Jorge Henrique Santos Oliveira

Desenvolvimento de sistemas para o reconhecimento automático de padrões biológicos para as diversas plataformas móveis Smartphone

Departamento de Matemática
Faculdade de Ciências da Universidade do Porto
Setembro de 2013

Jorge Henrique Santos Oliveira

Desenvolvimento de sistemas para o reconhecimento automático de padrões biológicos para as diversas plataformas móveis Smartphone

Dissertação submetida à Faculdade de Ciências da Universidade do Porto como parte dos requisitos para a obtenção do grau de Mestre em Engenharia Matemática

Orientador: Prof. Doutor André R.S. Marçal,
Professor Auxiliar do Departamento de Matemática Aplicada
da Faculdade de Ciências da Universidade do Porto

Departamento de Matemática
Faculdade de Ciências da Universidade do Porto
Setembro de 2013

Quero agradecer toda à minha família por todo o apoio que me deram ao longo deste ano lectivo

Agradecimentos

Gostaria de agradecer ao Professor Doutor André R.S. Marçal do departamento de Matemática da Faculdade de Ciências da Universidade do Porto por todo o apoio fornecido ao longo deste Projeto e pelo seu excelente trabalho como orientador que contribuíram para que este projeto fosse um sucesso. Em segundo lugar gostaria de agradecer à Professora Doutora Cristina Caridade do Instituto Superior de Engenharia de Coimbra, por todo o suporte técnico oferecido na parte de análise e processamento de imagens dot-blot.

Em terceiro lugar gostaria de agradecer ao Professor Doutor Miguel Coimbra do departamento de Ciências da Computação da Universidade do Porto e ao Senhor Bruno Lopes pelo seu auxílio prestado na cadeira de Visão Computacional pertence ao plano curricular do Mestrado em Ciências da Computação, cujo suporte foi vital para a construção da aplicação ŞAnálise e Processamento de Imagens dot-blot"para o sistema java e androide.

Durante o meu estágio académico que decorreu na empresa AppGeneration gostaria de agradecer ao Senhor Doutor Eduardo Carqueja (Fundador e Diretor Executivo da AppGeneration), pela oportunidade única que me forneceu ao aceitar-me como estagiário na AppGeneration.

Gostaria de agradecer a todos os colaboradores da AppGeneration pela amizade e carinho dado ao longo deste estágio, em especial ao Engenheiro Hugo Simões e ao Engenheiro Ricardo Gomes pela sua orientação tutorial dentro da empresa AppGeneration.

Gostaria de agradecer ao meu colega Pedro Silva pelo suporte técnico oferecido na parte de análise e processamento de imagens folha.

Por fim mas não menos importante, gostaria de agradecer a toda a minha família por todo o apoio prestado ao longo destes dois anos de estudo no Mestrado em Engenharia Matemática.

Resumo

A presente dissertação intitulada *Desenvolvimento de sistemas para o reconhecimento automático de padrões biológicos para as diversas plataformas móveis Smartphone* foi desenvolvida no ano letivo de 2012/2013 no âmbito do curso de Mestrado em Engenharia Matemática da Faculdade de Ciências da Universidade do Porto.

Hoje em dia existe uma grande tendência para a criação de aplicações para os mais diversos sistemas móveis, fruto do grande desenvolvimento tecnológico que se tem registado nos últimos anos, principalmente na área dos microprocessadores. Estas aplicações pretendem realizar de forma automática tarefas que anteriormente eram feitas pelos seres humanos, sendo que um dos exemplos de maior impacto ocorre na área da análise e processamento de imagens biológicas.

Nesta tese são abordados dois tópicos distintos, englobados numa vasta área de estudos científicos denominada análise e processamento automático de imagens biológicas.

O primeiro tópico de estudo debruça-se sobre a **análise e processamento automático de imagens de macroarrays (dot-blot)** para diversos sistemas fixos e móveis. O principal objetivo deste tópico consiste em fornecer uma aplicação para sistemas Java, Android e iOS, que permite detetar os microrganismos presentes numa amostra com base na análise de imagens do tipo macroarray (dot-blot). Os métodos utilizados nestas imagens para a detecção microbiana têm como base a utilização de marcadores moleculares que possuem diferentes reações fenotípicas consoante o agente patogênico em questão. Foi sobre esta característica fundamental que toda a aplicação APIdb foi desenvolvida. O classificador da aplicação APIdb mede o grau de intensidade da reação química com base na intensidade dos pixeis dos dot-blot presentes na imagem e compara essa informação com aquele que existe na base de dados, classificando desta forma todos os dot-blots presentes na imagem como sendo ON ou OFF com uma determinada probabilidade associada. Este tema surge como uma extensão do trabalho desenvolvido na FCUP [1], que incluí um protótipo (desenvolvido em Matlab) para análise e classificação de imagens dot-blot.

O segundo tópico foca-se na **análise e reconhecimento automático de espécies de plantas para o sistema móvel iOS** (APIF). O classificador do protótipo APIF utiliza um conjunto de informações relativamente à forma bidimensional do bordo da folha, nomeadamente em três tipos de descritores no seu processo de classificação: "aspecto ratio", "eccentricity"e "elongation". Este trabalho teve como suporte teórico a tese de Mestrado em Engenharia Matemática intitulada *Development of a System for Automatic Plant Species Recognition* [2]].

Abstract

The present work titled *Development of systems for automatic biological pattern recognition to several Smartphone mobile platforms* was developed in the academic year 2012/2013 in the context of the Master degree in Mathematical Engineering of the Faculty of Science of University of Porto.

Nowadays there is a massive tendency to create new applications for the wide range of existing mobile systems, due to the vast development that is taking place in the microprocessor field. These applications aim to perform the task in an automatic way, as it is common to be done by humans. One of the biggest examples is on the analysis of biological images. In this thesis two topics will be discussed, part of a major scientific studies area nominated analysis and biology image processing.

The first topic is the analysis and automatic processing of macroarray images (dot-blot) to smartphone. The main goal of this topic is to offer one application (for java, android and iOS) which will allow the identification of microorganisms present in samples by analysing dot-blot type of image. The methods used to microbial detection consist in molecular markers which have different phenotype reactions according to the microbial agent in question. The classifier of the APIdb application measures the extension of a chemical reaction based on the intensity of the pixels present in all dot-blots of an image and comparing it with those that exist in the databases, classifying, these way, all the pixels present in an image as ON or OFF according to the correspondent probability. This topic emerges as an extension of the work done at FCUP, which included one application (developed in Matlab) to analyse and classify dot-blot images. The second topic is focused on the analysis and automatic plant (APIF) species recognition for iOS mobile system. The prototype classifier APIF uses a vast range of information regarding the two-dimensional shape of the leaf edge, more specifically three descriptors; aspect ratio, eccentricity and elongation in the classification process. This work has as a theoretical support a master degree thesis titled *Development of a System for Automatic Plant Species Recognition* [2].

Acrónimos

API - Análise e Processamento de Imagens

APIdb - Análise e Processamento de Imagens dot-blot

APIF - Análise e Processamento de Imagens Folha

XML - Extensible Markup Language

JDK - Java Development Kit

MRC - Manual Reference Counting

ARC - Automatic Reference Counting

GC - Garbage Collection

PCR -Polymerase chain reaction- reacção em cadeia pela polimerase (enzima)

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Há atualmente um grande interesse no desenvolvimento de aplicações em plataformas móveis (smartphone) para análise e processamento automático de imagens. Por exemplo em biologia, a análise de imagens do tipo macroarrays (dotblot) ou a classificação\reconhecimento da espécie taxonômica de uma determinada planta é tradicionalmente realizada por técnicos especializados, sendo este processo subjetivo e demoroso. Todo o processo decorre num ambiente laboratorial com o auxílio de equipamentos sofisticados e dispendiosos não só no processo de aquisição da imagem como no processamento da mesma.

A análise de imagens do tipo macroarrays (dot-blot) são utilizadas para detetar microrganismos em amostras ambientais, como tal uma rápida e fidedigna detecções destes agentes patogênicos proporcionará inúmeras vantagens tais como: tratamentos direcionados, adoção de medidas profiláticas, monitorização dos microrganismos no ambiente,etc [1].

A análise automática de imagens do tipo folha para a classificação/reconhecimento de espécies taxonômicas de plantas é vista como uma área de enorme potencial econômico a ser explorado, não só pelo fator aliciante de qualquer indivíduo sem conhecimento prévio poder identificar taxonomicamente uma espécie de planta, mas também na área da industria e segurança alimentar na identificação de espécies venenosas de plantas [2].

Fruto desta crescente necessidade começou-se a pensar em alternativas que sejam mais portáteis e que acarretem menores custos para o utilizador, mantendo o mesmo compromisso de eficácia e exigência necessárias na avaliação e classificação destas imagens.

1.1 Solução proposta

A solução encontrada para este problema consiste na utilização de dispositivos móveis em substituição dos tradicionais computadores. Os telemóveis que utilizamos no nosso dia-a-dia são cada vez mais dotados de uma maior capacidade de processamento de dados (graças ao grande desenvolvimento que se têm verificado nos últimos anos na área dos microprocessadores) juntamente com um maior número de funcionalidades (internet, câmaras com vez maiores resoluções, suporte multimédia, etc). Foi neste nicho de novas ideias e oportunidades tecnológicas que surgiu o tema desta tese - **Desenvolvimento de Sistemas para o reconhecimento automático de padrões biológicos para as diversas plataformas móveis Smartphone**.

1.1.1 Objetivos

O principal objetivo deste trabalho foi fornecer dois protótipos de duas aplicações distintas intituladas **Analise e processamento de imagens dot-blot para diversas plataformas móveis Smartphone** e **Analise e processamento de imagens folha para o sistema iOS**.

1.1.2 Aplicações

Durante o desenvolvimento desta tese foram elaboradas as seguintes aplicações: A aplicação **Analise e Processamento de Imagens dot-blot** (APIdb) têm como principal funcionalidade permitir classificar os dot-blot (macroarray) presente na imagem como sendo ON ou OFF e qual o grau de confiança atribuída a essa classificação. Adjacente a todo este processo, existe uma base de dados que foi criada com o intuito de armazenar os resultados da classificação dos dot-blot usados durante a fase de Treino. O processo de classificação têm como base a análise da intensidade dos pixeis dos dot-blot existentes na imagem. Essa informação é depois comparada com os dados existentes na base sobre aqueles dots em particular, sendo posteriormente os dots classificados como sendo ON ou OFF.

A aplicação **Analise e Processamento de Imagens Folha** (APIF) têm como principal funcionalidade classificar a folha presente numa imagem na sua espécie taxonômica. E tal como aplicação APIdb esta usa uma base de dados no processo de classificação. A classificação taxonômica de uma folha é feita com base na extração de informações

relativas ao seu bordo. Utilizando apenas a informação sobre o bordo da folha, foi criado um protótipo que consegue discriminar um conjunto não superior do que cinco espécies de planta. Apesar das suas limitações, este protótipo poderá ser usado como plataforma no desenvolvimento de novas aplicações na análise e reconhecimento automático de especies planta.

1.2 Estrutura da tese

Os restantes capítulos desta tese são organizados da seguinte maneira:

- **Capítulo 2 Métodos de processamento de imagem** - Este capítulo visa fornecer alguns conceitos fundamentais sobre a análise e processamento de imagens digitais. Cada subcapítulo contém uma breve discussão dos seus conceitos, exemplos e resultados experimentais.
- **Capítulo 3 Processamento de imagens dot-blot** - Inicialmente é construído um modelo teórico que servirá como protocolo a ser adotado e implementado pela aplicação. Todo o procedimento é discutido aprofundadamente e acompanhado em cada etapa com resultados práticos.
- **Capítulo 4 Aplicações smartphone dot-blot** - Neste capítulo são apresentadas as três versões da aplicação **Análise e Processamento de Imagens dot-blot** para os sistemas java, androide e iOS.
- **Capítulo 5 Análise e Processamento de Imagens de Folhas para o Sistema iOS** - Inicialmente é apresentada uma breve discussão sobre o tópico em questão depois são apresentados os descritores utilizados pela aplicação no processo de classificação e por fim é apresentado uma aplicação para o sistema iOS.
- **Capítulo 7 Conclusão** - Na conclusão serão apresentados os resultados obtidos, uma síntese geral do trabalho desenvolvido ao longo desta tese e uma breve apresentação das possíveis linhas de trabalho a serem desenvolvidas no futuro.

Esta tese é suportada por uma vasta coleção de apêndices que visam fornecer os detalhes adicionais sobre a implementação das diversas aplicações desenvolvidas tanto para os sistemas java, android e iOS.

Capítulo 2

Métodos de processamento de imagem

2.1 *"Contrast Stretching"*

Algumas imagens dot-blot como é o caso da figura 2.2a sofrem de um baixo-nível de contraste entre os objetos corpo e fundo. Isto poderá ter ocorrido devido à falta de luminosidade, falta de gama dinâmica no sensor de imagem etc. Para corrigir estas deformações na imagem utilizaremos o método de contrast stretching este método visa aumentar a gama dinâmica de níveis cinza na imagem a ser processada [9], [10].

Na maior parte das imagens dot blot nem toda a gama de valores possíveis de intensidade é utilizada mas apenas uma faixa estreita de valores. Isto tem como consequência uma insuficiente discretização em determinadas regiões do espectro em prole de outras regiões aonde a discretização foi excessiva.

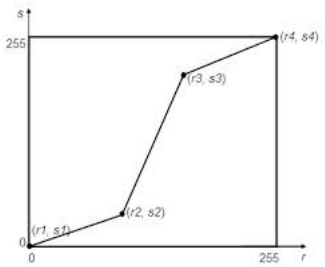


Figura 2.1: Função de transformação.

Para corrigir estas deformações na imagem é utilizada uma função de transformação de intensidade, como é ilustrado na figura 2.1, onde r representa o eixo de input do nível cinza e s representa o eixo de output do nível cinza. A região de interesse está compreendida no intervalo $[r2, r3]$, sendo $r2$ a intensidade mínima de input e $r3$ a

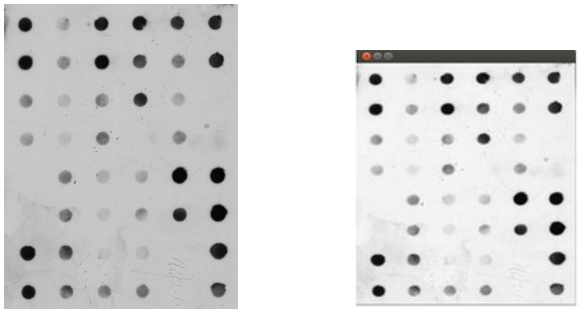
intensidade máxima de input. O resultado pretendido será de tal forma que o output na região de interesse $[s2, s3]$ (onde $s2$ representa o novo valor mínimo de intensidade e $s3$ o novo valor máximo de intensidade) tenha um intervalo de intensidade maior do que no input $|(s3 - s2)| > |(r3 - r2)|$.

A função de transformação de intensidade [11] usada foi a seguinte:

$$f(x, y) = 255 * \frac{g(x, y) - min)}{(max - min)}, \tag{2.1}$$

Onde g(x,y) representa a intensidade de um pixel com coordenas (x,y) na imagem, f(x,y) representa o novo valor de intensidade de um pixel com coordenadas (x,y) na imagem, max é o valor máximo de intensidade registado na imagem e min representa o valor mínimo de intensidade registado na imagem.

A figura 2.2 mostra um exemplo da aplicação desta transformação



(a) Imagem Original

(b) Imagem após "Contrast Streching"

Figura 2.2: Exemplo da aplicação de uma função de transformação linear.(a) Imagem antes de aplicar a transformada. (b) Imagem depois de aplicar a transformada.

