo verificar se os componentes funcionam corretamente, se os dados que o utilizador introduz e se os valores mostrados são os esperados.

Quando a aplicação passar neste teste sem nenhuma falha, a equipa de teste deve testar se os vários módulos funcionam corretamente entre eles como definido nas especificações da aplicação. Este teste é denominado teste de integração.

Após se testar a aplicação e se verificar que esta passa a informação entre os vários componentes corretamente de acordo com as especificações, deve-se testar as funcionalidades deste. A este teste é dado o nome de teste funcional e tem com objetivo verificar se a aplicação produz o resultado definido nos requisitos do sistema.

Enquanto que o teste funcional se preocupa com os requisitos funcionais, o teste de performance testa os requisitos não funcionais, como é o caso da usabilidade, segurança, disponibilidade e manutenção. Quando a aplicação passa a este último teste diz-se que se trata de um sistema válido, e a partir deste momento o software está pronto para seguir para o consumidor final.

Os últimos dois testes de que se fala neste capítulo são: teste de aceitação e teste de instalação. O primeiro passa por levar a aplicação até ao *stakeholder* e este testar a aplicação para garantir que o produto final corresponde ao pretendido por este. O outro teste serve para testar se a aplicação instala corretamente no dispositivo pretendido pelo consumidor.

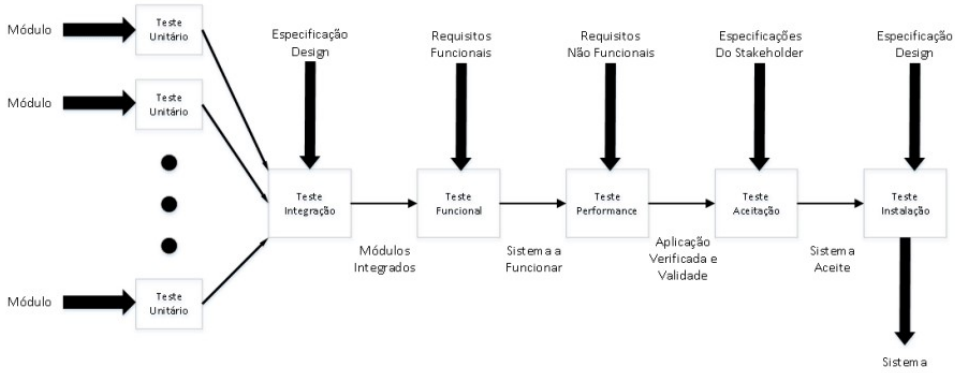
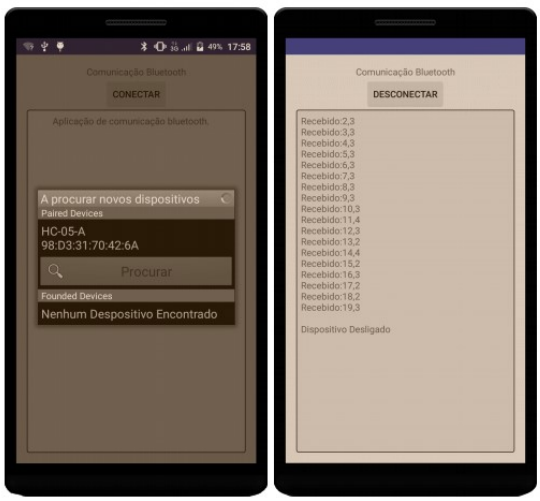


Figura 4.1 Passos dos testes [7].

4.1 Teste unitário

Como falado no texto acima, o teste unitário é feito analisando independentemente cada módulo da aplicação. No caso da aplicação desenvolvida, pode-se dividir em quatro o módulo como se pode ver pela figura 3.1.

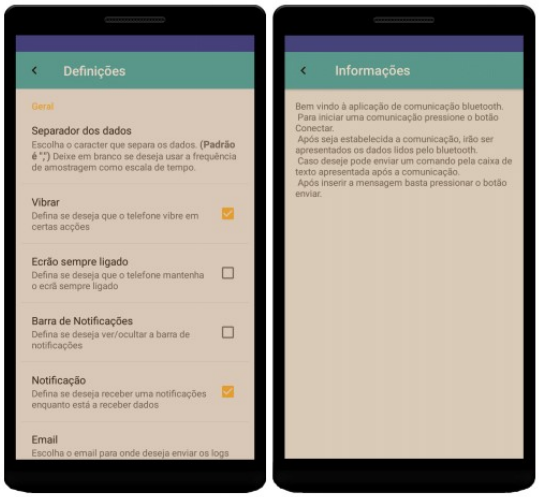
O primeiro módulo referente-se à conexão a um dispositivo e receção de dados através do mesmo, como se pode ver pela tabela 3.1 o módulo engloba os requisitos **A** e **B**. Como referido no capítulo anterior, para o desenvolvimento da aplicação foi utilizado o modelo em cascata com prototipagem, um dos protótipos realizados ao longo desta dissertação foi referente a este módulo, para que fosse possível testar estes componentes independentemente do resto dos módulos, a figura 4.2 apresenta algumas imagens do protótipo realizado.



(a) (b)

Figura 4.2 Protótipo comunicação.

Após implementado e testado corretamente o módulo de comunicação, passou-se para os requisitos **C** e **D**, estes como não dependem de nenhum módulo podem já ser testados na aplicação final. Na figura 4.3 pode-se ver imagens das janelas das definições e das informações da aplicação.



(a) (b)

Figura 4.3 Imagens definições e informações.

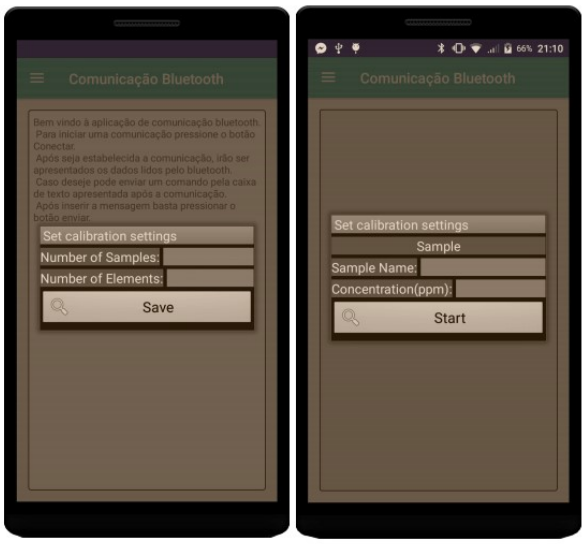
Quanto ao penúltimo módulo, como foi um requisito extra implementado após definida a tabela de requisitos da aplicação, não foi feito um protótipo para este pois, são utilizadas as mesmas janelas que para receber os dados em tempo real e, o princípio de leitura dos logs e o desenho da janela, é muito semelhante ao de conectar/descobrir dispositivos pelo que o teste deste módulo também foi feito na aplicação final.

Na figura 4.4 estão imagens de como os ficheiros e os dados são apresentados ao utilizador.

(a) (b) (c)

Figura 4.4 Imagens carregamento de ficheiros.

Quanto à calibração, esta utiliza as janelas já implementadas para a conexão a dispositivos e receção de dados pelo que neste ponto não são repetidas. Na figura 4.5a é apresentado o ecrã onde o utilizador irá definir os parâmetros da calibração e na figura 4.5b é mostrado o ecrã das definições de cada experiência.



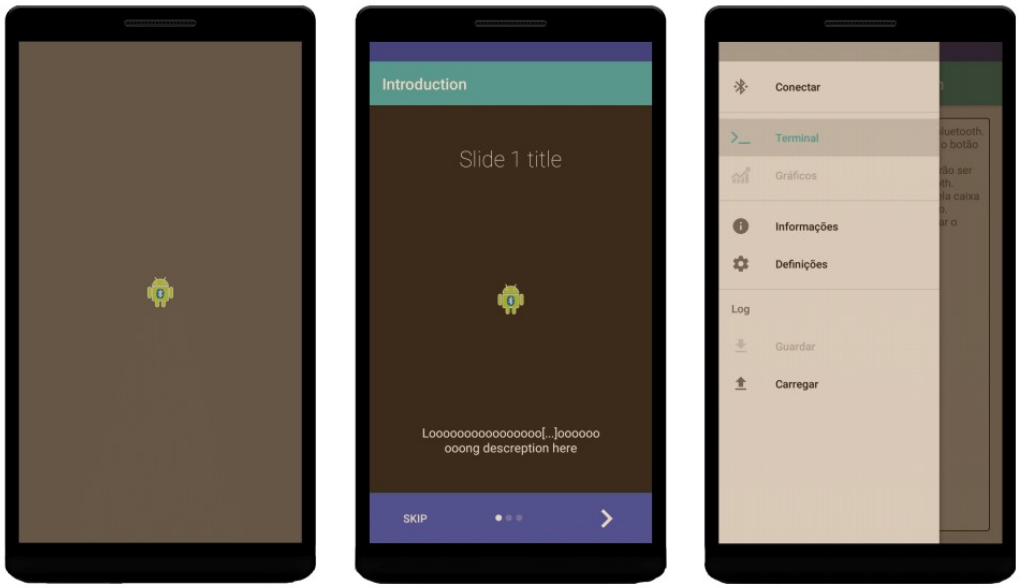
(a) (b)

Figura 4.5 Imagens calibração.

4.2 Teste de integração

Após ser feito o teste a cada módulo independentemente é altura de pegar na aplicação final e testar a interligação entre cada módulo e, a passagem entre janelas. Para este teste foi utilizado uma ferramenta da Google denominada Firebase [56] que permite realizar testes de software à aplicação utilizando robôs. O que este site faz é colocar os robôs a andar pela aplicação a testar as funcionalidades implementadas, o que permite garantir que a passagem entre janelas é feita corretamente, os únicos requisitos não testados pelo site são a receção de dados e o carregamento de ficheiros, que foram feitos manualmente.

Na figura 4.6 são apresentadas três imagens desses testes. As outras funcionalidades já foram apresentadas nas figura acima, pelo que não são apresentadas nesta figura.



(a) (b) (c)

Figura 4.6 Imagens aplicação.

4.3 Teste funcional

Os testes funcionais comparam a performance do sistema atual com os requisitos referidos no capítulo 3.1. Nestes testes desenvolveram-se testes de caso baseados nos requisitos funcionais definidos no capítulo 3.2. Para a realização deste teste foi utilizada a biblioteca Robotium [57] que permite a automatização dos testes pretendidos. Os códigos referentes aos testes encontram-se no anexo D.

O primeiro teste teve como objetivo por à prova as funcionalidades **C**, **D** e **E.1.1** da tabela 3.1. Quanto ao segundo teste, teve como objetivo avaliar as funcionalidades **A.1** e **E.1**. As funcionalidades **A.1.1**, **A.2**, **B** e **E.2** não foram automatizadas pelo que foram realizadas manualmente e são apresentadas no teste 3. Por último testou-se os métodos de extração de características e classificação de novas amostras utilizando para isso três ficheiros contendo amostras de Acetona, Álcool e Amónio.

Teste 1

Na figura 4.7 é apresentado o diagrama de atividade referente ao teste realizado e no anexo D.1 é apresentado o código implementado no Android Studio para realizar o teste, o teste foi efetuado utilizando o telefone MEO A88.

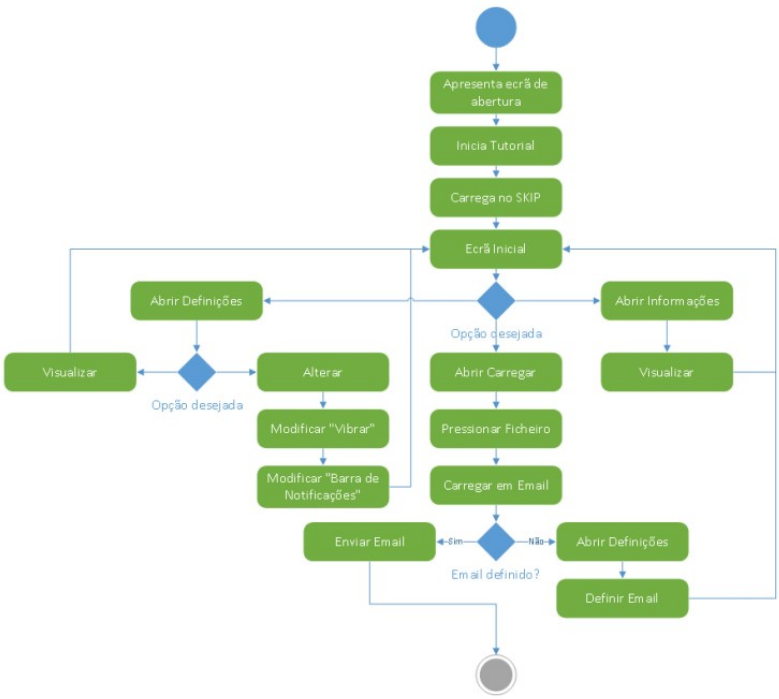


Figura 4.7 Diagrama de atividade teste 1.

Como se pode ver pela figura acima, a aplicação inicia pela primeira vez, pelo que é apresentado o tutorial inicial, após isso o teste faz "Skip" ao tutorial e abre o menu lateral onde se encontram as várias funcionalidades da aplicação. Após isso, é aberto o ecrã das definições da aplicação onde se altera as opções vibrar e barra de notificação, de seguida volta-se ao ecrã inicial e abre-se o ecrã das informações.

Após se visualizar as informações regressa-se ao ecrã inicial e reabre-se o ecrã das definições para garantir que a aplicação guarda corretamente as definições alteradas. Após se confirmar o bom funcionamento desta funcionalidade testa-se o envio dos logs através do menu carregar. Para aceder às opções de carregar o ficheiro e enviar por email é necessário pressionar breves segundos sobre o ficheiro pretendido, visto que o utilizador neste momento ainda não definiu um email válido, necessita de o fazer nas definições da aplicação. Tendo definido o email desejado, poderá então voltar a abrir o ecrã de carregar ficheiros e enviar o ficheiro.

Teste 2

Para este teste foi implementado o código no anexo D.2 e D.3 e na figura 4.8 é apresentado o diagrama de atividades referente ao mesmo.

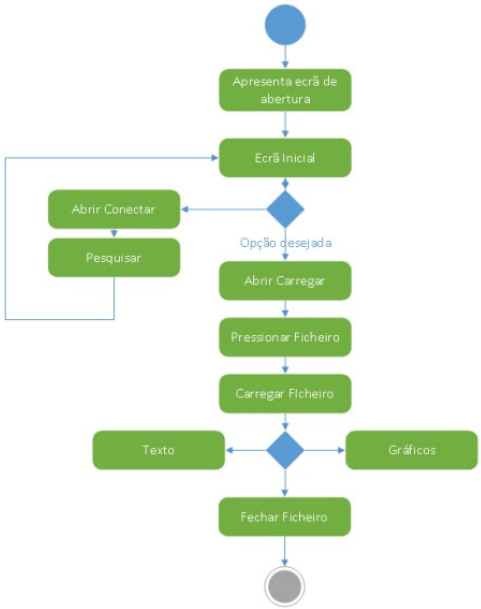


Figura 4.8 Diagrama de Atividade teste 2.

Como se pode ver pela figura acima, neste teste a aplicação já não inicia com o tutorial visto que já não é a primeira execução do utilizador, sendo assim começa-se por abrir o menu lateral e selecionar a opção conectar. Nesta, o utilizador tem a escolha de ligar ou não o bluetooth. Caso não o ligue irá receber uma mensagem de erro e voltará ao ecrã inicial. Se desejar ligar o bluetooth irá ser apresentado o ecrã com os dispositivos emparelhados e a possibilidade de procurar novos dispositivos. Neste teste é feita a procura de novos dispositivos e após feita a pesquisa sai-se do ecrã sem selecionar nenhum dispositivo.

Para realizar a segunda fase deste teste é necessário já ter um ficheiro previamente guardado com dados e para tal utiliza-se um ficheiro com dados aleatórios. Como se pode ver pela figura 4.8, para esta fase do teste é aberto de novo o menu e realizam-se os mesmo procedimentos usados para enviar um ficheiro por email, mas neste caso seleciona-se a opção Carregar. Após o ficheiro ser corretamente carregado para a aplicação, os dados são apresentados na forma de texto mas, acedendo ao menu lateral é possível alterar para o modo de gráficos. Quando não se desejar visualizar mais os dados guardados, pelo menu lateral é possível fechar a leitura.