Os detalhes na implementação do método "Contrast Stretching"poderão ser encontrados no anexo H

2.2 Transformação em binário

As imagens dot-blot originais (em tons de cinzento) são convertidas para formatos binários usando o método *"Thresholding"* de Otsu [9]. Em operações *"thresholding"* é necessário definir um critério que irá separar o conjunto inicial pixeis em dois conjuntos distintos. O critério de segmentação será simplesmente a intensidade de um pixel no modelo *RGB*.

$$I = \frac{(R + G + B)}{3} \tag{2.2}$$

Com o método de Otsu obtemos o limiar de segmentação τ .Um conjunto de pixeis com intensidades superiores a τ são colocados a "1"e valores inferiores a τ são colocados a "0".

As operações de linearização por "thresholding"apresentam algumas desvantagens como nomeadamente depender das condições de luminosidade e do contraste que os objetos têm com o fundo ao seu redor. Nestes casos levam a tomadas de decisões erradas por parte do algoritmo de segmentação e portanto um dos procedimentos que poderemos adotar será por exemplo normalizar a luminosidade presente na imagem, ou optar por segmentar um conjunto de sub-regiões individualmente em vez de uma segmentação global.

Independentemente no algoritmo que se utiliza para segmentar, eventualmente será necessário analisar um histograma de intensidades tal como é ilustrado na figura 2.3.

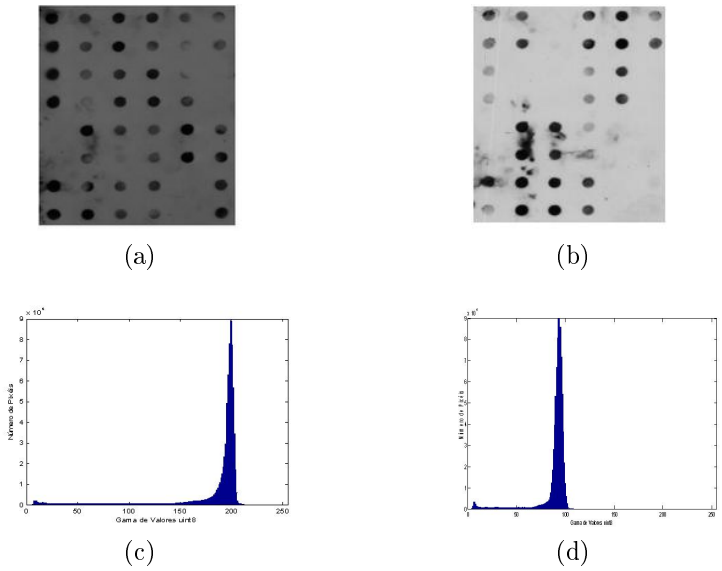


Figura 2.3: Exemplos de imagens dot-blot e os seus respetivos histogramas. (b)Imagem dot-blot com predominância de tons claros e o seu respetivo histograma de intensidade (c). (a) Imagem dot-blot com predominância de tons escuros e o seu respetivo histograma de intensidades(d).

Analisando os histogramas da figura 2.3 o valor do limiar de segmentação deverá ser escolhido de tal forma que melhor separe os pixeis pertencentes aos dots daqueles que

pertencem ao fundo da imagem.

Existem dois tipos de distribuições de intensidades que se podem encontrar nas imagens dot-blot. A figura 2.4 representa uma situação ideal com uma distribuição de intensidades bimodal, onde os dois modos ocupam regiões de intensidade distintas no espectro.

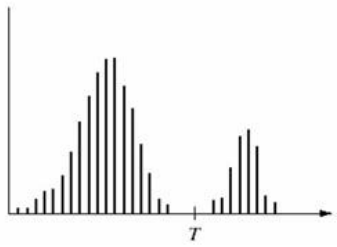


Figura 2.4: Histograma ideal para a segmentação por "thresholding".

Na prática o que acontece frequentemente é que os dois modos se intercetam, tal como é representado na figura 2.5.

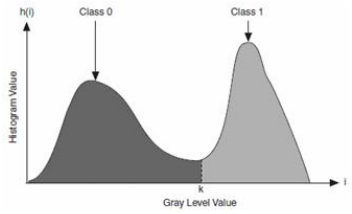


Figura 2.5: Histograma com sobreposição dos dois modos de intensidade numa determinada região do espectro.

Nestes casos é necessário alterar o critério de segmentação ou adicionar um novo critério que no seu conjunto consiga discriminar os dois tipos de pixeis presentes na imagem. Estas situações podem acontecer quando as imagens não se encontram nas melhores condições de luminosidade ou quando existe pouco contraste entre os corpos presentes na imagem e o fundo ao seu redor [9].

Aplicando o método de Otsu, com o critério de segmentação da equação 2.2 à imagem da figura 2.2a, obtemos os seguintes resultados que se encontram na figura 2.6.

Nas imagens dot-blot temos uma imagem com vários objetos dots embebidos na imagem e o próximo passo depois da segmentação será diferenciar cada dot presente na imagem como um corpo individual. Os detalhes na implementação do algoritmo de transformação binária poderão ser encontrados no anexo H.

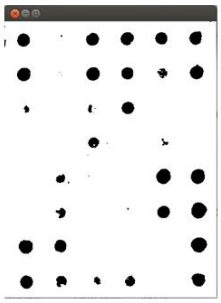


Figura 2.6: Transformação em binário- método de Otsu.

2.3 Operadores Morfológicos

As operações morfológicas são bastantes comuns no processamento de imagens e visam a extrair componentes da imagem que sejam uteis para descrever e/ou representar as características de um objeto como a sua forma, o seu esqueleto, a sua convexidade etc [10]. Em morfologia binária, uma imagem é vista como um subconjunto de um espaço euclidiano $\Re^d$ para uma dimensão d . A ideia básica destas operações morfológicas é a utilização de um elemento estrutural ou máscara. Esta máscara é uma estrutura de menor dimensão do que a dimensão da imagem e irá deslizar sobre esta mesma. Um ponto é classificado como sendo do tipo *background* ou *foreground*, dependendo do tipo de máscara e do tipo de sobreposição que existe entre a imagem e a máscara (centrada no ponto em estudo) [9]. As operações morfológicas fundamentais são erosão e dilatação. Combinando os efeitos dos operadores anteriores surgem abertura (erosão+dilatação) e fecho (dilatação+erosão)[9].

A utilização destas operações morfológicas tem algumas desvantagens:

- Os filtros não podem ser aplicados diretamente sobre a fronteira da imagem.

- Com o aumento do tamanho do filtro aumenta também os custos computacionais (filtros de grandes dimensões são ideais para eliminar o ruído na imagem e aumentar o grau de precisão nos cálculos efetuados).

2.4 Erosão

A erosão de uma imagem binária A por um elemento estrutural B num espaço euclidiano E é dado pela equação 2.3 [9].

$$A \ominus B = \{z \in E | B_z \subseteq A\}, \tag{2.3}$$

onde B_z é o conjunto formado por todos os pontos pertencentes a B que sofreram um deslocamento z, tal como é ilustrado na equação 2.4.

$$B_z = \{b + z | b \in B, \forall z \in E\}. \tag{2.4}$$

Por outras palavras a erosão de A sobre B é o conjunto formado por todos os pontos onde o elemento estrutural B se sobrepõe totalmente à imagem binária A.

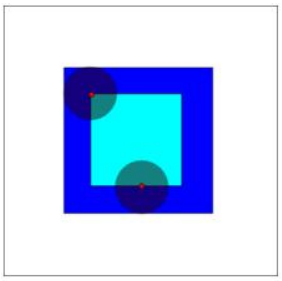


Figura 2.7: Ilustração do processo de erosão.

Na figura 2.7 encontra-se uma ilustração do processo de erosão. A imagem de input é representada pelo quadrado azul-escuro, a imagem de output é representada pelo quadrado azul-claro e o elemento estrutural tem a forma circular. O procedimento é o seguinte:

- Centrar a máscara circular no pixel em análise.
- Se todos os pixeis da máscara estiverem contidos dentro do objeto então o pixel em análise não sofre alterações, se pelo contrário, nem todos os pixeis da máscara estiverem contidos no objeto então esse pixel em análise é alterado e colocado como um pixel de fundo.

Aplicando um filtro de erosão com uma máscara quadrática 12×12 sobre a figura 2.6 obtemos o resultado da figura 2.8.

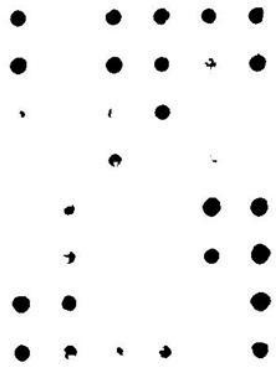


Figura 2.8: Erosão usando uma máscara 12×12.

Como podemos observar neste exemplo, todos os objetos que não possuem um diâmetro maior do que 12 pixeis são eliminados na imagem. Normalmente estes objetos de pequeno tamanho são apenas ruídos presentes na imagem. Pelo contrário, os objetos relativos a marcas ON têm grandes tamanhos e ocupam áreas extensas na imagem não sendo portanto eliminados por este processo.

Aplicando apenas este método o tamanho de todos os objetos será diminuído, pelo que é normal aplicar um novo tipo de filtro chamado dilatação para restabelecer as dimensões originais dos objetos na imagem. Os detalhes de implementação da operação erosão poderão ser encontrados no anexo H

2.5 Dilatação

O processo de dilatação será portanto o homólogo do processo de erosão [9]. Em termos matemáticas a dilatação é definida como sendo:

$$A \bigoplus B = \{z \in E | B_z \bigcap A \neq \emptyset\}, \tag{2.5}$$

onde A representa a imagem, B o elemento estruturante, z o deslocamento e $\emptyset$ o conjunto vazio. A dilatação de B sobre A é o conjunto formado por todos os deslocamentos z onde o elemento estruturante B sobrepõem uma porção mínima da imagem binária A [9].

Na figura 2.9 mostra um exemplo do processo de dilatação, a imagem de input é representada pelo quadrado azul-escuro, a imagem de output é representada pelo quadrado azul-claro e o elemento estruturante tem a forma circular.

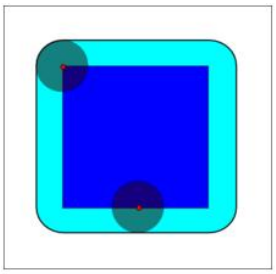


Figura 2.9: Dilatação exemplo prático

O procedimento é o seguinte:

- Centrar a máscara circular no pixel em análise
- Se alguns dos pixeis que formam a máscara estiverem contidos dentro do objeto então todos os pixeis na imagem que se encontram dentro da máscara são colocados como sendo um pixel do objeto.

Aplicando um filtro de dilatação com uma máscara quadrática 12×12 sobre a figura 2.6 obtemos o resultado da figura 2.10.

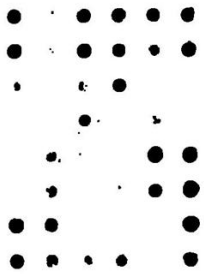


Figura 2.10: Dilatação usando uma máscara 12×12.

Como podemos constatar a dilatação aumentou o tamanho dos objetos presentes na imagem, eliminou saliências nos objetos, preencheu possíveis espaços vazios no interior dos objetos e favoreceu a forma circular dos objetos. Os detalhes de implementação da operação dilatação poderão ser encontrados no anexo H

2.6 Abertura

A operação morfológica abertura de uma imagem binária A por um elemento estruturante B denota-se como $A \circ B$,sendo simplesmente uma erosão de A por B e posteriormente dilatado por B [9], com a seguinte equação matemática:

$$A \circ B = \cup\{(B_z)|B_z \subseteq A\}. \tag{2.6}$$

Esta equação tem uma simples interpretação geométrica, $A \circ B$ é a união de todas as translações de B que encaixam inteiramente em A. O resultado da operação morfológica abertura utilizando máscaras quadráticas 12×12 sobre a figura 2.6 estão representados na figura 2.11.

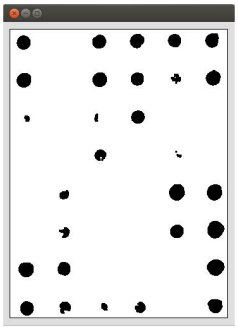


Figura 2.11: Abertura usando uma máscara 12×12

A operação abertura [10] permite remover ruído na imagem com um diâmetro inferior ao tamanho do elemento estruturante, suavizar o contorno dos objetos presentes na imagem, atenuar protuberâncias nos corpos e favorecer a forma circular dos objetos. Os detalhes na implementação da operação abertura poderão ser encontrados no anexo H

2.7 Fecho

A operação morfológica fecho de A por B denomina-se como $A \bullet B$ [10]. Trata-se de uma dilatação seguida de uma erosão, representado como:

$$A \bullet B = (A \oplus B) \ominus B. \tag{2.7}$$

Geometricamente $A \bullet B$ é o complemento da união de todas as translações de B que não se sobrepõem a A [10].

O resultado da operação morfológica fecho utilizando máscaras quadráticas 12×12 sobre a figura 2.6 está representado na figura 2.12 [10].

Tal como na operação abertura, a operação de fecho tende a suavizar o contorno dos

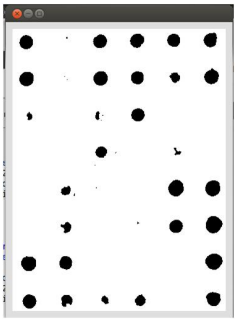


Figura 2.12: Fecho com uma máscara 12×12.

objetos, mas ao contrário desta, a operação de fecho tende a ocupar o espaço vazio existente dentro dos corpos na imagem de menor tamanho que o elemento estruturante. Tem como principal desvantagem a possibilidade de unir corpos que se encontram a distâncias inferiores ao tamanho do elemento estruturante [10].

Os detalhes na implementação da operação fecho poderão ser encontrados no anexo H.

2.8 Detecção da fronteira dos objetos

Há diversos processos de detetar a fronteira ou o contorno dos objetos numa imagem. Neste trabalho utilizou-se um tipo de filtros espaciais de Sobel, tal como é representado na figura 2.13.

-1	0	-1
-2	0	-2
-1	0	-1

G_x

-1	-2	-1
0	0	0
-1	-2	-1

G_y

Figura 2.13: Máscaras usadas no cálculo do gradiente na imagem

Utilizando os filtros de Sobel é calculado uma aproximação ao gradiente da imagem. Estes filtros de Sobel são preferenciais em relação a outros filtros tais como filtros de Prewitt [9], isto porque, este suprimem o ruído mais eficientemente embora os custos computacionais sejam os mesmos. A utilização de apenas um destes filtros resulta apenas na capacidade de detecção fronteiras retilíneas, mas a combinação dos dois filtros resulta na capacidade de detetar fronteiras também curvilíneas, apesar dos resultados não serem isotrópicos [9] [11]. Como temos uma imagem binária, a maior parte das regiões são áreas de intensidade constante e portanto o resultado da

filtragem é zero, mas aplicando o filtro em regiões na fronteira dos objetos o caso já não acontece. Foi adotado como valor limiar de segmentação, o valor máximo da magnitude do gradiente [9] [11].

Aplicando os filtros representados na figura 2.13 sobre a imagem da figura 2.6, obtemos os resultados que se encontram na figura 2.14.

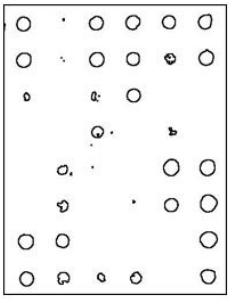


Figura 2.14: Detetar a fronteira dos objetos- utilização de filtros de Sobel 3×3.

Os detalhes na implementação deste algoritmo poderão ser encontrados no anexo H. Tendo detetado a fronteira de todos os objetos na imagem, o próximo passo consistirá em etiquetar todos os objetos presentes na imagem.

2.9 *"Labeling"*

O algoritmo de "Labeling"é utilizado para identificar todos os objetos presentes na imagem. Para tal é necessário definir a noção de objeto e vizinhança. Um objeto é um corpo único composto por um conjunto de pixeis que partilham a mesma característica em comum e se encontram interligados entre si. A noção de vizinhança surge como uma variável que terá grande impacto no desempenho da aplicação. Neste trabalho um ponto P_1 diz-se vizinho a outro ponto P_2, se o ponto P_1 se encontrar na 4-vizinhança de P_2 tal como é ilustrado na figura 2.15 [9] [11].

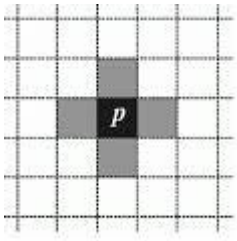


Figura 2.15: Esquema da filosofia 4-vizinhança.

Na definição de 4-vizinhança são apenas considerados como sendo vizinhos os pixeis que se encontram imediatamente na horizontal e na vertical. O algoritmo irá continuar a processar enquanto houver pixeis na imagem "por visitar". Em qualquer passo do algoritmo um determinado pixel na imagem irá sempre despoletar "a visita"aos seus vizinhos imediatos. Se os vizinhos do pixel tiverem as mesmas características deste, então serão classificados como pertencentes ao mesmo corpo, caso contrário serão classificados como sendo pixeis do fundo da imagem. Por fim, no final do processo o pixel é marcado como tendo sido "visitado"e o algoritmo passará a analisar um próximo vizinho que ainda não tenha sido "visitado"[9] [11]. Nas imagens dot-blot os marcadores ON são objetos maiores do que os do tipo ruído, com base nesta característica foi criada uma variável de controlo que contará o número de pixeis de cada objeto e caso estes não possuam as dimensões exigidas serão eliminados da imagem [9] [11]. A imagem apresentada na figura 2.16 foi obtida utilizando o algoritmo "labeling"sobre a imagem da figura 2.6, com um critério de seleção 30 pixeis por corpo [9] [11].

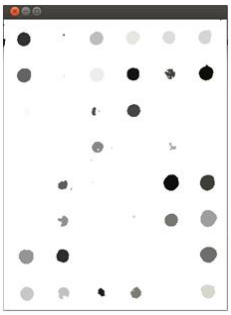


Figura 2.16: Algoritmo de identificação de corpos numa imagem dot-blot.

Os detalhes na implementação do algoritmo labeling poderão ser encontrados no anexo H.

2.10 Deteção dos centros de massa dos objetos

Tendo anteriormente utilizado o algoritmo labeling e identificados os corpos presentes na imagem, o próximo passo consiste em determinar os seus centros de massa. Com a matriz mapeamento [11] calculada anteriormente, cada objeto estará associado a um identificador único nesta matriz. Um método simples seria apenas percorrer esta matriz à procura desse identificador, e somar as coordenadas de todos os pixeis com esse identificador a dividir pelo número destes. Este método não é eficaz na procura dos

pontos que constituem um objeto, isto porque, para cada novo pixel seria necessário fazer uma nova pesquisa exaustiva na imagem para o encontrar [12]. Este método tem uma complexidade temporal:

$$Cmp_{tp} = ncorpos \times \mathcal{O}(n), \tag{2.8}$$

onde n representa o tamanho da imagem. Uma possível otimização deste método seria a utilização da informação sobre as dimensões da grelha. A grelha utilizada é do tipo 6×8 ou seja existe 48 sub-regiões dentro desta imagem e cada dot ocupará uma destas sub-regiões. Portanto podemos procurar por cada pixel de um determinado objeto dentro destas sub-regiões em vez de procurar em toda a imagem. Isto permitirá reduzir os custos computacionais deste procedimento. Na figura 2.17 estão representados os objetos e os seus respetivos centro de massa, os objetos que se encontram mais próximos dos cantos da imagem são marcados com diferentes cores (vermelho, azul, verde, amarelo) dos restantes objetos na imagem.

Figura 2.17: Deteção dos centros de massa com um critério de filtragem 30 pixéis por objeto.

O pseudocódigo da implementação deste algoritmo pode ser encontrado no anexo H. O próximo algoritmo visa a descrever como detetar os dots que se encontram mais próximos dos quatro cantos.

2.11 Deteção dos cantos numa imagem e o seu raio ótimo

Tendo determinado o centro de massa de cada objeto presente na imagem, a próxima tarefa consiste em determinar quais desses se encontram mais próximo dos quatro cantos da imagem. Para determinar quais são os objetos que se encontram mais

próximos dos quatro cantos da imagem poderemos utilizar uma lista ligada, em que cada nó desta lista contém informações acerca de um objeto particular na imagem (consultar anexo G). Percorrendo esta mesma lista e medindo distância euclidianas entre as coordenadas dos cantos da imagem e as coordenadas dos objetos, rapidamente concluímos quais são os objetos que se encontram mais próximos dos quatro cantos da imagem. A tarefa complica-se quando a imagem não possui os quatro Cantos ON (um pré-requisito essencial para o funcionamento do nossa aplicação). Quando tal acontece é crucial identificar qual é o dot em falta e a localização dos três dots ON que se encontram mais próximos dos quatro cantos da imagem. Para resolver este problema utilizaremos o conhecimento sobre o tamanho da grelha. Sabendo o tamanho da grelha é possível inferir qual é a distribuição dos dots na imagem (isto para imagens com um atenuado grau de rotação, translação e mudança de escala) e portanto se um dot está ON, então este deverá encontrar-se dentro de uma determinada sub-região na imagem, caso isto não aconteça então é porque esse dot está OFF. Determinados os centros de massa dos corpos ON que se encontram mais próximos dos quatro cantos da imagem, o próximo passo consistirá em determinar qual é o raio ótimo.

O raio ótimo é calculado tendo apenas como referência os centros de massa dos quatros dots ON que se encontram mais próximos dos cantos de uma imagem. Com os centros de massa dos dots ON e as suas respetivas fronteiras calculadas anteriormente, a medição do raio ótimo será a média das distâncias euclidianas entre um ponto na fronteira e o seu respetivo centro de massa, tal como é ilustrado na figura 2.18.

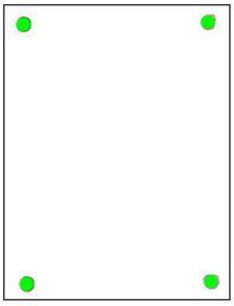


Figura 2.18: Deteção dos cantos da imagem e cálculo do raio médio.

2.12 Espaço de Cor

Os modelos de cor aparecem como métodos padrão para especificar a noção de cor. Alguns exemplos de modelos são: o modelo RGB ("Red", "Green", "Blue") fre-

quentemente utilizado em processamento de imagens digitais sendo o modelo nativo implementado em vários dispositivos móveis [9].

O modelo HSI ("hue, "saturation", "intensity") que se aproxima mais do modo como os seres humanos descreve e interpretam a cor. Quando um ser humano vê a cor de um objeto, descreve-a com os seguintes parâmetros: tonalidade, saturação e brilho. A tonalidade é um atributo de pureza da cor, a saturação é um atributo que mede o grau de diluição da cor em luz branca. O brilho é um descritor subjetivo que é praticamente impossível de medir, mas encontra-se embebido na noção acromática de intensidade. Na figura 2.19 encontra-se um esquema representativo do modelo de cor HSI [9].

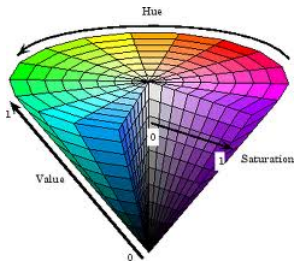


Figura 2.19: Modelo HSI.

O pseudocódigo da implementação deste algoritmo pode ser encontrado no anexo H.

2.13 Espaço de cor RGB

No modelo de cor RGB cada cor aparece nas suas componentes primárias do espectro ("Red", "Green", "Blue") [9]. Este modelo adota o sistema de coordenadas cartesiano, tal como é ilustrado na figura 2.20.

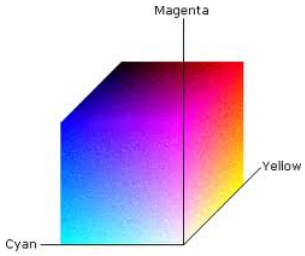


Figura 2.20: Modelo RGB.

Neste sistema a cor negra é a origem do referencial (0,0,0,0) e a cor branca é o ponto mais afastado da origem (1,1,1,1). Para projetar a imagem no monitor são combinados

os três planos RGB (cada um dos planos é uma imagem monocromática) sendo o resultado final uma imagem colorida [9].

O número de bits usados para representar cada pixel no espaço RGB é chamado de profundidade do pixel ou resolução radiométrica.

2.14 Representação numérica

Uma cor no modelo RGB pode ser descrita pela indicação da quantidade de vermelho, verde e azul que contém. Cada uma pode variar entre o mínimo (escuro) e máximo (branco). Quando todas as cores estão no mínimo, o resultado é a cor preta. Se todas estão no máximo, o resultado é a cor branca. Uma das representações mais usuais para as cores é a utilização da escala [0,255], bastante encontrada na computação pelo facto de se poder guardar cada valor de cor em 1 byte (8 bits)[13].

Alguns exemplos de cor nesta representação são:

- Branco - RGB (255,255,255);
- Azul - RGB (0,0,255);
- Vermelho - RGB (255,0,0);
- Verde - RGB (0,255,0);
- Amarelo - RGB (255,255,0);
- Magenta - RGB (255,0,255);
- Ciano - RGB (0,255,255);
- Preto - RGB (0,0,0).

Nos programas de edição de imagem, esses valores são habitualmente representados por meio de notação hexadecimal, variando de 00 (mais escuro) até FF (mais claro) para o valor de cada uma das cores. Assim, a cor #000000 é a cor escura, pois não há projeção de nenhuma das três cores; em contrapartida, $\#FFFFFF$ representa a cor branca, pois as três cores estão na sua intensidade máxima [13].

2.15 Classificação

A classificação tem como base associar a cada classe de folha um protótipo vetor padrão. Este vetor padrão é composto por um conjunto de descritores que melhor caracterizam os elementos daquela classe. Uma folha que exibe um padrão desconhecido é classificada como sendo de uma determinada espécie na qual é mais próxima em termos de uma métrica pré-definida [9]. A métrica usado neste trabalho foi a distância euclidiana (tal como é ilustrado na equação 2.9), e portanto fazendo medições entre as diferentes classes presentes na base de dados é escolhida a classe que possui uma menor distância em relação ao padrão desconhecido, de tal forma que:

$$D_j(x) = \|\vec{x} - (\vec{m_j})\|, \tag{2.9}$$

onde $D_j(x)$ é a distância euclidiana do padrão desconhecido $\vec{x}$ ao padrão $(\vec{m_j})$, $\|a\| = (a^T a)^{1/2}$. A menor distância é a que corresponde à melhor classe [9].

2.16 Descritores usados

Os descritores implementados pela aplicação *Análise e Processamento de imagens do tipo folha para o sistema iOS* são:

- "Eccentricity"
- "Aspect Ratio"
- "Elongation"

Os pseudocódigos utilizados na extração destes descritores encontram-se no anexo H.

2.16.0.1 Notação

Seja I o objeto de interesse presente na imagem binária, ∂I representa o seu bordo e $D(I)$ o seu diâmetro (a distância máxima entre quais dois pertencentes ao bordo ∂I) [2].

2.16.0.2 *"Eccentricity"*

"Eccentricity"pode ser vista como uma medida de quanto uma determinada forma se desvia da forma circular. A gama de valores possíveis é [0,1], sendo 0 uma forma circular e 1 uma forma parabólica [2].

2.16.0.3 *"AspectRatio"*

Considere qualquer dois pontos $X, Y \in \partial I$, escolhe-se X, Y de tal modo que $d(X, Y) = D(I)$. Descobri dois pontos $Z, W \in \partial I$ de tal modo que maximize $D^{\perp} = d(Z, W)$ .Num determinado conjunto ortogonal ao segmento $[X, Y]$. O "Aspect Ratio"é então definido como sendo o quociente $\frac{D(I)}{D^{\perp}}$, valores perto de 0 indicam formas alongadas [2].

2.16.0.4 *"Elongation"*

Calcula-se a distância de escape $d_{max} = max_{(x \in \partial I)} d(x, \partial I)$. A "Elongation"é obtida como $1 - \frac{2d_{max}}{D(I)}$,com uma gama de valores entre [0,1], o mínimo é atingido na região circular [2].

Capítulo 3

Processamento de imagens dot-blot

Neste capítulo é apresentada a aplicação computacional para análise automática de imagens dot-blot para as várias plataformas móveis (Smartphone).

3.1 Introdução

A deteção rápida e fidedigna de microrganismos a partir de amostras ambientais é de extrema importância em microbiologia de diagnóstico permitindo tratamentos direcionados, adoção de medidas profiláticas, estudos epidemiológicos e a monitorização de microrganismos no ambiente [1]. Nos últimos anos, os métodos moleculares de detecção de bactérias baseados na detecção de assinaturas específicas de ADN têm sido cada vez mais reconhecidas como uma alternativa promissora para contornar as limitações dos métodos clássicos de detenção [1]. Estes são dependentes do prévio isolamento do microrganismo em culturas puras e centrados na caracterização fenotípica através de estudos estruturais e bioquímicos, nalguns casos completados com análises sorológicas e/ou de patogenicidade. De facto, os métodos moleculares ao terem como alvo a detecção de assinaturas genômicas e não os organismos e as suas funções, isto é o fenótipo, permitem superar a culturabilidade, o que é particularmente pertinente para microrganismos simbióticos obrigatórios, fastidiosos ou com longos tempos de geração [1]. A especificidade e eficácia dos métodos moleculares de detecção microbiana dependem principalmente de dois fatores fundamentais:

- a seleção das assinaturas moleculares, também designados de assinaturas de ADN ou loci taxa-específicos, capazes de discriminar grupos taxonômicos (taxa) afins;

- técnicas de deteção dessas assinaturas, seja por *polymerase chain reaction*(PCR) ou por hibridação.

Ao contrário do PCR, em que apenas um ou um número muito limitado de marcadores pode ser detetado, as técnicas de hibridação, tais como *"microarrays"* e *"macroarrays"*, permitem a análise simultânea de numerosos marcadores moleculares, aumentando a robustez da deteção [1].

Os *"macroarrays"* (representado na figura 3.1), isto é hibridação utilizando como suporte uma membrana de nitrocelulose ou nylon, permite uma melhor relação custo-benefício em análises de rotina relativamente aos *"microarrays"*, em que é necessário uma nova plataforma de equipamentos para as diferentes fases de processamento, hibridação e leitura dos resultados [1].

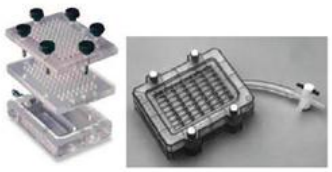
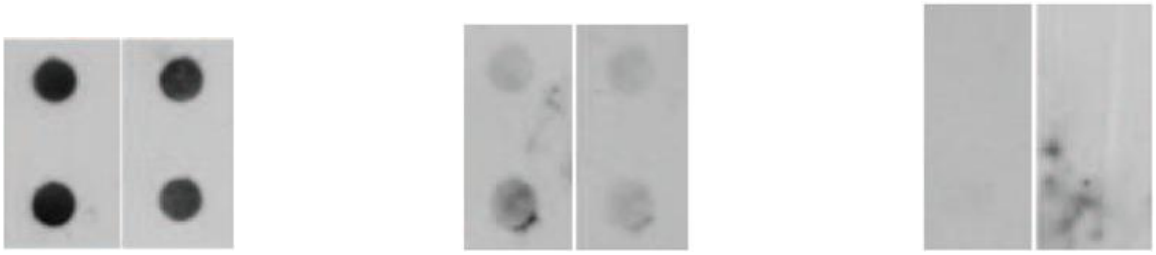


Figura 3.1: Dot blot macroarray [1].