Teste 3

Na figura 4.9 está apresentado o diagrama de atividades do último teste. Após a aplicação iniciar, realiza-se os mesmos passos do teste anterior até abrir o ecrã de conectar a um dispositivo, neste ecrã é feita a pesquisa por novos dispositivos e após a conclusão dessa pesquisa é feito o emparelhamento com o dispositivo desejado, após o emparelhamento é feita a conexão ao mesmo. Quando a conexão for realizada com êxito começa-se a receber os dados, aí pode-se alternar entre o ecrã de texto e o de gráficos, quando não se desejar receber mais dados pode-se desconectar do dispositivo e no fim abrir o menu lateral e guardar os dados.

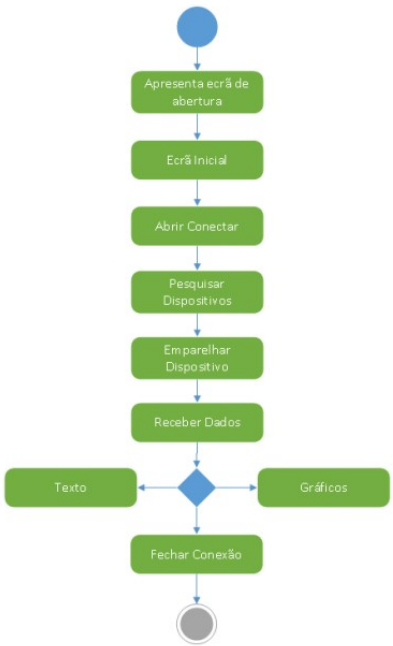


Figura 4.9 Diagrama de Atividade teste 3.

Teste 4

O último teste refere-se aos métodos implementados para análise e classificação de amostras. Para tal, começa-se por realizar uma uniformização dos dados, subtraindo a cada valor obtido por cada sensor a média de todos os valores desse mesmo sensor, este processo faz com que os dados fiquem centrados em zero; de seguida realiza-se a normalização dos dados, pegando no máximo valor de cada sensor e dividindo todos os valores obtidos pelo máximo correspondente ao sensor. Este é um processo importante devido à sensibilidade de cada sensor aos odores, o que faz com que a variação nas amostras entre sensores possa ser muito grande o que cause um outliner nos dados. Assim sendo, a variação de todas as mostras é igual, o que faz com que o peso de todas as amostras seja o mesmo.

Após realizado o pré-processamento dos dados, começa-se por determinar a matriz com os principais componentes (PCA). Para esta dissertação apenas se analisou os dois componentes principais, visto que apenas se desenhou gráficos em 2D. Pelo código E.1, no anexo E, pode-se ver que com os dados normalizados se calculou a matriz de covariância utilizando a biblioteca Commons Math [58] e utilizando a mesma biblioteca retirou-se a partir da matriz de covariância os valores e vetores próprios dessa mesmo matriz. Os valores próprios representam a variância de cada um dos componentes principais, ou seja, os valores próprios com menor valor contêm menor informação sobre a distribuição dos dados, como se pode visualizar na figura 2.5b. Quanto aos vetores próprios estes irão formar os eixos, mas estes apenas dão indicação da direção.

Sabendo que o método utilizado organiza já os valores e vetores próprios, pode-se então utilizar as duas primeiras colunas de cada matriz e multiplicar os vetores próprios pelos dados normalizados e obtém-se assim a projeção dos dados no espaço das características, onde cada par (PC1, PC2) tem a mesma correspondência nos dados anteriormente obtidos (X1, X2,...), onde X1 e X2 representam cada sensor que compõem o nariz eletrónico. Sabendo que os dados analisados provêm de uma calibração, é possível saber a partição dos dados e assim realizar automaticamente o agrupamento das amostras. Na figura **??** está o gráfico obtido através da aplicação.

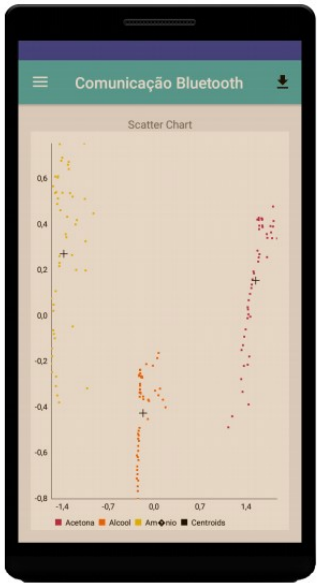
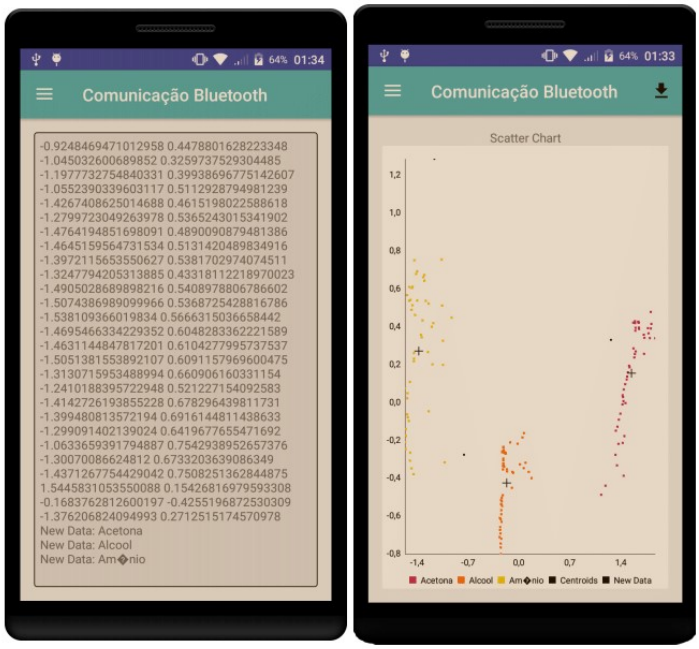


Figura 4.10 Gráfico obtido pela aplicação aplicando o método PCA.

Após se calcular o PCA e o centro de cada grupo de amostras, dá-se por concluída a calibração. Aí, a aplicação irá guardar todos os dados obtidos para puder utilizar quando o utilizador desejar obter novos dados. O método kNN pode ser visto no anexo E.2, aí pode-se ver que quando se obtêm novos dados, estes são primeiramente centrados, normalizados e feita a projeção dos dados no espaço das características como foi realizado na calibração. Após isso é feita uma procura pelos k vizinhos mais próximos utilizando a distância euclidiana. Por fim é feita uma ponderação para descobrir a que grupo de amostras os novos dados pertencem. Na figura 4.11a é apresentado o resultado do algoritmo onde é apresentado ao utilizador o nome do cluster onde esse dado se insere. Na figura 4.11b é apresentado o gráfico com o novo dado representado.



(a) (b)

Figura 4.11 Gráficos obtidas pela aplicação utilizando o método kNN.

4.4 Teste de performance

Como referido no início do capítulo, o teste de performance tem como objetivo verificar os requisitos não funcionais. No capítulo 3 definiram-se alguns requisitos não funcionais que estão presentes na secção 3.2.1.2.

No que toca ao desempenho sempre que foi necessário realizar ações que o utilizador tivesse que esperar algum tempo sem ter o ecrã parado, foram utilizadas *AssyncTasks* com barras de progresso apresentadas no ecrã.

Na parte da usabilidade, foi criado um pequeno tutorial na primeira execução do utilizador e no menu das informações também são apresentadas algumas sugestões de funcionamento da aplicação.

Quanto à disponibilidade, sempre que ocorre um erro esperado pela aplicação, o utilizador recebe um aviso e a aplicação retoma o funcionamento normal, voltando ao ponto anterior que o utilizador estava.

Por fim, tem-se a manutenção. Neste ponto foi implementado um servidor online que recebe todos os erros não esperados pela aplicação, para que a equipa de desenvolvimento possa corrigir os erros.

4.5 Teste de aceitação

Quanto ao teste de aceitação, este não foi possível realizar na sua totalidade visto que não foi possível testar a aplicação com um dispositivo externo, pelo que as funcionalidades de comunicação, mais concretamente o protocolo de comunicação implementado não foi testado num dispositivo real.

4.6 Teste de instalação

Quanto ao teste de instalação, este tem como objetivo garantir que a aplicação correrá no dispositivo pretendido pelo interessado no software. Para tal, utiliza-se de novo o site Firebase [56] para testar no máximo número de dispositivos disponíveis, para além disso o teste foi feito também num tablet Asus Nexus 7 disponibilizado para esta dissertação, em baixo fica uma lista com todos os dispositivos testados com uma nota se o teste passou ou falhou, tabela 4.1.