Neste contexto, compreende-se o esforço feito na última década para desenvolvimento de macroarrays para a detecção/identificação de vários microrganismos, como demonstrado pelos exemplos já referidos, incluindo a detecção de bactérias patogênicas da batateira, *Pseudomonas* fitopatogênicas, espécies dos gêneros *Lactobacillus*, *Pythium* e de *Aeromonas*, entre outras. Atualmente uma das principais dificuldades na implementação de protocolos de detecção de microrganismos com base em macroarrays é uniformizar a análise das imagens de forma a evitar enviesamentos nos resultados causados pela interpretação do operador. Importa salientar que nos macroarrays por dotblot uma marca positiva ideal é definida por uma área escura num fundo cinzento claro e uma marca negativa ideal é indistinguível do fundo (figura 3.2). No entanto, as diferentes afinidades da hibridação entre sondas e os alvos, assim como a heterogeneidade do ruído de fundo, resultam numa imagem em níveis de cinzento a partir do qual nem sempre é fácil distinguir um resultado positivo de um negativo [1].

Existe portanto um grande interesse no desenvolvimento de um processo automático de detecção, análise e classificação de imagens dot blot a fim de permitir o processamento totalmente automático de imagens e consequentemente aumentar o potencial dos macroarrays na detecção microbiológica de rotina [1].



(a) Marcas ON (b) Marcas ON fracas (c) Marcas OFF

Figura 3.2: Exemplo de marcas das imagens dot blot [1].

3.2 Imagens de teste

Para obter os dot blots foi aplicado numa membrana de nylon 100 ng de ADN desnaturado relativo a cada marcador, utilizando um aparelho Bio-Dot (Bio-Rad, Hercules, CA, Figura 3.1). Cada dot blot é representado por uma grelha pré-definida de marcas uniformemente espaçadas, neste caso 48 marcas dispostas em 8 linhas e 6 colunas [1]. O ADN genômico de diferentes bactérias a analisar foi marcado com Digoxigenina (sondas) usando o DIG-Hight Prime Kit (Roche, Basel, Suíça). A hibridação das sondas com os marcadores alvos foi realizada durante a noite com uma temperatua 68℃, utilizando uma concentração final de 100 ng/mL de sonda por cada hibridação[2]. Sinais de hibridação positiva foram detetados por quimioluminescência utilizando filmes de raios-X (GE Healthcare). Os resultados dos dot blot foram adquiridos com um densitómetro GS-800 (Bio-Rad, Hercules, CA), produzindo imagens digitais em níveis de cinzento com uma resolução 1100 Œ 820 pixeis. A intensidade dos pixeis de fundo é variável consoante o tempo de exposição á quimioluminescência [1].

3.3 Metodologia e modelo

O algoritmo desenvolvido para a análise e detecção de marcas em imagens dot blot [1] recebe como input uma imagem digital e pressupõe o conhecimento prévio do tamanho da grelha (número de linhas e colunas) bem como da posição na imagem das marcas de controlo ON e OFF. O processo está dividido em seis etapas tal como é ilustrado na figura 3.3. As setas vermelhas indicam as transições entre as etapas no modelo e os círculos azuis a ordem dessas transição.

A sequência de etapas demarcadas com círculos azuis são:

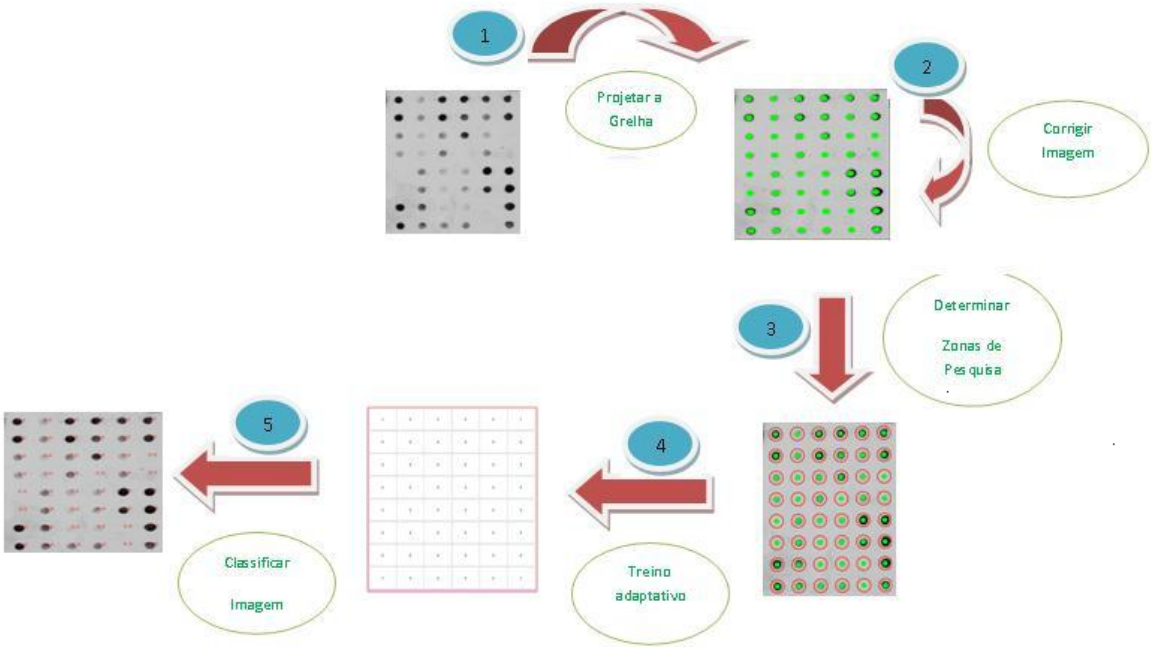


Figura 3.3: Representação esquemática do modelo utilizado.

1. Projeção da grelha pré-definida sobre a imagem;
2. Estimação orientação da grelha e correção da imagem;
3. Determinação das zonas de pesquisa;
4. Treino adaptativo utilizado para construir a função probabilidade adequada a cada tipo de marca;
5. Classificação da imagem que poderá ser: avaliar o ruído existente em torno de cada dot, classificar cada dot como sendo do tipo ON ou OFF com a sua respetiva probabilidade e estimar o grau de confiança associada a classificação de cada marca.

3.3.1 Projeção da grelha

Para detetar a grelha são necessárias várias etapas:

1. Dividir a imagem em sub-regiões (no caso das imagens de teste 6×8 dividir em 48 sub-regiões) e sobre cada uma delas operar individualmente as seguintes operações encadeadas:
   (a) Calcular o histograma de cada sub-região;

(b) Contrast stretching;

(c) Normalizar o histograma;

(d) Linearização por *thresholding*

2. A partir da imagem binária resultante é eliminado o ruído usando operadores morfológicos, neste caso uma abertura (erosão seguida de uma dilatação) com máscaras de Kernel 3×3.

3. Detecção de componentes ligadas, etiquetagem e junção de um critério de seleção 50 pixeis por corpo.

4. Calcula-se o número de objetos presentes na imagem, e os seus centros de massa.

5. A partir da matriz de mapeamento testar se os objetos nos 4 cantos estão ON.

6. Consoante o número de marcas de controlo ON presentes nos cantos da imagem o algoritmo procede de forma diferenciada:

   - Se o número de marcas de controlo ON presentes nos cantos da imagem for menor que três o processo é interrompido e gerada uma mensagem de erro ao utilizador. (Uma das condições propostas é a existência de pelo menos três cantos ON).

   - Se o número de marcas de controlo ON presentes nos cantos da imagem for três é construída uma matriz A com as coordenadas dos ON na grelha virtual segundo uma determinada convenção tal como é ilustrado na figura 3.4. Na figura 3.4 encontram-se todas as possíveis situações de falhas de marcas de controlo ON nos cantos da imagem, a ordem de inserção das coordenadas dos dots na matriz A é o dot marcado a vermelho, seguido do dot a verde e por fim o dot a azul.

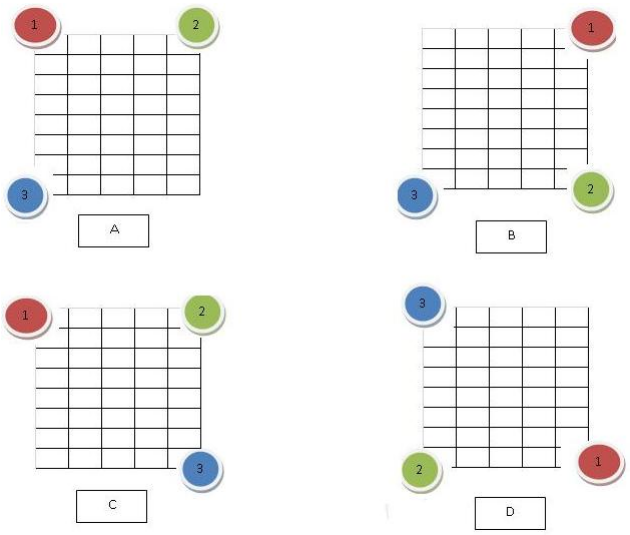


Figura 3.4: Grelha virtual com os três dots ON e a convenção utilizada para a construção da matriz A.

7. Se o número de marcas de controlo ON presentes nos cantos da imagem for quatro, é construída uma matriz A com as coordenadas dos ON na grelha virtual tal como é ilustrado na figura 3.5. A ordem de inserção das coordenadas dos dots na matriz A é o dot marcado a vermelho, seguido do dot a verde e por fim o dot a azul.

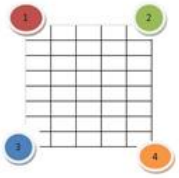


Figura 3.5: Grelha virtual com os quatro dots ON e a convenção utilizada para a construção da matriz A.

8. Construir a matriz das observações L com as coordenadas dos centros de massa das marcas de controlo ON presentes nos cantos da imagem, com a mesma convenção discutida anteriormente.

9. Utilizando o método dos mínimos quadrados e manipulações algébricas, obter os coeficientes da transformação $[x_1 \dots x_n]$ (no caso das imagens de teste utilizadas n=6) através da seguinte manipulação matricial:

$$X = (A^t A)^{-1} A^t L. \tag{3.1}$$

10. Após obter o vector com os coeficientes da transformada X, multiplicar pela matriz das coordenadas da rede R, (matriz que contém as coordenadas de todos os pontos virtuais da rede) para obter o vector com coordenadas dos dot-blot na imagem original D (equação 3.2), de tal forma que:

$$D = XR. \tag{3.2}$$

3.4 Correção da orientação da grelha

Após ter projetado os pontos da grelha sobre a imagem, a próxima fase consiste em detetar a orientação da grelha afim de corrigir possíveis anomalias ópticas na imagem.

Inicialmente são calculados todos os ângulos entre pares de marcas. Como as duas direções principais da grelha são ortogonais, os ângulos calculados são reduzidos ao domínio $[0^0, 90^0]$. Assim, para n marcas numa imagem é necessário calcular $\frac{(n(n-1))}{2}$ direções distintas [1].

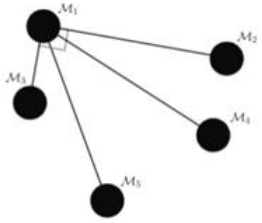


Figura 3.6: Estimação da orientação inicial da grelha para as marcas M_1, M_2, M_3, M_4, M_5 [1].

A figura 3.6 estão assinaladas as 4 direções associadas à marca M_1. Neste caso é necessário calcular 10 direções distintas, sendo as direções $\overline{M_1M_2}$ e $\overline{M_1M_3}$ idênticas, uma vez que são ortogonais entre si.

A orientação da grelha é obtida pelo ângulo mais frequente (α) entre as direções calculados anteriormente, com um incremento de 2°, o que corresponde a um erro absoluto máximo de 2°. Finalmente a imagem é rodada de $-\alpha^\circ$, obtendo-se uma nova versão da imagem original com a grelha de marcas aproximadamente alinhada com a imagem [1].

3.5 Determinação das zonas de pesquisa

Uma parte crucial do projeto consiste em determinar as regiões de pesquisa na imagem. Isto porque, para calcular a probabilidade de uma marca estar ON ou para avaliar o ruído de uma marca, é necessário retirar uma amostra dos pixeis pertencentes ao objeto do dot e do fundo ao seu redor. Para tal é necessário dividir o conjunto inicial de pixeis e determinar os limites do que é considerado uma zona interna, (objeto dot), uma zona intermediária (onde existe uma grande incerteza sobre a natureza dos pixeis) e uma zona externa (pixeis do fundo).

Para determinar a zona interna é calculado o raio interno com base na informação da fronteira dos dot-blot presentes nos cantos da imagem. Para determinar a zona externa é calculado o raio externo como sendo um terço da distância média entre cada par de dot-blots presentes na imagem.

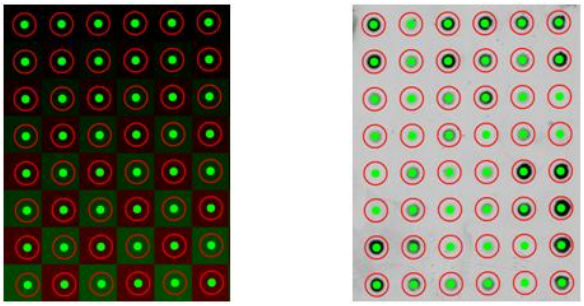
Como todo este processo é sensível ao ruído e também pelo facto que nem todos os dots terem o mesmo raio, é necessário criar uma região intermediária onde existe uma elevada incerteza em relação à natureza dos pixeis. Através de vários testes foi estipulado um parâmetro δ, de acordo com a equação 3.3.

$$\delta = \frac{r_{int}}{r_{int} + r_{ext}} \times r_{ext} \tag{3.3}$$

As regiões são divididas da seguinte forma:

$$\begin{cases} \text{zona interna} & \text{if} \quad r < r_{int} - \delta; \\ \text{zona intermédia} & \text{if} \quad r_{ext} - \delta \leq r \leq r_{int} - \delta; \\ \text{zona externa} & \text{if} \quad r_{ext} > r \geq r_{ext} - \delta. \end{cases} \tag{3.4}$$

As zonas de pesquisa encontram-se ilustradas na figura 3.7.



(a) (b)

Figura 3.7: Determinação das regiões de pesquisa : (a) Imagem dividida em 6×8 sub-regiões . (b) Zonas de pesquisa: zonas internas encontram-se representadas a verde e as externas a vermelho.

3.6 Identificação final das marcas

Nas imagens dot-blot cada marca τ pertence a um determinado tipo T_τ com uma assinatura de ADN específico. As marcas de controlo são representadas por 0 e 1 ($T_{OFF} = 0$, $T_{ON} = 1$) e as restantes marcas com valores inteiros maiores do que um $(2, \ldots, n)$. Um exemplo de uma grelha que pode ser utilizada na fase de treino ou na classificação encontra-se representada na tabela 3.1

	C1	C2	C3	C4	C5	C6
L1	1	2	3	4	5	1
L2	1	2	3	4	5	1
L3	6	7	8	9	10	0
L4	6	7	8	9	10	0
L5	0	11	12	13	14	15
L6	0	11	12	13	14	15
L7	1	16	17	18	0	1
L8	1	16	17	18	0	1

Tabela 3.1: Grelha virtual com marcas de diferentes tipos. Onde C_j indica a coluna j da grelha e L_i indica a linha i da grelha.

Neste exemplo a grelha possui 6 marcas de controlo OFF, 8 marcas de controlo ON e 2 réplicas de cada um dos restantes tipos ($T_\tau : \tau = 2, \ldots, 18$). Os controlos ON estão situados nos cantos das imagens, nas posições (1, 1), (2, 1), (1, 6), (2, 6), (7, 1), (8, 1), (7, 6) e (8, 6), os controlos OFF estão situados nas posições (5, 1), (6, 1), (3, 6), (4, 6), (7, 5) e (8, 5) e nas restantes posições encontram-se marcas regulares (marcas

que não são de controlo ON ou OFF).

Esta estratégia de dispor diferentes tipos de marcas na mesma placa experimental não é um procedimento sempre adotado, sendo frequentemente mais vantajoso considerar todas as como sendo do mesmo tipo.

3.7 Determinação da probabilidade de uma marca estar ON

Normalmente os controlos ON são marcas com assinaturas bem definidas, enquanto os controlo OFF são praticamente indistinguíveis do fundo da imagem. Mas nem sempre é fácil distinguir uma marca ON de um OFF, isto leva á necessidade de elaborar uma fórmula de cálculo para função de densidade de probabilidade de uma determinada marca ser ON ou OFF [1].

Para tal é calculada a média da intensidade dos pixeis dentro de cada marca I_τ e a média da intensidade dos pixeis no fundo local a cada marca I_{fundo}. A intensidade normalizada para cada marca($\overline{I_\tau}$) é calculada pela equação 3.5 [1]

$$\overline{I_\tau} = \begin{cases} 1 - \frac{I_\tau}{I_{fundo}}, & \text{if } I_\tau \leq I_{fundo}; \\ 0, & \text{if } I_\tau > I_{fundo}. \end{cases} \tag{3.5}$$

Para as marcas regulares é calculada a média das intensidades normalizadas $\overline{\mu_{I_\tau}}$ para cada tipo de assinatura e comparado com a média das intensidades normalizadas para as marcas de controlo OFF $\overline{\mu_{OFF}}$ [1].

Pela análise experimental dos valores $\overline{\mu_{I_\tau}}$ e $\overline{\mu_{OFF}}$ para diferentes tipos de marcas, observou-se que a diferença entre estes dois valores $\Delta = \overline{\mu_{I_\tau}} - \overline{\mu_{OFF}}$ deve ser um valor positivo compreendido entre 0.05 e 0.10. A função de densidade de probabilidade foi estabelecida como sendo:

$$P(\overline{I_\tau}) = \begin{cases} 0, & \text{if } \overline{I_\tau} < \overline{\mu_{OFF}}; \\ \frac{\arctan(u) - \arctan(-2)}{\arctan(8) - \arctan(-2}, & \text{if } \overline{\mu_{OFF}} \leq \overline{I_\tau} \leq \overline{\mu_{I_\tau}}; \\ 1 & \text{if} \overline{I_\tau} > \overline{\mu_{I_\tau}}. \end{cases} \tag{3.6}$$

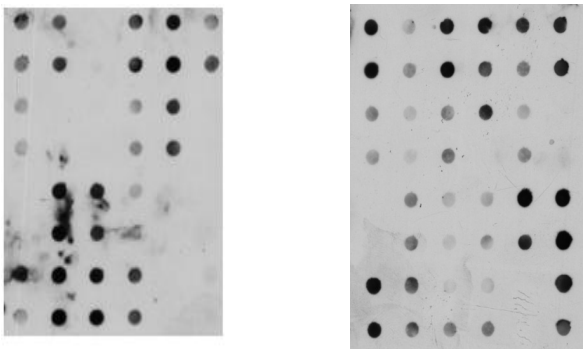
onde:

$$u = -2 + \frac{10 \times (\overline{I_\tau} - \overline{\mu_{OFF}})}{\Delta}.$$

Para valores de intensidade normalizada inferiores a $\overline{\mu_{OFF}}$ a probabilidade da marca estar ON é 0, para valores superiores a $\overline{\mu_{I_\tau}}$, a probabilidade da marca ser ON é 1 e para valores de intensidade normalizada entre $[\overline{\mu_{OFF}} \quad , \quad \overline{\mu_{I_\tau}}]$, o valor da probabilidade está compreendido entre $]0, 1[$. Durante a fase de treino adaptativo são utilizadas algumas imagens (no mínimo uma) para construir a função de probabilidade $P(\overline{I_\tau})$ dado pela equação H.1 adaptada a cada tipo de marca [1].

3.8 Avaliação do ruído

As imagens do tipo dot blot são frequentemente afetadas por ruído que podem corromper o processo de classificação das marcas. Na figura 3.8 são apresentadas duas imagens contrastantes: a imagem da figura 3.8a contagiada com ruído e a imagem da figura 3.8b quase sem a presença de ruído [1].



(a) (b)

Figura 3.8: Duas imagens usadas na classificação do nível de ruído: (a) Imagem contagiada com ruído. (b) Imagem quase sem a presença de ruído.

Para calcular o nível de ruído em torno de cada marca τ, é utilizada a informação dos pixeis na sua vizinhança. A área em volta da marca (definida num anel exterior) é dividida em oito sectores iguais, como é ilustrado na figura 3.9 [1].

Foi utilizado como estimador do nível de ruído a diferença entre a média da intensidade dos pixeis dentro de cada sector e a média da intensidade dos pixeis de fundo. Cada sector pode ser classificado como tendo um nível de ruido (Baixo, Médio, Alto). Quando Δ_{ruido} é inferior a 30, o sector tem o nível de ruído baixo sendo-lhe atribuído

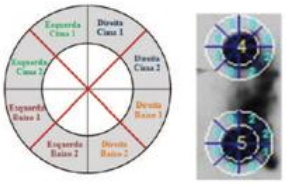


Figura 3.9: Processo de avaliação do nível de ruido com base em sectores e exemplos práticos [1].

um indicador de ruído igual 0. Com Δ_{ruido} compreendido entre $]30, 100]$ é atribuído um nível de ruído médio com um indicador igual a 1. Para valores superiores a 100 é atribuído um nível de ruído alto com um indicador igual a 2 [1], tal como é ilustrado na figura 3.10.

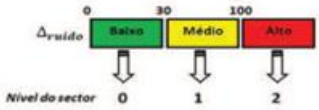


Figura 3.10: Critério de classificação do nível de ruído por sectores [1].

Para calcular o ruído total de cada marca N_{rudo} é utilizada a equação 3.7

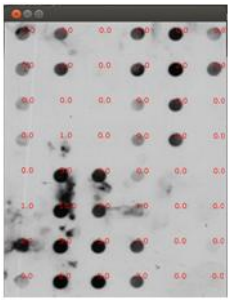
$$N_{ruido} = \frac{\sum_{j=1}^{8} ind_j}{8}, \tag{3.7}$$

onde ind_j é o indicador do nível de ruido no sector j. Se o nível de ruído for inferior a 3 a marca é classificada como tendo um nível de ruído baixo. Para um nível de ruído compreendido entre] 3,10] a marca é classificada como tendo um ruído médio. Para níveis de ruído superiores a 10 é classificada como tendo um nível de ruído alto, tal como é ilustrado na figura 3.11 [1].

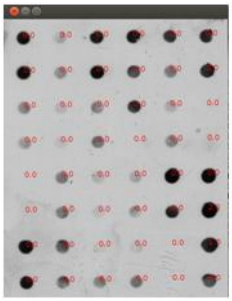
Figura 3.11: Critério para a classificação do nível de ruido por marca [1].

Nas figuras 3.12a e 3.12b estão representados os resultados da avaliação do nível de ruído em dois casos distintos:

Todo o processo de avaliação de ruído depende de uma correta projeção da grelha,



(a) Imagem com ruído



(b) Imagem quase sem ruído

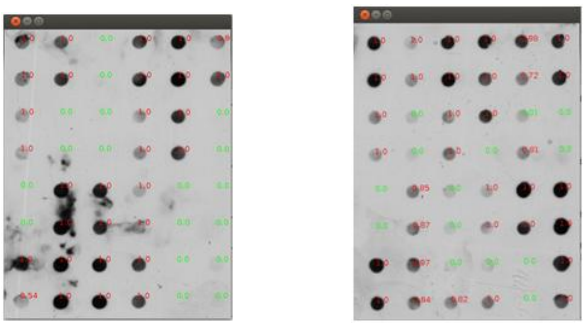
Figura 3.12: - Exemplos de avaliação do nível de ruído: em (a) uma imagem contagiada com ruído. (b) Uma imagem praticamente sem ruído.

caso isto não aconteça os centros dos dots serão incorretamente projetados, dando origem a resultados inválidos.

3.9 Classificação

A classificação é feita assumindo à priori que pelo menos uma imagem foi usada para treinar o classificador. Cada marca na imagem é classificada como sendo um dot ON ou OFF com determinado valor de probabilidade associado. A probabilidade de uma marca ser ON é calculada pela equação H.1, usando os valores de $\overline{\mu_{I_\tau}}$ e $\overline{\mu_{OFF}}$ obtidos durante a fase de treino adaptativo para cada tipo de marca. Se o valor de $P(I_\tau)$ é inferior a 0.5, a marca é rotulada de OFF, com a probabilidade de $1 - P(I_\tau)$. Caso contrário a marca é rotulada de ON com probabilidade $P(I_\tau)$.

Nas figuras 3.13a e 3.13b estão representadas as probabilidades 48 marcas estarem ON [1].



(a) Imagem com ruído

(b) Imagem quase sem ruído

Figura 3.13: - Exemplos de classificação de imagens dot-blot: em (a) uma imagem contagiada com ruído e em (b) uma imagem praticamente sem ruído.

Na figura 3.13 encontram-se representadas a verde as marcas que foram classificadas como sendo OFF (com a sua respetiva probabilidade de estarem ON), a vermelho estão representadas as marcas que foram classificadas como sendo ON e a sua respetiva probabilidade.

3.10 Estimativa do grau de confiança

Para produzir uma estimativa global de confiança são necessários três parâmetros fundamentais: um parâmetro de confiança no treino, um no ruído e outro na classificação. Todos estes parâmetros estão definidos num intervalo de 0 a 1, correspondendo 1 ao valor ótimo [1]. O parâmetro de confiança no treino PC_t foi estabelecido para cada

tipo de marca, como sendo a média das probabilidades de todos as marcas daquele tipo usadas na fase de treino. O parâmetro de confiança no ruído PC_n foi obtido a partir do nível de ruído de cada marca. Níveis de ruído iguais a 0 ou 1 correspondem a $PC_n = 1$. Níveis de ruído maiores do que 10 correspondem a $PC_n = 0$. Para níveis de ruído compreendidos entre [2,10] foi utilizada uma interpolação linear dada pela equação 3.8 [1].

$$PC_n = \begin{cases} 1 & \text{if} \quad N_{rudo} = \{0, 1\}; \\ \frac{-N_{rudo}+10}{8} & \text{if} \quad N_{rudo} = \{2, \ldots, 10\}; \\ 0 & \text{if} \quad N_{rudo} = \{11, \ldots, 16\}. \end{cases} \tag{3.8}$$

O parâmetro de confiança na classificação é dado pela equação 3.9 [1]

$$PC_c = \begin{cases} 2P(\overline{I_\tau}) - 1 & \text{if} \quad P(\overline{I_\tau}) \geq 0.5; \\ 1 - 2P(\overline{I_\tau}) & \text{if} \quad P(\overline{I_\tau}) < 0.5. \end{cases} \tag{3.9}$$

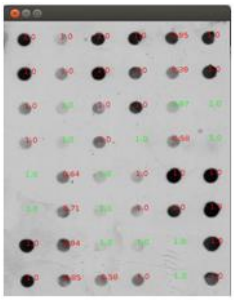
A primeira parte da equação 3.9 refere-se ao grau de confiança atribuída na classificação de uma marca como sendo ON e a segunda parte da equação 3.9 refere-se ao grau de confiança atribuída na classificação de uma marca como sendo um OFF.

A estimativa do grau de confiança total E_c é dado pela equação 3.10

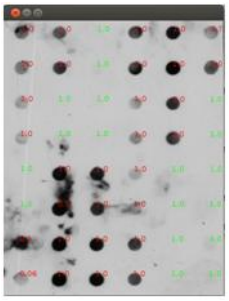
$$E_c = PC_c \times \frac{(PC_n + PC_t)}{2}. \tag{3.10}$$

Para as marcas de controlo (ON e OFF) o parâmetro global de estimação de confiança é obtido pela equação $E_c = PC_c \times PC_n$ uma vez que não há nenhum treino associado a estas marcas.

Na figura 3.14 encontram-se representados os graus de confiança atribuídos na classificação das imagens da figura 3.13.



(a) Imagem com ruído



(b) Imagem quase sem ruído

Figura 3.14: - Exemplos de estimativas do grau de confiança: em (a) uma imagem contagiada com ruído e em (b) uma imagem praticamente sem ruído.

Podemos observar que no caso de imagens contagiadas com ruido (como é o caso da figura 3.14b), estas possuem estimativas de confiança mais baixas do que imagens praticamente ausentes de ruído (como seja o caso da imagem 3.14a).

Capítulo 4

Aplicações smartphone dot-blot

Este capítulo descreve o desenvolvimento de 3 aplicações distintas para os sistemas java, android e iOS com o objetivo de analisar e processar imagens dot-blot. Os conceitos fundamentais sobre os sistema android e iOS encontram-se no anexo A

4.1 Ambiente java

A interface gráfica ilustrada na figura 4.1, consiste num painel de dimensões 300 × 300 pixéis, constituído por duas ComboBox. Uma Combox para selecionar uma determinada imagem referenciada estaticamente em **Abrir Imagem**, a outra para selecionar o tipo de operação que se deseja realizar sobre a imagem em **Escolha uma Opção**. Existem ainda outros quatros botões dentro deste painel, um para **Criar uma Imagem**, um outro para abrir dinamicamente uma imagem no disco em **Abrir Imagem**, um terceiro botão para guardar os resultados experimentais numa Base de Dados em **Guardar na Base de Dados** e por fim um quarto botão que permite guardar a imagem manipulada no disco em **Guardar Imagem**.

Figura 4.1: Interface gráfica da aplicação em java.

Funcionamento da aplicação em java

Para abrir uma foto que esteja armazenada no disco rígido do computador o utilizador poderá escolher uma foto que se encontra referenciada na ComboBox ou pressionar o botão **Abrir Imagem** (abrindo deste modo uma janela de pesquisa típico do Ubunto) tal como é ilustrado na figura 4.2.

Figura 4.2: Abrir uma imagem em java.

A aplicação possui suporte para abrir imagens com extensão jpg e png. Se o utilizador desejar processar a imagem deverá escolher uma das opções que se encontram disponíveis na comboBox **Escolha uma opção** tal como é ilustrado na figura 4.3.

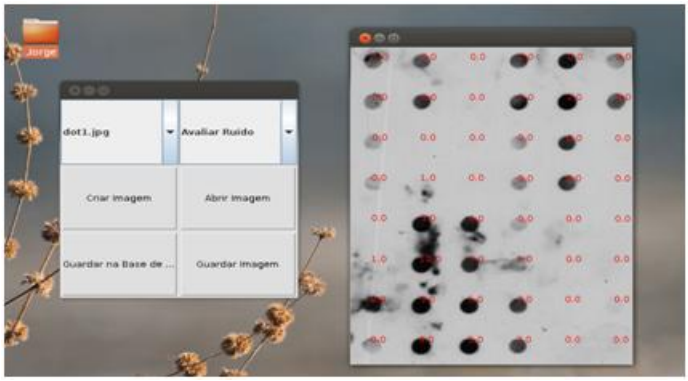


Figura 4.3: Processar uma imagem em java.

4.2 Ambiente android

A aplicação **Análise e processamento de imagens para o sistema Android** é constituída fundamentalmente por duas "Activity's":

- MainActivity;
- ShowImage.

Um esquema de funcionamento da aplicação encontra-se ilustrado na figura 4.4, onde as transições possíveis entre cada "activity"encontram-se demarcadas com setas vermelhas.

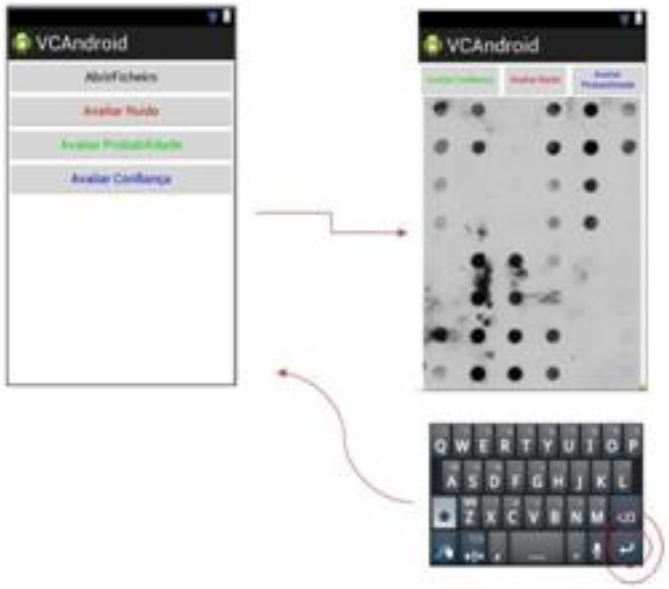


Figura 4.4: Esquema de funcionamento da aplicação dot-blot para o sistema android.