Tabela 4.1 Dispositivos testados

Dispositivo	Versão(API)	Resultado
LG G4	22	✓
Galaxy S7	24	✓
Pixel	25	✓
Nexus 5	23	✓
Xperia Z2	21	✓
OnePlus One	22	✓
Galaxy S4 mini	19	✓
Nexus 9	21	✓
Nexus 7	19	✓
Smart A88	22	✓
LENNY 3	23	✓

Como se pode ver pela tabela acima a aplicação correu corretamente em todos os dispositivos testados, e em várias versões diferentes pelo que se pode concluir que a aplicação foi bem desenvolvida para ser compatível com telefones com ecrã de vários tamanhos e versões diferentes.

Capítulo 5

Conclusão e Trabalho Futuro

Para a realização desta dissertação foi proposto desenvolvimento de uma aplicação na plataforma android e que, a mesma, fosse capaz de se conectar com dispositivos externos através de Bluetooth. Pretendia-se que esta recebesse em tempo real os dados que são obtidos pelo dispositivo externo e os apresentasse preferencialmente na forma gráfica. Como referido nos testes realizados no capítulo 4, esses requisitos foram implementados com sucesso.

Para além disso a aplicação deveria ser capaz de realizar uma calibração dos dados, onde o utilizador introduz quantas medições diferentes deseja obter e quantas amostras a aplicação deve obter. Após isso, a aplicação analisa os dados através dos métodos referidos no capítulo 2.1. Essa calibração foi testada utilizando ficheiros contendo amostras previamente obtidas.

Fazendo uma breve análise a todo o trabalho desenvolvido nesta dissertação, seria interessante implementar diferentes tipos de gráficos, como por exemplo gráficos com três dimensões para que se possa analisar o PCA com as três principais componentes e não apenas em duas como realizado nesta dissertação. Para além disso, será também interessante aplicar outros métodos de análise de dados testando a sua robustez e rapidez de computação em ambiente Android. Quanto à calibração, deve-se implementar um sistema que permita adicionar novos pontos de calibração ao ficheiro já guardado, e refazer automaticamente os cálculos necessários.

Referências

[1] eMarketer, "Mobile phone, smartphone usage varies globally." [https://www.emarketer.com/Article/Mobile-Phone-Smartphone-Usage-Varies-Globally/1014738 Online; acesso 26-Julho-2017].

[2] J. P. Santos, J. Lozano, and M. Aleixandre, "Electronic noses applications in beer technology," in *Brewing Technology*, InTech, 2017.

[3] K.-T. Tang, S.-W. Chiu, C.-H. Pan, H.-Y. Hsieh, Y.-S. Liang, and S.-C. Liu, "Development of a portable electronic nose system for the detection and classification of fruity odors," *Sensors*, vol. 10, no. 10, pp. 9179–9193, 2010.

[4] Z. Haddi, N. El Barbri, K. Tahri, M. Bougrini, N. El Bari, E. Llobet, and B. Bouchikhi, "Instrumental assessment of red meat origins and their storage time using electronic sensing systems," *Analytical Methods*, vol. 7, no. 12, pp. 5193–5203, 2015.

[5] MathWorks, "Matlab." [https://www.mathworks.com/products/matlab.html Online; acesso 9-Setembro-2017].

[6] J. Liu and J. Yu, "Research on development of android applications," in *Intelligent Networks and Intelligent Systems (ICINIS), 2011 4th International Conference on*, pp. 69–72, IEEE, 2011.

[7] S. L. Pfleeger and J. M. Atlee, *Software engineering: theory and practice*. Pearson, 4 ed., 2009.

[8] Balsamiq, "Balsamiq mockups 3." [https://balsamiq.com/products/mockups/ Online; acesso 9-Setembro-2017].

[9] S. M. Scott, D. James, and Z. Ali, "Data analysis for electronic nose systems," *Microchimica Acta*, vol. 156, no. 3-4, pp. 183–207, 2006.

[10] Google, "Painéis." [https://developer.android.com/about/dashboards/index.html Online; acesso 4-Setembro-2017].

[11] L. Buck and R. Axel, "A novel multigene family may encode odorant receptors: a molecular basis for odor recognition," *Cell*, vol. 65, no. 1, pp. 175–187, 1991.

[12] A. S. M. Mosa, I. Yoo, and L. Sheets, "A systematic review of healthcare applications for smartphones," *BMC medical informatics and decision making*, vol. 12, no. 1, p. 67, 2012.

[13] M. A. Habib, M. S. Mohktar, S. B. Kamaruzzaman, K. S. Lim, T. M. Pin, and F. Ibrahim, "Smartphone-based solutions for fall detection and prevention: challenges and open issues," *Sensors*, vol. 14, no. 4, pp. 7181–7208, 2014.

[14] W. S. Cheung and K. F. Hew, "A review of research methodologies used in studies on mobile handheld devices in k-12 and higher education settings," *Australasian Journal of Educational Technology*, vol. 25, no. 2, 2009.

[15] M. Milrad and D. Spikol, "Anytime, anywhere learning supported by smart phones: Experiences and results from the musis project," *Educational Technology & Society*, vol. 10, no. 4, pp. 62–70, 2007.

[16] C.-y. Chiu and Z. Zhang, "The air quality evaluation based on gas sensor array," in *Semiconductor Technology International Conference (CSTIC), 2017 China*, pp. 1–5, IEEE, 2017.

[17] D. Oletic and V. Bilas, "Design of sensor node for air quality crowdsensing," in *Sensors Applications Symposium (SAS), 2015 IEEE*, pp. 1–5, IEEE, 2015.

[18] K. Persaud and G. Dodd, "Analysis of discrimination mechanisms in the mammalian olfactory system using a model nose," *Nature*, vol. 299, no. 5881, pp. 352–355, 1982.

[19] J. W. Gardner and P. N. Bartlett, *Electronic noses: principles and applications*. Oxford University Press, 1999.

[20] A. D'Amico, C. Di Natale, R. Paolesse, A. Macagnano, E. Martinelli, G. Pennazza, M. Santonico, M. Bernabei, C. Roscioni, G. Galluccio, *et al.*, "Olfactory systems for medical applications," *Sensors and Actuators B: Chemical*, vol. 130, no. 1, pp. 458–465, 2008.

[21] I. A. Casalinuovo, D. Di Pierro, M. Coletta, and P. Di Francesco, "Application of electronic noses for disease diagnosis and food spoilage detection," *Sensors*, vol. 6, no. 11, pp. 1428–1439, 2006.

[22] K. C. Persaud, "Medical applications of odor-sensing devices," *The international journal of lower extremity wounds*, vol. 4, no. 1, pp. 50–56, 2005.

[23] M. Peris and L. Escuder-Gilabert, "A 21st century technique for food control: Electronic noses," *Analytica chimica acta*, vol. 638, no. 1, pp. 1–15, 2009.

[24] S. Ampuero and J. Bosset, "The electronic nose applied to dairy products: a review," *Sensors and Actuators B: Chemical*, vol. 94, no. 1, pp. 1–12, 2003.

[25] E. Schaller, J. O. Bosset, and F. Escher, "'electronic noses' and their application to food," *LWT-Food Science and Technology*, vol. 31, no. 4, pp. 305–316, 1998.

[26] T. C. Pearce, S. S. Schiffman, H. T. Nagle, and J. W. Gardner, *Handbook of machine olfaction: electronic nose technology*. John Wiley & Sons, 2006.

[27] P. Jurs, G. Bakken, and H. McClelland, "Computational methods for the analysis of chemical sensor array data from volatile analytes," *Chemical Reviews*, vol. 100, no. 7, pp. 2649–2678, 2000.

[28] X.-Y. Tian, Q. Cai, and Y.-M. Zhang, "Rapid classification of hairtail fish and pork freshness using an electronic nose based on the pca method," *Sensors*, vol. 12, no. 1, pp. 260–277, 2011.

[29] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial intelligence*, vol. 97, no. 1, pp. 245–271, 1997.

[30] L. I. Smith *et al.*, "A tutorial on principal components analysis," *Cornell University, USA*, vol. 51, no. 52, p. 65, 2002.

[31] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. Wiley, New York, 2012.

[32] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent component analysis*, vol. 46. John Wiley & Sons, 2004.