4.2.1 MainActivity

A MainActivity ilustrada na figura 4.5 é composta (na parte central do monitor do telemóvel) por uma ImageView onde o utilizador poderá observar em tempo real todas as imagens que estão sendo captadas pela câmara do dispositivo (tal como uma simples aplicação vídeo). Na parte superior do monitor é possível observar quatro tipos de botões diferentes:

- Abrir Ficheiro - O utilizador inicializa a galeria de foto do dispositivo podendo desta forma escolher uma imagem que fora previamente armazenada no disco, quando o utilizador tiver selecionado a foto o sistema transitará para ShowImage "activity".
- Avaliar Ruído - O utilizador ao pressionar o botão **Avaliar Ruido** é tirada uma foto com a câmara do dispositivo, sendo posteriormente avaliado o ruído

em torno de cada marca na imagem. O ruído é classificado numa determinada escala apresentada na secção 3.8.

- Avaliar Probabilidade - Ao pressionar o botão **Avaliar probabilidade** é adquirida uma foto com a câmara do dispositivo, sendo posteriormente inferido qual o grau de probabilidade de uma determinada marca corresponder a um ON (desde que este tipo de marca esteja presente no treino adaptativo).

- Avaliar Confiança - Carregando no botão **Avaliar Confiança** é processada uma foto com a câmara do dispositivo e o utilizador poderá saber qual é o grau de confiança na classificação de um determinado dot

Figura 4.5: Interface Gráfica MainActivity

4.2.2 ShowImage

Quando o utilizador tiver selecionado uma determinada imagem da sua galeria de imagens o sistema transita para a "ShowImage activity", ilustrada na figura 4.6. Ao iniciar esta "activity"é mostrada na parte principal do monitor a foto que foi anteriormente escolhida. Esta "activity"é composta, tal como a anterior (MainActivty), pelo conjunto de botões **Avaliar Ruído**, **Avaliar Confiança** e **Avaliar Probabilidade**. Estes botões possuem as mesmas funcionalidades daqueles que foram discutidos na secção 4.2.1.

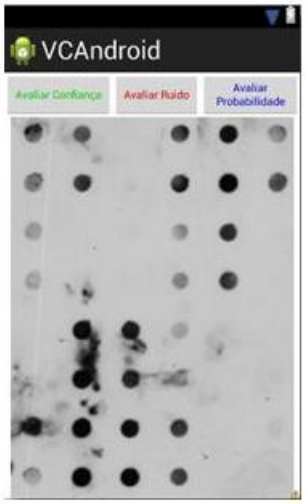


Figura 4.6: Interface Gráfica ShowImage.

Estrutura do projeto android

Na figura 4.7 encontra-se uma representação da estrutura do projeto dot-blot no IDE Eclipse.

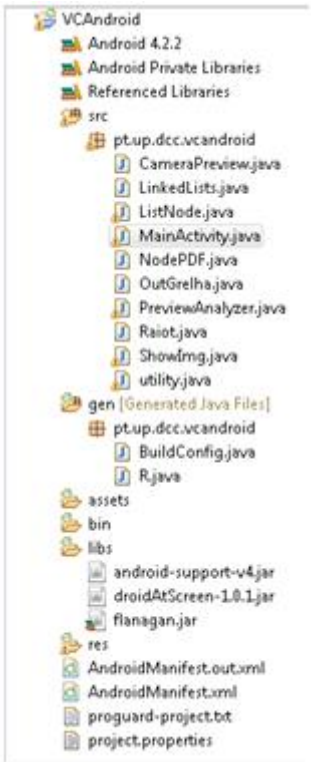


Figura 4.7: Estrutura do projeto dot-blot para o sistema Android.

O projeto contém um diretório **src** onde se encontram as principais classes da aplicação:

- MainActivity.java - Classe principal do projeto sendo a primeira "activity"a ser invocada no arranque da aplicação.
- PreviewAnalyzer.java - Classe responsável pelo processamento de imagens oriundas da câmara do dispositivo.
- CameraPreview.java - Responsável por mostrar as imagens vindas da câmara.
- ShowImg.java - "Activity"responsável pelo processamento de imagens previamente armazenadas em disco.
- RaioT.java - Classe que representa o raio Interno e Externo dos dot-blot.
- OutGrelha.java - Classe que representa a grelha virtual.
- ListNode.java - Tipo de nó que armazena informações relativos a um dot-blot.
- LinkedList.java - Classe que representa a lista ligada implementada neste projeto.
- utility.java - Classe que contém um conjunto de métodos uteis para o processamento de imagens tais como Labeling, Otsu, Contrast-Streching, etc.

Dentro do diretório **gen** encontram-se as classes de configuração geradas automaticamente pelo android tais como a classe R.java aonde são armazenados os ID de todos os recursos do projeto. No diretório **libs** encontram-se as referências a todas as bibliotecas externas utilizadas pela aplicação, em particular a libraria flanagan que foi usada no cálculo matricial. No diretório **res**, tal como é representado na figura 4.8, encontram-se todos os recursos utilizados pela aplicação como por exemplo: ícones de procura e arranque ou imagens estáticas utilizadas para carregar inicialmente alguma ImageView na aplicação.

Figura 4.8: Directório res do projeto dot-blot para o sistema android.

Um subdiretório de extrema importância é **res/layout**, neste subdiretório encontram-se os ficheiros xml que são utilizados para a construção/criação das "views"e todos as "children"que estas possam conter. Um outro subdiretório importante é **res/value**, onde são definidos alguns parâmetros essências para a construção da interface, tais como os títulos de "Views"e "Buttons"que são definidos no ficheiro strings.xml. O tipo de cor é definido no ficheiro colors.xml, etc. No ficheiro AndroidManifest.xml são declaradas todas as "activity"usadas na aplicação.

4.2.2.1 MainActivity()

MainActivity() é a principal "activity"da aplicação dot-blot sendo também a primeira a ser invocada pelo sistema operativo. Dentro desta existe o método **OnCreate()** que será a primeira função a ser invocada. Um exemplo do código do método é ilustrado na figura 4.9.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
    setContentView(R.layout.camera_layout);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
    setButtonClickListener();
}
```

Figura 4.9: Exemplo de código do método onCreate().

Dentro deste método podem ser encontrados as seguinte chamadas a outros métodos:

- SetContentView(int) - Preenche o conteúdo da "activity's"ecrã com um determinado Layout resources dado como input [3].
- SetButtonClickListener(void) - Quais os objetos que serão observados para ações do tipo click [3].
- onPause()- Este método é invocado quando o utilizador deixar a corrente "activity"[3].
- setRequestedOrientation(int requestedOrientation) - Muda a orientação atual da corrente "activity"para uma determinada orientação dada como parâmetro de entrada [3].
- RequestWindowFeature(window.FeatureNoTitle) - esconder a barra de status [3].

Por fim todos os botões são instanciados na interface são associados um "Listener", que ficará à espera de eventos de click que possam ser despoletados por parte do utilizador. Dois exemplos são o botão **buttonabrir** e o botão **buttonStart** representados na figura 4.10 .

```
Button buttonabrir = (Button) findViewById(R.id.buttonabrir);
buttonabrir.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setType("image/*");
        intent.setAction(Intent.ACTION_GET_CONTENT);
        startActivityForResult(
                Intent.createChooser(intent, "Select Picture"),
                SELECT_PICTURE);

    }
});
```

Figura 4.10: buttonabrir - Inicializa uma nova "activity"*Select Picture*.

O utilizador ao carregar no **buttonabrir** irá inicializar uma nova Activity onde poderá escolher uma imagem que fora previamente guardada no disco, todo este processo é inicializado com a troca de mensagem assíncronas do tipo "Intents"(com este serviço é possível requisitar funcionalidades de componentes do android [18]).

Ao pressionar o botão *buttonStart* o sistema inicializa a câmara do android, sendo posteriormente a foto adquirida e processada. Um exemplo de código que poderá ser usado no processo de aquisição da foto encontra-se no anexo B. A estratégia após o processar a imagem passa por desenhar ou redesenhar no ecrã os resultados experimentais [3]. Para desenhar no ecrã do dispositivo são necessárias quatro componentes básicas:

- Um BitMap para armazenar os pixéis.
- Uma classe Canvas para desenhar objetos gráficos no ecrã do dispositivo.

```
Button buttonStart = (Button) findViewById(R.id.buttonStart);
buttonStart.setOnClickListener(new OnClickListener() {

    public void onClick(View arg0) {
        mCamera.takePicture(new ShutterCallback() {
            public void onShutter() {
            }
        }, new PictureCallback() {

            public void onPictureTaken(byte[] data, Camera camera) {
            }
        }, new PictureCallback() {

            public void onPictureTaken(byte[] data, Camera camera) {

                canvas = mHolder.lockCanvas(null);

                sizex = mPreview.getPreviewSizeX();
                sizey = mPreview.getPreviewSizeY();

                processFrame(data, canvas);
                // fazer coisas
                mHolder.unlockCanvasAndPost(canvas);
                canvas = null;