[33] J. V. Stone, "Independent component analysis: an introduction," *Trends in cognitive sciences*, vol. 6, no. 2, pp. 59–64, 2002.

[34] S. Raschka, "Principal component analysis in 3 simple steps." [https://plot.ly/ipython-notebooks/principal-component-analysis/ Online; acesso 19-Setembro-2017].

[35] A. P. Santana and J. F. Queiró, *ALGEBRA LINEAR E GEOMETRIA ANALÍTICA*. Departamento de Matemática - Universidade de Coimbra, 2003.

[36] D. G. Stork, R. O. Duda, P. E. Hart, and D. Stork, "Pattern classification," *A Wiley-Interscience Publication*, vol. 1, pp. 281–297, 2001.

[37] R. E. Shaffer, S. L. Rose-Pehrsson, and R. A. McGill, "A comparison study of chemical sensor array pattern recognition algorithms," *Analytica Chimica Acta*, vol. 384, no. 3, pp. 305–317, 1999.

[38] N. Instruments, "Labview." [http://www.ni.com/en-nz/shop/labview.html Online; acesso 19-Setembro-2017].

[39] M. Nissfolk, "Development of an electronic nose-tongue data acquisition system using a microcontroller," Master's thesis, Uppsala University, 05 2009.

[40] M. Kasbe, S. Deshmukh, T. Mujawar, V. Bachuwar, L. Deshmukh, and A. Shaligram, "An electronic nose with labview using sno2 based gas sensors: Application to test freshness of the fruits," *International Journal of Scientific & Engineering Research*, vol. 6, no. 4, p. 1977, 2015.

[41] J. Srinonchat, "Development of electronic nose and program for monitoring air pollutions and alarm in industrial area," *International Journal of Computer and Electrical Engineering*, vol. 5, no. 1, p. 61, 2013.

[42] N. Gandhewar and R. Sheikh, "Google android: An emerging software platform for mobile devices," *International Journal on Computer Science and Engineering*, vol. 1, no. 1, pp. 12–17, 2010.

[43] D. Gavalas and D. Economou, "Development platforms for mobile applications: Status and trends," *IEEE software*, vol. 28, no. 1, pp. 77–86, 2011.

[44] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in *Proceedings of the 9th international conference on Software Engineering*, pp. 328–338, IEEE Computer Society Press, 1987.

[45] J. Lehtimaki, *Smashing android UI: responsive user interfaces and design patterns for android phones and tablets*. John Wiley & Sons, 2012.

[46] P. Jahoda, "Mpandroidchart." [https://github.com/PhilJay/MPAndroidChart Online; acesso 19-Agosto-2017].

[47] F43nd1r, "Acra." [https://github.com/ACRA/acra Online; acesso 19-Agosto-2017].

[48] P. Rotolo, "Appintro." [https://github.com/apl-devs/AppIntro Online; acesso 19-Agosto-2017].

[49] I. Square, "Leakcanary." [https://github.com/square/leakcanary Online; acesso 19-Agosto-2017].

[50] MrBIMC, "Materialseekbarpreference." [https://github.com/MrBIMC/MaterialSeekBarPreference Online; acesso 19-Agosto-2017].

[51] Google, "Google java style guide." [https://google.github.io/styleguide/javaguide Online; acesso 19-Agosto-2017].

[52] Google, "Asynctask." [https://developer.android.com/reference/android/os/AsyncTask.html Online; acesso 22-Agosto-2017].

[53] Google, "Handler." [https://developer.android.com/reference/android/os/Handler.html Online; acesso 22-Agosto-2017].

[54] Google, "Runnable." [https://developer.android.com/reference/java/lang/Runnable.html Online; acesso 22-Agosto-2017].

[55] J. M. C. de Figueiredo, "Portable sensor for explosives detection," Master's thesis, DEEC, 2016.

[56] Google, "Firebase." [https://console.firebase.google.com Online; acesso 28-Agosto-2017].

[57] RobotiumTech, "Robotium." [https://github.com/RobotiumTech/robotium/ Online; acesso 4-Setembro-2017].

[58] A. Commons, "Commons math." [http://commons.apache.org/proper/commons-math/ Online; acesso 19-Setembro-2017].

Anexo A

Diagrama de Classes - MainActivity

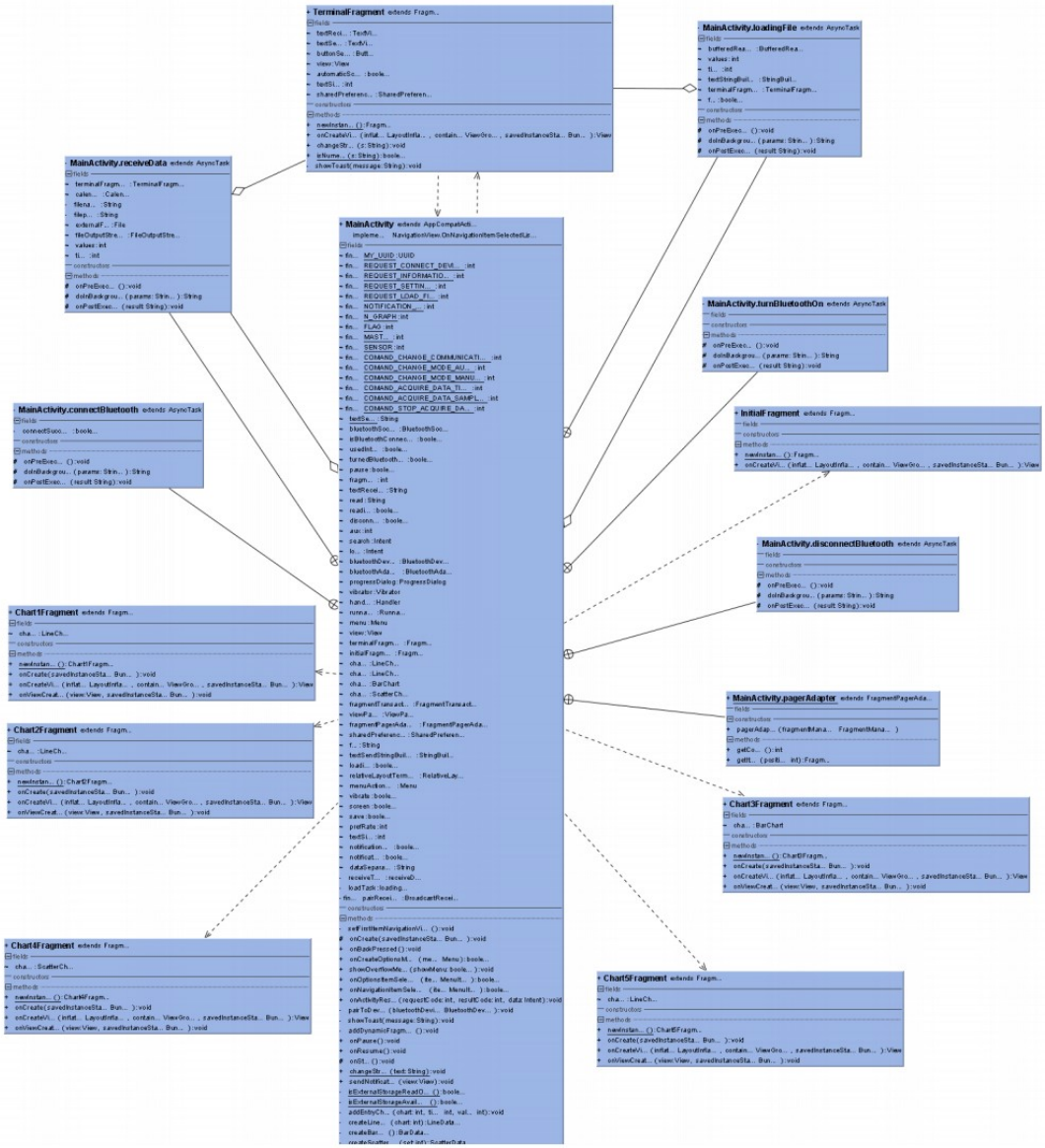
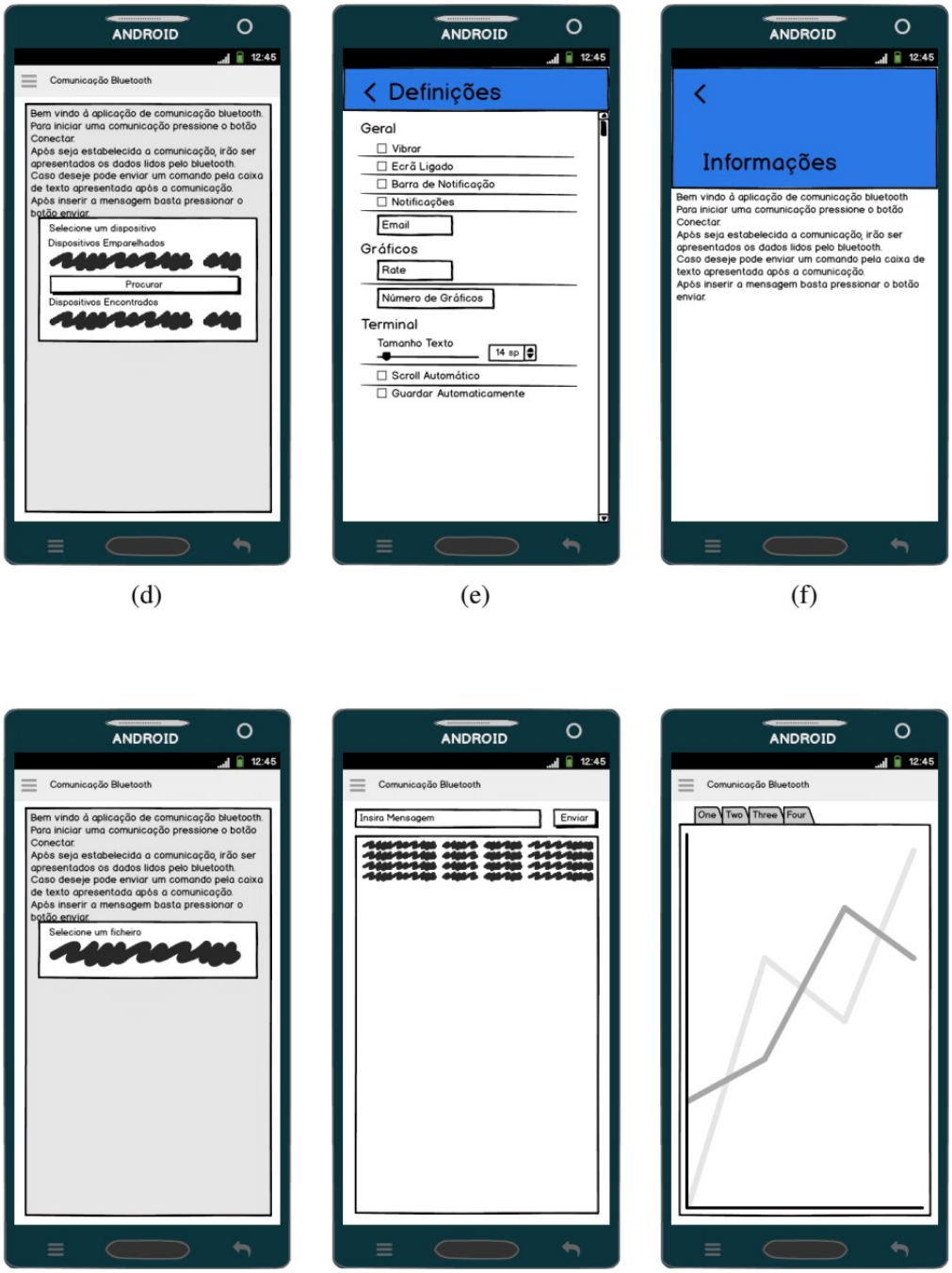


Figura A.1 Diagrama de Classes - MainActivity

Anexo B

Interface com o Utilizador

(a) (b) (c)



(d) (e) (f)

(g) (h) (i)

Figura B.1 Design Conceptual [8].

Anexo C

AsyncTasks desenvolvidas na aplicação

```
private class searchingTask extends AsyncTask<String, Void, String> {
  @Override
  protected void onPreExecute() {
    if (bluetoothAdapter != null) {
      if (bluetoothAdapter.isDiscovering())
        bluetoothAdapter.cancelDiscovery();
    }

    intentFilter.addAction(BluetoothDevice.ACTION_FOUND);
    registerReceiver(foundDevices, intentFilter);
    foundedDevicesArrayAdapter.add(getResources().getText(R.string.
   noDeviceFound).toString());
  }

  @Override
  protected String doInBackground(String... params) {
    bluetoothAdapter.startDiscovery();
  }

  @Override
  protected void onPostExecute(String result) {
    Handler handler = new Handler();
    handler.postDelayed(new Runnable() {
      public void run() {
        if (bluetoothAdapter != null) {
          if (bluetoothAdapter.isDiscovering())
            bluetoothAdapter.cancelDiscovery();
          }
      }
    }, 10000);
```

```
  }
}
```

Código C.1 searchingTask

```
private final BroadcastReceiver foundDevices = new BroadcastReceiver
    () {
  public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();

    if (BluetoothDevice.ACTION_FOUND.equals(action)) {
      BluetoothDevice bluetoothDeviceInfo = intent.getParcelableExtra
    (BluetoothDevice.EXTRA_DEVICE);
      String deviceInfo = bluetoothDeviceInfo.getName() + "\n" +
    bluetoothDeviceInfo.getAddress();
      foundedDevicesArrayAdapter.add(deviceInfo);
    }
  }
};
```

Código C.2 BroadcastReceiver

```
private class turnBluetoothOn extends AsyncTask<String, Void, String>
    {
  @Override
  protected String doInBackground(String... params) {
    bluetoothAdapter.enable();
  }

  @Override
  protected void onPostExecute(String result) {
    Handler handler = new Handler();
    handler.postDelayed(new Runnable() {
      public void run() {
        startActivityForResult(search, REQUEST_CONNECT_DEVICE);
      }
    }, 5000);
  }
}
```

Código C.3 turnBluetoothOn

```
private class loadingFile extends AsyncTask<String, Void, String> {
  BufferedReader bufferedReader;
  StringBuilder textStringBuilder;
```

```
@Override
protected void onPreExecute() {
  try {
    textStringBuilder  = new StringBuilder();
    String filepath = getResources().getText(R.string.
 applicationName).toString();
    File Directory = new File(Environment.
 getExternalStorageDirectory() + File.separator + filepath + File.
 separator + file);
    bufferedReader = new BufferedReader(new FileReader(Directory));
  } catch (IOException e) {
    e.printStackTrace();

    AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.
 this).create();
    alertDialog.setTitle(getResources().getText(R.string.crash).
 toString());
    alertDialog.setCancelable(false);
    alertDialog.setCanceledOnTouchOutside(false);
    alertDialog.setMessage(getResources().getText(R.string.
 loadError).toString());
    alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
    new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
        dialog.dismiss();
        loadTask.cancel(true);
      }
    });

    alertDialog.show();
  }
}

@Override
protected String doInBackground(String... params) {
  try {
    while ((textReceive = bufferedReader.readLine()) != null) {
      textStringBuilder.append(textReceive);
      textStringBuilder.append('\n');
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
```

```
  }

  @Override
  protected void onPostExecute(String result) {
    try {
      bufferedReader.close();
    } catch (IOException e) {
      e.printStackTrace();
    }

    String[] lines = textStringBuilder.toString().split("\n");
    loading = true;

    for (String s : lines) {
      terminalFragment.changeString(s);
      terminalFragment.changeString("\n");

      if(!dataSeparator.equals("")){
        String[] StringSplit = s.split(dataSeparator);
        if(StringSplit.length != 2) {
          break;
        } else{
          values = Integer.parseInt(StringSplit[1]);
          time = Integer.parseInt(StringSplit[0]);

          addEntryChart(1, time, values);
        }
      } else{
        String[] StringSplit = s.split(",");
        values = Integer.parseInt(StringSplit[1]);
        time = Integer.parseInt(StringSplit[0]);

        addEntryChart(1, time, values);
      }
    }
  }
}
```