            }
        });
```

Figura 4.11: buttonStart.

- Primitivas de desenho (Rect, Path, texto, etc).
- Uma biblioteca gráfica chamada Paint (para descrever cores e estilos de desenho).

Dois dos métodos disponíveis na classe Canvas com grande interesse são lockCanvas(Rect dirty) e unlockCanvasAndPost(). No início do processo com a chamada do método lockCanvas(Rect dirty) é reservada/obtida uma área de desenho no ecrã do android, seguidamente a imagem é processada levando a possíveis alterações no bitmap. No final do processo é chamado o método unlockCanvasAndPost() para redesenhar (na área reservada) os pixéis que foram alterados no bitmap e libertada a posse da área do monitor [3].

4.3 Ambiente iOS

A estrutura da aplicação *Análise e Processamento de Imagens dot-blot para o sistema iOS* (ilustrada na figura 4.12) é composta pelo seguinte conjunto de diretórios:

- Diretório FD
- Framework
- Products

Dentro do diretório **FD** podemos encontrar o seguinte conjunto de ficheiros:

- AppDelegated.m e AppDelegated.h - Responsáveis pelo ciclo de vida da aplicação.

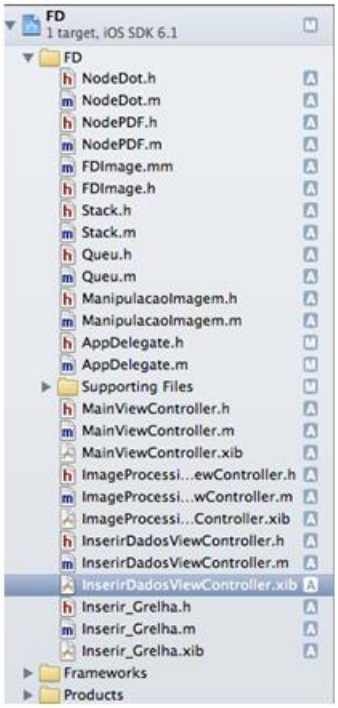


Figura 4.12: Esquema Geral da Aplicação Dot-blot no sistema iOS.

- Ficheiros Stack.m e Stack.h - Subclasse da classe NSArray utilizada frequentemente durante o código da aplicação como classe de apoio [19].
- Ficheiros Queu.m e Queu.h - Subclasse da classe NSArray utilizada frequentemente durante o código da aplicação como classe de apoio [19].
- FDImage.mm e FDImage.h - extensão .mm indica ao compilador que para compilar o ficheiro FolhasProcessing.mm como um ficheiro C. Estes ficheiros contêm o núcleo de toda a aplicação e todas as funções, estruturas utilizadas para o processamento da imagem.
- ViewController.m e ViewController.h - Responsáveis por controlar o conteúdo da user Interface.
- Subdiretório Supporting File - Contém um conjunto de ficheiros responsáveis pela interação com o utilizador.

Dentro do subdiretório **Supporting File** podemos encontrar o seguinte conjunto de ficheiros:

- MainViewController.m, MainViewController.h e MainViewController.xib - Ficheiros de controle da "view"MainView da aplicação.

- Main.m - função que inicia a execução da aplicação iOS.
- MyDataBase.txt- Ficheiro onde será guardado os resultados treino.
- ImageProcessingViewController.h, ImageProcessingViewController.m e ImageProcessingViewController.xib - Ficheiros de controle da "view"ImageProcessingView da aplicação. Esta "view"contém no seu centro um painel principal com a imagem dot-blot a ser processada, três botões na parte superior da "view"para processamento (**Avaliar Ruido, Avaliar Probabilidade e Avaliar Confiança**) e dois botões na parte inferior da "view"para **Guardar Treino** (guardar os resultados experimentais num ficheiro txt) e **Retroceder** (voltar para uma "view"anterior).
- Inserir_Grelha.m, Inserir_Grelha.h e Inserir_Grelha.xib - Ficheiros de controle da "view"InserirGrelhaView da aplicação, nesta "view"o utilizador irá inserir o nome dos marcadores numa grelha virtual.
- InserirDadosViewController.m,InserirDadosViewController.h e InserirDadosViewController.xib - Ficheiros de controle da "view"InserirDadosView da aplicação, nesta "view"o utilizar irá preencher um formulário com informações sobre as dimensões da grelha e a sua composição.
- FotoViewController.m, FotoViewController.h e FotoViewController.xib - Ficheiros de controle da "view"FotoView da aplicação, nesta "view"o utilizador poderá escolher entre processar uma foto que foi previamente armazenada no disco ou tirar uma foto utilizando a câmara do iPad/ iPhone.

4.3.1 Panorama da aplicação

A aplicação Dot-Blot para o Sistema iOS é composta por cinco "views":

- MainView;
- InserirDadosView;
- InserirGrelhaView;
- FotoView;
- ImageProcessingView.

Na figura 4.13 encontra-se representado um esquema do funcionamento da aplicação dot-blot para o sistema iOS (as transições possíveis entre "views"estão representadas

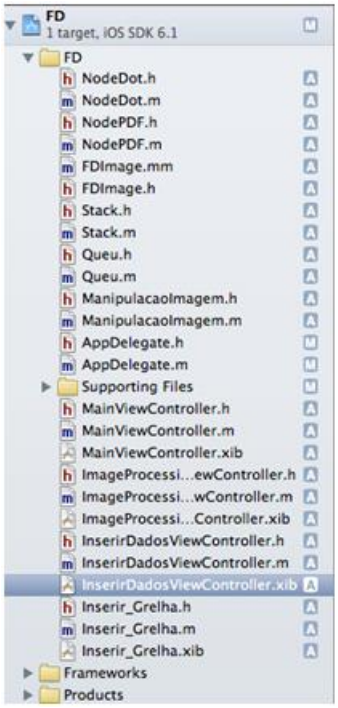


Figura 4.13: Esquema geral do funcionamento da aplicação dot-blot para o sistema iOS.

com setas laranjas e a ordem de transição com círculos azuis).

4.3.1.1 MainView

A "view"MainView (ilustrada na figura 4.14) será a primeira interface apresentada ao utilizador, esta é composta por um único botão **Iniciar Dots** e ao ser pressionado o sistema transita para a próxima "view- InserirDadosView.

Figura 4.14: MainView da aplicação dot-blot para o sistema iOS.

4.3.1.2 InserirDadosView

A "view"InserirDadosView ilustrada na figura 4.15 será a segunda "view"mostrada ao utilizador, esta "view"é composta por três TextFields sendo x as dimensões horizontais da grelha, y as dimensões verticais da grelha e P o identificador do tipo de marcador a povoar a grelha. A TextField P surgiu da necessidade de uma maior eficácia no processo de inserção do nome de cada tipo de marcador na grelha virtual. Esta melhoramento foi necessário uma vez que muitos dos ensaios experimentais utilizavam grelhas compostas por dots do mesmo tipo. Quando o utilizador tiver inserido os dados corretamente é necessário pressionar o botão Gerar Grelha para transitar para a próxima "view".

Figura 4.15: InserirDadosView da aplicação dot-blot para o sistema iOS

4.3.1.3 InserirGrelhaView

Quando o utilizador pressiona o botão Gerar Grelha na "view"InserirDadosView este transita para a próxima "view"InserirGrelhaView (representado na figura 4.16). Nesta "view"é mostrada ao utilizador a grelha virtual com as dimensões definidas na "view"anterior. Nesta grelha virtual os quatro cantos são preenchidos com marcadores de controlo ON (com o identificador 1) e as restantes posições da grelha compostas por marcadores do tipo P (especificados na "view"anterior).
Caso a grelha não seja homogênea então é necessário inserir cada tipo de marcador na sua posição específica da grelha. Para tal deve-se inserir o nome do tipo de marcador na TextField **Inserir Grelha** e seguidamente clicar na posição da grelha que se deseja

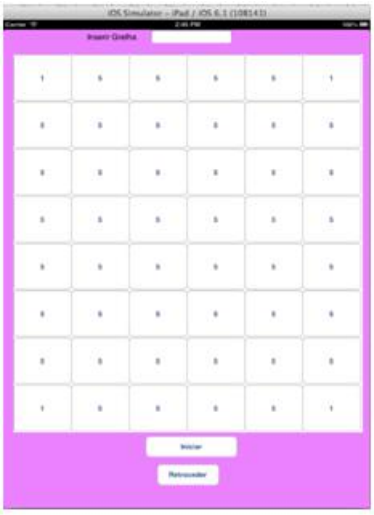


Figura 4.16: InserirGrelhaView da aplicação dot-blot para o sistema iOS.

inserir. Quando todos os campos estiverem preenchidos, o utilizador deverá carregar no botão **Iniciar** para avançar para a próxima "view".

4.3.1.4 FotoViewController

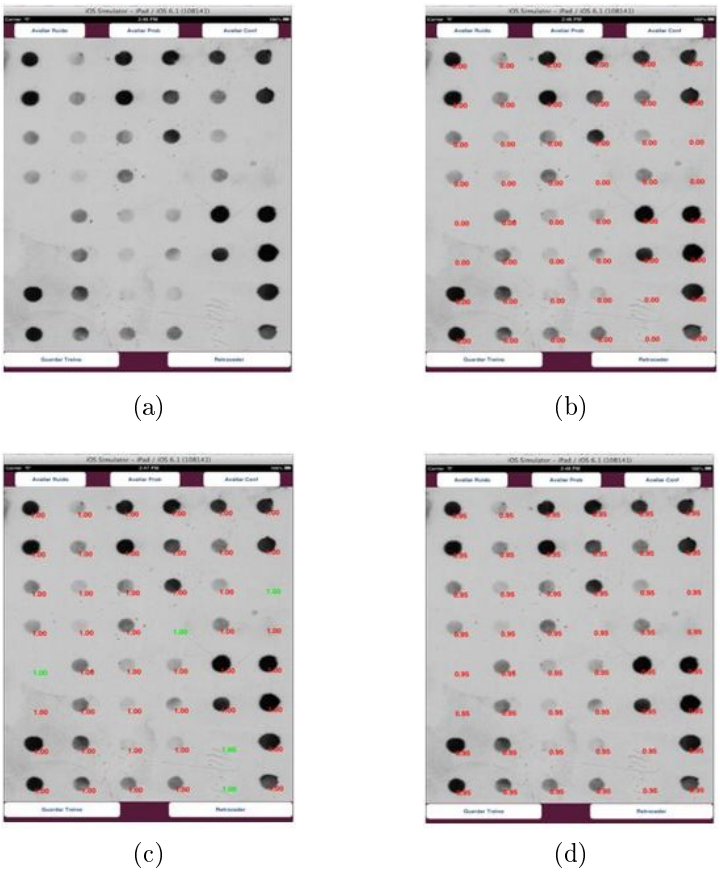
A "view"FotoView (ilustrada na figura 4.17) é a quarta "view"a ser apresentada ao utilizador, esta é composta por uma "Tab Bar"com dois botões **Camera** e **Camera-Roll**.

Figura 4.17: FotoViewController da aplicação dot-blot para o sistema iOS.

Ao pressionar o botão **Camera** o sistema inicializa a câmara do smartphone. Ao carregar no botão **CameraRoll** o sistema inicializa a galeria de foto onde poderá escolher uma foto que foi previamente armazenada no disco. Toda a estrutura e implementação desta "view"poderá ser encontrada no anexo 4.

4.3.1.5 ImageProcessingView

A ImageProcessingView (ilustrada na figura 4.18) é a "view"onde irá ser realizada todo o processamento de cálculo da aplicação e é composta por quatro partes fundamentais:



(a) (b)

(c) (d)

Figura 4.18: A "view"ImageProcessingView em diferentes momentos do seu ciclo de vida: (a) ao carregar a "view", (b) ao pressionar o botão **Avaliar Ruído**, (c) ao pressionar o botão **Avaliar Confiança**, (d) ao pressionar o botão **Avaliar Confiança**.

- Avaliar o ruído da imagem - O utilizador ao pressionar o botão **Avaliar Ruido** poderá avaliar o ruído em torno de cada marca na imagem. O ruído é classificado numa determinada escala apresentada na secção 4.10

- Avaliar a probabilidade de um determinado dot estar ON - Ao pressionar o botão **Avaliar Prob** o utilizador poderá inferir com que grau de probabilidade uma determinada marca corresponde a um ON (desde que este tipo de marca esteja presente no treino adaptativo).
- Avaliar o grau de confiança na classificação - Carregando no botão **Avaliar Conf** o utilizador poderá saber qual o grau de confiança de um determinado dot estar ON ou OFF.
- Guardar os resultados experimentais - Caso se deseje guardar uma imagem para treino, o utilizador deverá pressionar o botão **Guardar Treino**.

Se o utilizador desejar processar outra imagem deverá pressionar o botão **Retroceder** transitando desta forma para a "view"MainView, iniciando de novo todo o processo.

Capítulo 5

Processamento de imagens folha em iOS

Este capítulo descreve o desenvolvimento em iOS de uma aplicação simplificada para o reconhecimento de plantas com base de uma imagem folha.

5.1 Introdução

A identificação taxonômica de plantas é realizada normalmente por um conjunto de especialistas chamados taxonomistas. Contudo nos últimos anos têm-se assistido a uma tendência crescente para automação de tarefa, fruto de um desenvolvimento cada vez mais acentuado de ferramentas computacionais e de algoritmos cada vez mais eficientes. A possibilidade de identificar taxonomicamente uma planta com base em informações morfométricas (de forma automática) obtidas através de imagens digitais das folhas é um assunto que motiva cada vez mais a comunidade científica. O desenvolvimento de um sistema para o reconhecimento automático de imagens do tipo de folha poderá proporcionar, tanto a especialistas como não especialistas, uma valiosa ferramenta de trabalho com custos muito reduzidos [2]. No entanto existe um conjunto de dificuldades técnicas e teóricas no processamento destas imagens, tais como: espécies deformáveis, definição do contorno, inapropriada digitalização e alterações do aspeto da folha como consequência de diversos tipos de contaminações. Os operadores humanos (botânicos) utilizam muitas características das folhas no seu estudo morfológico e taxonômico, nomeadamente a forma bidimensional do bordo da folha, considerado por muitos cientistas uma das variáveis com maior poder discrimi-

nante [2].

O método utilizado para descrever a forma da folha é o método "Shape Feature"que consiste num conjunto de medições que descrevem uma determinada forma de acordo com as suas propriedades geométricas fundamentais. Dentro desta categoria os descritores que foram implementados foram "aspect ratio", "Eccentricity e "Elongation"[2].

5.2 Estrutura da aplicação análise e processamento de imagens do tipo folha para o sistema iOS

A estrutura da aplicação **análise e processamento de imagens do tipo folha para o sistema iOS** (ilustrado na figura 5.1) é composta pelo seguinte conjunto de diretórios:

- Diretório AppDelegated ;
- AppDelegatedTests - Não é necessário;
- Framework;
- Products;

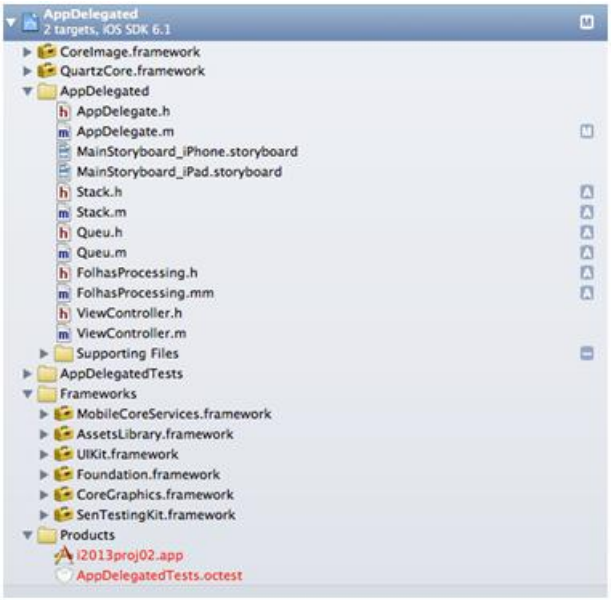


Figura 5.1: A estrutura da aplicação análise e processamento de imagens do tipo folha para o sistema iOS.

Dentro do diretório AppDelegated podemos encontrar o seguinte conjunto de ficheiros:

- AppDelegated.m e AppDelegated.h - Responsáveis pelo ciclo de vida da aplicação.
- Ficheiros Stack.m e Stack.h - Subclasse da classe NSArray utilizada frequentemente durante o código da aplicação como classe de apoio [19].
- Ficheiros Queu.m e Queu.h - Subclasse da classe NSArray utilizada frequentemente durante o código da aplicação como classe de apoio [19].
- FolhasProcessing.mm e FolhasProcessing.h - A extensão .mm indica ao compilador para compilar o ficheiro FolhasProcessing.mm como um ficheiro C. Estes ficheiros contêm o núcleo de toda a aplicação e todos os métodos, estruturas utilizadas para o processamento da imagem.
- ViewController.m e ViewController.h - Responsáveis por controlar o conteúdo da *user Interface*.
- Subdiretório Supporting File - Contém um conjunto de ficheiros responsáveis pela interação com o utilizador.

O subdiretório Supporting File (ilustrado na figura 5.2) contêm o seguinte conjunto de ficheiros:

- MainViewController.m, MainViewController.h e MainViewController.xib - Ficheiros de controle da view MainView da aplicação.
- Main.m - Função que inicia a execução da aplicação iOS.
- MyDataBase.txt- Ficheiro onde será guardado os resultados treino.
- ImageProcessingViewController.h, ImageProcessingViewController.m e ImageProcessingViewController.xib - Ficheiros de controle da view ImageProcessingView da aplicação. Esta view contém um painel principal com os resultados da segmentação da imagem e três painéis secundários (na parte superior da view) contendo três espécies distintas de folhas que apresentam maior grau de semelhança com a folha em análise de entre todas as folhas presentes no treino.
- ManipularImagem.h, ManipularImagem.m e ManipularImagem.xib - Ficheiros de controle da view ManipularImagemView da aplicação. Nesta view o utilizador seleciona a área da imagem que será utilizada no processamento na view ImageProcessingView.
- FotoViewController.m, FotoViewController.h e FotoViewController.xib - Ficheiros de controle da view FotoView da aplicação. Nesta view o utilizador poderá

escolher entre processar uma foto que foi previamente armazenada no disco ou tirar uma foto utilizando a câmara do iPad/ iPhone.

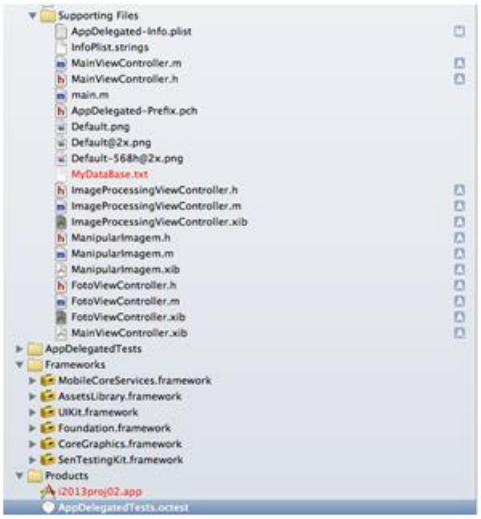


Figura 5.2: Subdiretório Supporting Files da aplicação análise e processamento de imagens do tipo folha para o sistema iOS.

5.3 Panorama da aplicação

A aplicação **análise e processamento de imagens do tipo folha para o sistema iOS** é composta por quatro views:

- MainView;
- FotoView;
- ManipularImagemView;
- ImageProcessingView.

A figura 5.3 representa um esquema geral do funcionamento da aplicação APIF para o sistema iOS onde as transições possíveis entre views encontram-se representadas com setas vermelhas e a sequência de transições com círculos azuis.