Código C.4 loadingFile

```
private class connectBluetooth extends AsyncTask<String, Void, String
    > {
  private boolean connectSuccess = true;
```

```
@Override
protected String doInBackground(String... params) {
  try {
    if (bluetoothSocket == null || !isBluetoothConnected) {
      bluetoothSocket = bluetoothDevice.
 createInsecureRfcommSocketToServiceRecord(MY_UUID);
      BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
      bluetoothSocket.connect();
    }
  } catch (IOException e) {
    connectSuccess = false;
    e.printStackTrace();
  }
}

@Override
protected void onPostExecute(String result)
{
  if (!connectSuccess) {
    AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.
 this).create();
    alertDialog.setTitle(getResources().getText(R.string.
 connectionFail).toString());
    alertDialog.setCancelable(false);
    alertDialog.setCanceledOnTouchOutside(false);
    alertDialog.setMessage(getResources().getText(R.string.
 connectionFailMessage).toString());
    alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
    new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
        isBluetoothConnected = false;
        bluetoothSocket = null;

        dialog.dismiss();
      }
    });

    alertDialog.show();
  } else {
    isBluetoothConnected = true;

    if(vibrate)
      vibrator.vibrate(500);
```

```
      fragment = 2;
      addDynamicFragment();
    }
  }
}
```

Código C.5 connectBluetooth

```
private class receiveData extends AsyncTask<String, String, String> {
  Calendar calendar = Calendar.getInstance();
  private String filename = calendar.get(Calendar.DAY_OF_MONTH) + "-"
    + calendar.get(Calendar.MONTH) + "-" + calendar.get(Calendar.YEAR
   ) + "_" + calendar.get(Calendar.HOUR_OF_DAY) + "-" + calendar.get(
   Calendar.MINUTE) + ".txt";
  private String filepath = getResources().getText(R.string.
   applicationName).toString();
  File externalFile;
  FileOutputStream fileOutputStream;
  int values;
  int time;

  @Override
  protected void onPreExecute() {
    super.onPreExecute();

    try {
      bluetoothSocket.getOutputStream().write(FLAG);
      bluetoothSocket.getOutputStream().write(SENSOR);
      bluetoothSocket.getOutputStream().write(0x01);
      bluetoothSocket.getOutputStream().write(
   COMAND_CHANGE_COMMUNICATION);
      bluetoothSocket.getOutputStream().flush();

      bluetoothSocket.getOutputStream().write(FLAG);
      bluetoothSocket.getOutputStream().write(SENSOR);
      bluetoothSocket.getOutputStream().write(0x01);
      bluetoothSocket.getOutputStream().write(
   COMAND_CHANGE_MODE_MANUAL);
      bluetoothSocket.getOutputStream().flush();

      bluetoothSocket.getOutputStream().write(FLAG);
      bluetoothSocket.getOutputStream().write(SENSOR);
      bluetoothSocket.getOutputStream().write(0x03);
      bluetoothSocket.getOutputStream().write(COMAND_ACQUIRE_DATA);
```

```
    bluetoothSocket.getOutputStream().write(Integer.toHexString(
prefRate).getBytes());
    bluetoothSocket.getOutputStream().flush();
  } catch (IOException e) {
    e.printStackTrace();

    AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.
this).create();
    alertDialog.setTitle(getResources().getText(R.string.crash).
toString());
    alertDialog.setCancelable(false);
    alertDialog.setCanceledOnTouchOutside(false);
    alertDialog.setMessage(getResources().getText(R.string.
connectionFailMessage).toString());
    alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
    new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
        dialog.dismiss();
        receiveTask.cancel(true);
        new disconnectBluetooth().execute("");
      }
    });

    alertDialog.show();
  }

  if (!isExternalStorageAvailable() || isExternalStorageReadOnly())
  {
    save = false;
  } else {
    File Directory = new File(Environment.
getExternalStorageDirectory() + File.separator + filepath );
    Directory.mkdirs();
    externalFile = new File(Directory, filename);
  }

  if(save){
    try {
      fileOutputStream = new FileOutputStream(externalFile);
    } catch (Exception e) {
      e.printStackTrace();
      save = false;
```

```
        AlertDialog alertDialog = new AlertDialog.Builder(
  MainActivity.this).create();
        alertDialog.setTitle(getResources().getText(R.string.crash).
  toString());
        alertDialog.setCancelable(false);
        alertDialog.setCanceledOnTouchOutside(false);
        alertDialog.setMessage(getResources().getText(R.string.
  saveError).toString());
        alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
        new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
          }
        });

        alertDialog.show();
      }
    }
  }

@Override
protected String doInBackground(String... params) {
  byte[] buffer = new byte[1024];
  int bytes, length, pos;
  StringBuilder queue = new StringBuilder("");
  InputStream tmpIn;
  String readMessage = "";
  boolean constructed = false;
  time = 0;

  while(bluetoothSocket != null) {
    try {
      reading = true;
      tmpIn = bluetoothSocket.getInputStream();
      bytes = tmpIn.read(buffer);

      if (bytes == 0xFF) {
        bytes = tmpIn.read();
        if (bytes == 0xFA) {
          length = tmpIn.read();
          if (length == 0)
            readMessage = "ACK";
          else {
            for (int i = 0; i < length; i++) {
```

```
            bytes = tmpIn.read();
            readMessage += (char) bytes;
          }
        }

        constructed = true;
      }
    }

    if (constructed && !readMessage.equals("ACK")) {
      queue.append(readMessage);
      pos = queue.indexOf("\n");

      while (pos >= 0) {
        read = queue.substring(0, pos - 1);
        textReceive = read;
        queue.delete(0, pos + 1);
        pos = queue.indexOf("\n");
      }

      constructed = false;
      readMessage = "";
    }
  } catch (IOException e) {
    reading = false;
    disconnect = true;
    break;
  }

  reading = false;

  if (disconnect)
    break;
  else {
    runOnUiThread(new Runnable() {
      @Override
      public void run() {
        if (textReceive.equals(read)) {
          terminalFragment.changeString(getResources().getText(R.
string.textReceived).toString() + ":" + textReceive + "\n");

          if(!dataSeparator.equals("")) {
            String[] StringSplit = textReceive.split(
dataSeparator);
```

```
            values = Integer.parseInt(StringSplit[1]);
            time = Integer.parseInt(StringSplit[0]);

            addEntryChart(1, time, values);

            if (save) {
              try {
                fileOutputStream.write(textReceive.getBytes());
                fileOutputStream.write('\n');
              } catch (Exception e) {
                e.printStackTrace();
              }
            } else {
              textSendStringBuilder.append(textReceive);
              textSendStringBuilder.append('\n');
            }
          } else {
            values = Integer.parseInt(textReceive);

            addEntryChart(1, time, values);

            if (save) {
              try {
                fileOutputStream.write(time);
                fileOutputStream.write(';');
                fileOutputStream.write(textReceive.getBytes());
                fileOutputStream.write('\n');
              } catch (Exception e) {
                e.printStackTrace();
              }
            } else {
              textSendStringBuilder.append(time);
              textSendStringBuilder.append(';');
              textSendStringBuilder.append(textReceive);
              textSendStringBuilder.append('\n');
            }

            time += prefRate;
          }

          read = "";
        }
      }
    });
```

```
      }
    }
  }

  @Override
  protected void onPostExecute(String result) {
    if(disconnect) {
      terminalFragment.changeString("\n" + getResources().getText(R.
    string.deviceOFF).toString());

      if(save){
        try {
          fileOutputStream.close();
        } catch (Exception e) {
          e.printStackTrace();
          showToast(getResources().getText(R.string.errorSavingFile).
    toString());
        }
      }

      try {
        bluetoothSocket.getOutputStream().write(FLAG);
        bluetoothSocket.getOutputStream().write(SENSOR);
        bluetoothSocket.getOutputStream().write(0x03);
        bluetoothSocket.getOutputStream().write(
    COMAND_STOP_ACQUIRE_DATA);
        bluetoothSocket.getOutputStream().flush();
      } catch (IOException e) {
        e.printStackTrace();
      }

      new disconnectBluetooth().execute("");
    }
  }
}
```

Código C.6 receiveData

```
private class disconnectBluetooth extends AsyncTask<String, String,
    String> {
  @Override
  protected String doInBackground(String... params) {
    try {
      if (bluetoothSocket != null || isBluetoothConnected) {
```

```
        bluetoothSocket.close();
        bluetoothSocket = null;
        isBluetoothConnected = false;
        disconnect = false;
        reading = false;
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Código C.7 disconnectBluetooth

```
private class savingFile extends AsyncTask<String, Void, String> {
  Calendar calendar = Calendar.getInstance();
  String filename = calendar.get(Calendar.DAY_OF_MONTH) + "-" +
   calendar.get(Calendar.MONTH) + "-" + calendar.get(Calendar.YEAR) +
    "_" + calendar.get(Calendar.HOUR_OF_DAY) + "-" + calendar.get(
   Calendar.MINUTE) + ".txt";
  String filepath = getResources().getText(R.string.applicationName).
   toString();
  File externalFile;
  FileOutputStream fileOutputStream;
  boolean save;

  @Override
  protected void onPreExecute() {
    if (isExternalStorageAvailable() || !isExternalStorageReadOnly())
    {
      save = true;
      File Directory = new File(Environment.
   getExternalStorageDirectory() + "/" + filepath );
      Directory.mkdirs();
      externalFile = new File(Directory, filename);

      try {
        fileOutputStream = new FileOutputStream(externalFile);
      } catch (Exception e) {
        e.printStackTrace();
        save = false;

        AlertDialog alertDialog = new AlertDialog.Builder(
   MainActivity.this).create();
```

```
      alertDialog.setTitle(getResources().getText(R.string.crash).
  toString());
      alertDialog.setCancelable(false);
      alertDialog.setCanceledOnTouchOutside(false);
      alertDialog.setMessage(getResources().getText(R.string.
  saveError).toString());
      alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
      new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
          dialog.dismiss();
        }
      });

      alertDialog.show();
    }
   } else {
    save = false;

    AlertDialog alertDialog = new AlertDialog.Builder(MainActivity.
  this).create();
    alertDialog.setTitle(getResources().getText(R.string.crash).
  toString());
    alertDialog.setCancelable(false);
    alertDialog.setCanceledOnTouchOutside(false);
    alertDialog.setMessage(getResources().getText(R.string.
  saveError).toString());
    alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
    new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
        dialog.dismiss();
      }
    });

    alertDialog.show();
   }
}

@Override
protected String doInBackground(String... params) {
  String[] lines = textSendStringBuilder.toString().split("\n");

  if (save) {
    try {
      for (String line : lines) {
```

```
            fileOutputStream.write(line.getBytes());
            fileOutputStream.write('\n');
          }
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    }

    @Override
    protected void onPostExecute(String result){
      if(save){
        try {
          fileOutputStream.close();
        } catch (Exception e) {
          e.printStackTrace();
          showToast(getResources().getText(R.string.errorSavingFile).
     toString());
        }
      }
    }
}
```

Código C.8 savingFile

Anexo D

Testes de Software

```
public void testSettingsInformationEmail() throws Exception {
  solo.clickOnButton("SKIP");
  solo.clickOnImageButton(0);
  solo.clickOnText(solo.getString(R.string.settings));
  solo.assertCurrentActivity("Expected Settings Activity",
   SettingsActivity.class);
  solo.clickOnText(solo.getString(R.string.vibrate));
  solo.clickOnText(solo.getString(R.string.notificationBar));
  solo.goBack();

  solo.clickOnImageButton(0);
  solo.clickOnText(solo.getString(R.string.information));
  solo.assertCurrentActivity("Expected Settings Activity",
   InformationsActivity.class);
  solo.goBack();

  solo.clickOnImageButton(0);
  solo.clickOnText(solo.getString(R.string.settings));
  solo.assertCurrentActivity("Expected Settings Activity",
   SettingsActivity.class);
  solo.goBack();

  solo.clickOnImageButton(0);
  solo.clickOnText(solo.getString(R.string.load));
  solo.assertCurrentActivity("Expected Settings Activity",
   LoadActivity.class);
  solo.clickLongInList(1);
  solo.clickOnText(solo.getString(R.string.send));
  solo.goBack();
  solo.clickOnImageButton(0);
```

```
  solo.clickOnText(solo.getString(R.string.settings));
  solo.assertCurrentActivity("Expected Settings Activity",
   SettingsActivity.class);
  solo.clickOnText(solo.getString(R.string.email));
  solo.enterText(0, NOTE_1);
  solo.clickOnButton("OK");
  solo.goBack();
  solo.clickOnImageButton(0);
  solo.clickOnText(solo.getString(R.string.load));
  solo.assertCurrentActivity("Expected Settings Activity",
   LoadActivity.class);
  solo.clickLongInList(1);
  solo.goBack();
  solo.goBack();
}
```

Código D.1 Teste Definições, Informações e Enviar email

```
public void testConnect() throws Exception {
  solo.clickOnImageButton(0);
  solo.sendKey(KeyEvent.KEYCODE_DPAD_DOWN); // select first item
  solo.sendKey(KeyEvent.KEYCODE_DPAD_CENTER);  // press the first
   item
  solo.clickOnButton(solo.getString(R.string.no));
  solo.clickOnButton("OK");
  solo.clickOnImageButton(0);
  solo.sendKey(KeyEvent.KEYCODE_DPAD_DOWN); // select first item
  solo.sendKey(KeyEvent.KEYCODE_DPAD_CENTER);  // press the first
   item
  solo.clickOnButton(solo.getString(R.string.yes));
  solo.assertCurrentActivity("Expected Settings Activity",
   DeviceListActivity.class);
  solo.clickOnButton(solo.getString(R.string.searchDevice));
  solo.clickOnText(solo.getString(R.string.noDeviceFound));

  testLoad();
}
```

Código D.2 Teste Conectar

```
public void testLoad() throws Exception {
  solo.clickOnImageButton(0);
  solo.clickOnText(solo.getString(R.string.load));
  solo.assertCurrentActivity("Expected Settings Activity",
   LoadActivity.class);
```

```
    solo.clickLongInList(1);
    solo.clickOnText(solo.getString(R.string.load));
    solo.assertCurrentActivity("Expected Settings Activity",
     MainActivity.class);
    solo.clickOnImageButton(0);
    solo.sendKey(KeyEvent.KEYCODE_DPAD_DOWN); // select first item
    solo.sendKey(KeyEvent.KEYCODE_DPAD_DOWN); // select first item
    solo.sendKey(KeyEvent.KEYCODE_DPAD_DOWN); // select first item
    solo.sendKey(KeyEvent.KEYCODE_DPAD_CENTER);  // press the first
     item
    solo.clickOnImageButton(0);
    solo.clickOnText(solo.getString(R.string.close));
}
```

Código D.3 Teste Carregar Ficheiro

Anexo E

Testes de Software

```
static dataInformation PCA_Start(dataInformation data){
  data = centerData(data);
  data = normalizeData(data);
  RealMatrix mx = createRealMatrix(data.data);
  RealMatrix cov = new Covariance(mx).getCovarianceMatrix();
  data.pcaData.covMatrix = cov.getData();

  EigenDecomposition evd = new EigenDecomposition(createRealMatrix(
   data.pcaData.covMatrix));
  data.eigenData.values = evd.getRealEigenvalues();
  RealMatrix vectors = evd.getV();
  data.eigenData.vectors = vectors.getData();

  double tot = sum(data.eigenData.values);
  double[] var\_exp = new double[data.eigenData.values.length];
  for(int i = 0; i < data.eigenData.values.length; i++)
    var\_exp[i] = (data.eigenData.values[i] / tot)*100;
  data.eigenData.percent = var\_exp;

  double[][] aux = new double[2][data.numComponents];
  for(int i = 0; i < data.numComponents; i++){
    aux[0][i] = data.eigenData.vectors[i][0];
    aux[1][i] = data.eigenData.vectors[i][1];
  }

  data.pcaData.pca = new double[data.size][2];

  for(int i = 0; i < data.size; i++){
    data.pcaData.pca[i][0] = dot(data.data[i],aux[0]);
    data.pcaData.pca[i][1] = dot(data.data[i],aux[1]);
```

```
  }

  return data;
}
```

Código E.1 Cálculo PCA

```
static dataInformation knn_Start(dataInformation data, double[]
   testData, int k){
  for(int i = 0; i < testData.length; i++) {
    testData[i] -=  data.pcaData.meanData[i];
    testData[i] /=  data.pcaData.maxData[i];
  }

  double[][] aux = new double[2][data.numComponents];
  for(int i = 0; i < data.numComponents; i++){
    aux[0][i] = data.eigenData.vectors[i][0];
    aux[1][i] = data.eigenData.vectors[i][1];
  }

  double[] newData = new double[2];
  for(int i = 0; i < 2; i++)
    newData[i] = dot(testData,aux[i]);

  double[][] neighborsData = getNeighbors(newData, data, k);
  data.testData.testData = newData;
  data.testData.ind = classify(neighborsData, data);

  return  data;
}
```

Código E.2 Algoritmo kNN