5.3.1 MainView

A view MainView (ilustrada na figura 5.4) será a primeira view a ser apresentada ao utilizador. Esta é composta por dois botões: o botão **Iniciar Aplicação** e o botão

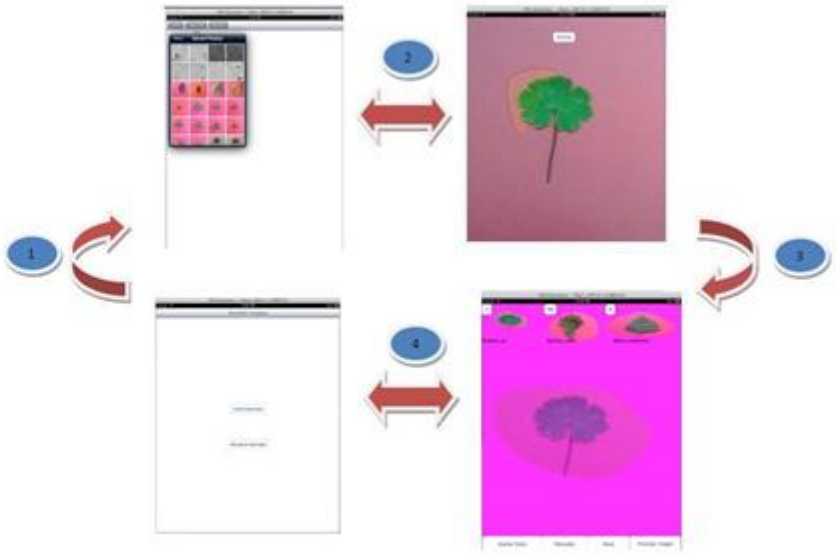


Figura 5.3: Esquema geral do funcionamento da aplicação APIF para o sistema iOS

Recuperar Aplicação.

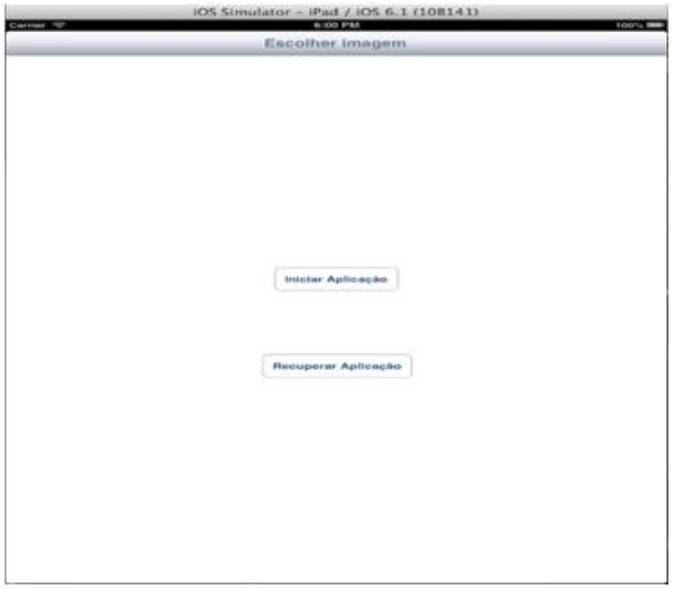


Figura 5.4: View MainView da aplicação APIF para o sistema iOS

Ao pressionar o botão **Iniciar Aplicação** o sistema inicializa a aplicação, transitando para a view FotoView. O botão **Recuperar Aplicação** deve ser utilizado para recuperar o ficheiro MyDataBase.txt que serve como base de dados da aplicação, uma vez que nesta aplicação o utilizador tem permissões de adicionar e remover elementos

da base de dados e este poderá causar acidentalmente problemas de inconsistência na base de dados

5.3.2 FotoView e ManipularImagemView

A view FotoView trata-se da mesma view presente na aplicação APIdb para o sistema iOS (a discussão sobre esta view encontra-se no apêndice E. Após ter selecionado a foto (quer esta tenha sido tirada com a câmara do dispositivo ou tenha sido escolhida uma previamente guardada no disco) transitamos para a próxima view ManipularImagemView (representado na figura 5.5). Nesta view o utilizador irá selecionar a área da imagem que será utilizada para o processamento, por norma é esperado que o utilizador selecione apenas a área circunscrita a folha e não selecione outros objetos ou componentes da folha como seja o caule da folha.

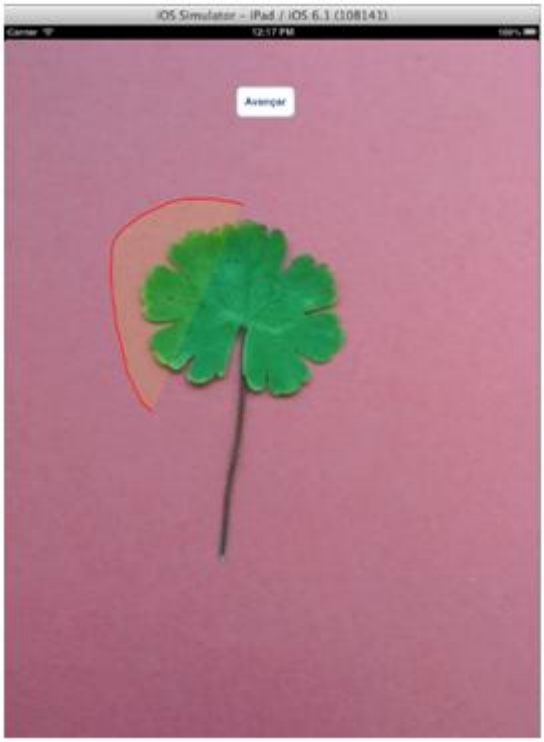


Figura 5.5: View ManipularImagemView da aplicação análise e processamento de imagens do tipo folha para o sistema iOS.

Para evitar potenciais problemas de processamento a foto terá de ter uma cor de fundo contrastante em relação à folha, isto porque o único critério de segmentação que está a ser utilizado pelo algoritmo é a cor do pixel. Como tal, dois objetos com cores semelhantes serão classificados pelo algoritmo como sendo apenas um único

objeto, levando a comportamentos anómalos e imprevisíveis por parte da aplicação. Quando a cor do fundo não é contrastante em relação à cor da folha, o único modo do algoritmo funcionar corretamente é que o utilizador selecione cuidadosamente todo o bordo da folha. Após o utilizador ter selecionado a área da imagem é necessário carregar no botão **Avançar** para progredir para a próxima view ImageProcessingView (mais informações sobre esta view podem ser encontradas no anexo F).

5.3.3 ImageProcessingView

Ao pressionar o botão **Avançar** na view ManipularImagemView o sistema avança para a view ImageProcessingView (ilustrada na figura 5.6). Ao carregar esta view o sistema irá realizar todo o processo de extração do bordo da folha e das variáveis que serão utilizadas para o treino ou para o processo de classificação taxionômica. Caso não tenha ocorrido nenhum erro é carregado para painel principal da view a imagem do bordo da folha sobreposta com a imagem da folha original.

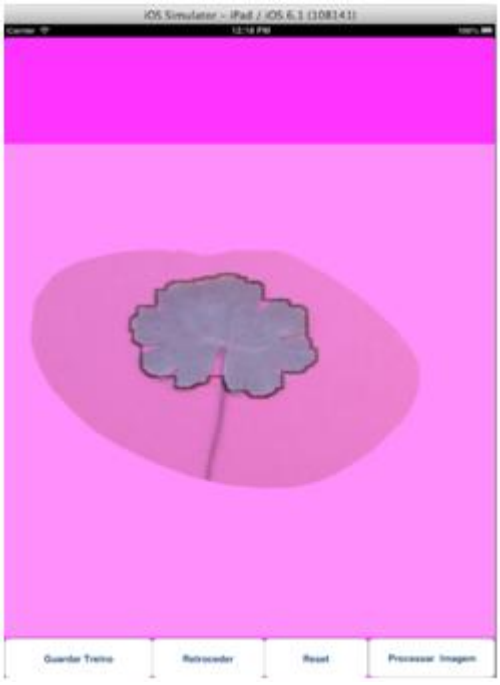


Figura 5.6: View ImageProcessingView da aplicação análise e processamento de imagens do tipo folha para o sistema iOS (após carregamento).

Após o sistema ter extraído informações sobre o bordo da folha, o utilizador poderá escolher uma das seguintes opções:

- Carregar no botão **Retroceder** para retornar a view anterior;
- Pressionar o botão **Processar Imagem** para identificar quais são as três espécies de folha que apresentam maior grau de semelhança com a folha em análise;
- Guardar os resultados do processamento em **Guardar Treino**;
- Corrigir manualmente a base de dados em **Reset**.

5.3.3.1 Processar imagem

Caso o utilizador tenha pressionado o botão **Processar Imagem** são mostradas três possíveis alternativas de classificação na parte superior da view (tal como é ilustrado na figura 5.7), com o resultado mais provável de classificação encontrada na primeira subview (canto superior esquerdo do monitor). Cada subview é constituída por uma imagem com o respetivo nome da espécie e uma etiqueta (no canto superior esquerdo) indicando o grau de confiança atribuído a essa classificação.

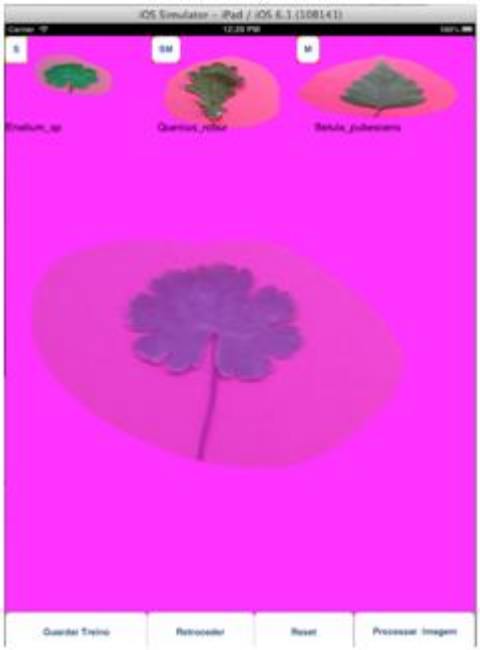


Figura 5.7: View ImageProcessingView da aplicação APIF para o sistema iOS (após pressionar o botão **Processar Imagem**.

5.3.3.2 Guardar treino

No caso em que o utilizador opte por guardar os resultados do treino (tal como é ilustrado na figura 5.8) é mostrada ao utilizador uma caixa de texto no centro do monitor onde poderá inserir o nome da espécie na base de dados. O nome a inserir não poderá conter espaçamento entre as palavras e a norma adotada é utilização do underscore (ABC_DFG_) entre cada palavra. Caso todo o procedimento tenha ocorrido sem nenhum erro é apresentado ao utilizador no canto superior do monitor, a folha que anteriormente fora selecionada.

(a) ImageProcessingViewC (b) ImageProcessingViewD

Figura 5.8: A view ImageProcessingView da aplicação análise e processamento de imagens do tipo folha para o sistema iOS (após pressionar o botão Guardar Treino): (A) processo de inserção do nome da base de dados. (B) Após inserção do nome na base de dados.

5.3.3.3 Reset

Quando o utilizador pressiona o botão **Reset** é apresentada numa caixa de texto com a base de dados ao (tal como é ilustrado na figura 5.9). O utilizador possui todas as permissões tanto de escrita como de leitura podendo deste modo alterar toda a base de dados sem nenhum mecanismo de controlo por parte da aplicação, sendo portanto da responsabilidade do utilizador manter a consistência da base de dados da aplicação.

Figura 5.9: View ImageProcessingView da aplicação análise e processamento de imagens do tipo folha para o sistema iOS (após pressionar o botão **Reset**).

Caso o utilizador deseje restaurar a base de dados original da sua aplicação deverá transitar para a MainView e pressionar o botão **Recuperar Aplicação**. Esta opção irá substituir a base de dados em uso pela versão inicial, sem quaisquer alterações feitas por parte do utilizador.

Capítulo 6

Conclusões

Neste capítulo resumiremos todo o trabalho desenvolvido, referenciando as principais contribuições e indicando possíveis perspetivas futuras de trabalho.

6.1 Contribuições deste trabalho

Uma das contribuições desta tese foi a produção de três versões da mesma aplicação intituladas **Análise e Processamento de Imagens dot-blot** para os sistemas java, androide, iOS. Esta aplicação (APIdb) têm como principal funcionalidade permitir classificar os dot-blot (macroarray) presente na imagem como sendo On ou Off e qual o grau de confiança atribuída a essa classificação. A aplicação APIdb (em quaisquer das três versões) possui um conjunto de funcionalidades únicas tais como: permitir ao utilizador requisitar uma foto (quer esta tenha sido adquirida pela câmara do dispositivo ou pela galeria de fotos), criar uma grelha virtual com as dimensões e tipo de marcador específicas para cada tipo de imagem dot-blot (esta informação é inserida pelo utilizador), classificar a imagem dot-blot relativamente ao nível de ruído presente na imagem e classificar as marcas presentes nas imagens como sendo dots On ou Off com um grau de confiança associada.

O outro grande contributo foi o desenvolvimento de uma aplicação intitulada **Análise e Processamento de Imagens Folha** para o sistema iOS, esta aplicação utiliza três tipos de descritores (aspecto ratio, eccentricity e elongation) de forma a descrever uma determinada folha. A aplicação (APIF) possui características únicas, tais como: permitir ao utilizador requisitar uma imagem (quer esta tenha sido adquirida pela câmara do dispositivo ou pela galeria de imagens), selecionar a área da imagem a pro-

cessar e fornecer três possíveis alternativas para a classificação taxonômica da imagem folha em análise. A versão simplificada da aplicação (APIF) consegue descriminar uma planta num conjunto não superior a cinco espécies de planta. Esta limitação está relacionada com o reduzido número de variáveis (descritores) considerados.

6.2 Possibilidades de trabalho futuro

As aplicações desenvolvidas têm naturalmente funcionalidades que podem ser melhoradas e novas tarefas que podem ser implementadas, que acordo com os requisitos de potências utilizadores. A aplicação APIdb necessita que o utilizador insira previamente as dimensões da grelha antes de a criar, uma possível melhoria poderá ser a inclusão de uma funcionalidade que permita identificar as dimensões da grelha de forma automática e independente do utilizador.

Uma possível melhoria da aplicação APIF consistiria em aumentar o número de descritores utilizados de tal forma que a aplicação seja capaz de lidar com um vasta gama de espécies de planta. Eventualmente em trabalhos futuros a questão da base de dados tornar-se-ia extremamente importante, ou seja, esta aplicação manipula um conjunto relativamente pequeno de dados e portanto questões relativas ao espaço de armazenamento ou pesquisas eficientes na base de dados não se propõem. No entanto, o problema agrava-se quando esta mesma base de dados aumenta de tamanho (o que acontece à medida que o utilizador vai inserindo novas espécies na sua aplicação), levando a que os custos computacionais e os tempos de espera comecem a ser de tal modo insuportáveis que a aplicação em si tornar-se-ia completamente inútil. Uma possível solução seria a transferência da base de dados da aplicação para um servidor, de tal forma que todo o processo de cálculo seria realizado por parte do servidor, aliviando deste modo a carga de processamento na aplicação.

Um outro aspecto a salientar é que todas as aplicações APIdb e APIF (java, android, iOS) utilizam ficheiros txt no processo de armazenamento (o que é desvantajoso quando o volume de dados a manipular é elevado). Uma possível alternativa é utilização de ferramentas específicas disponíveis tanto para o sistema android (SQLite) como para o sistema iOS (Core Data).

Apêndice A

Tópicos de programação

Neste capítulo são introduzidos alguns conceitos essências dos sistemas operativos android e iOS. O android é um sistema operacional baseado no núcleo do linux para dispositivos móveis, foi desenvolvido pela Open Handset Alliance sendo atualmente liderada pela Google e por outras companhias [14]. Pelo contrário o sistema operativo móvel iOS foi desenvolvido pela Apple Corporation, companhia rival da Google. O android utiliza como linguagem de programação o java enquanto que o iOS utiliza como linguagem de programação o objective-C [15].

A.1 Sistema android

O android possuiu características únicas tais como:

- Linguagem aberta à comunidade, baseada na plataforma Linux e open source. Partes de uma aplicação podem ser utilizadas por outras aplicações [3].
- Dezenas de serviços disponíveis: GPS, SQL database, Browser, map views podem ser utilizados pela aplicação [3].
- Gerência automática do ciclo de vida da aplicação. Os programas no sistema operativo são isolados por múltiplas camadas de segurança, impedindo assim que uma aplicação possa tomar controlo do sistema ou corromper o funcionamento de outras aplicações [3].
- Grande qualidade gráfica e de som. Utilização do Open GL 3D inspirada no Flash. Codecs para a maioria dos formatos padrões áudio e vídeo, incluindo H.264 (AVC), MP3 e AAC [3].

- Portabilidade. Todos os programas são escritos em java e executados pelo *Android's Dalvik virtual Machine*, isto é, o código é portátil nas mais diversas arquiteturas de computadores como ARM ou x86 [3].
- Suporte para uma variedade de inputs tais como: TouchScreen, keyboard, e trackball [3].

A.2 Arquitetura do sistema android

O sistema android é construído por uma série consecutiva de camadas, em que cada camada visa a fornecer os seus serviços a camadas superiores tal como é ilustrado na figura A.1. O sistema android é construído por cima de uma camada de software chamado Linux Kernel. O linux fornece uma camada de abstração do hardware ao android, permitindo deste modo que android seja portável numa grande variedade de plataformas. Internamente o android usa o linux como seu sistema operativo. O linux é responsável gestão de memória das aplicações, gestão de input/output, gestão de processos etc [3]. A camada por cima do Kernel contém as librarias nativas do android. Estas librarias escritas maioritariamente em C ou C++, existem para auxiliar na escrita das aplicações. Estas são invocadas por camadas de software de nível superior [3]. Por cima da camada de Kernel existe *Android's Dalvik virtual Machine* e o core Java libraries [3]. Dalvik virtual Machine é uma implementação Google do java otimizado para sistemas móveis Por cima desta encontra-se a camada Application Framework aqui encontra-se o núcleo da aplicação [3]. Algumas componentes importantes da framework são as seguintes:

- Activity Manager - Controla o ciclo de vida da aplicação;
- Content providers - Encapsulam os dados que serão partilhados por outras aplicações;
- Resource Manager - Controlo os recurso da aplicação;
- Location Manager - Controlo a transição entre estados na aplicação e manter informações de estado;
- Notification Manager - Controlo o serviço de mensagens dentro e fora da aplicação.

A camada mais acima de uma aplicação android é a camada Application and Widgets [3], que é a parte da aplicação que será visível aos utilizadores.

Um esquema geral do arquitetura do sistema android é dado pela figura A.1

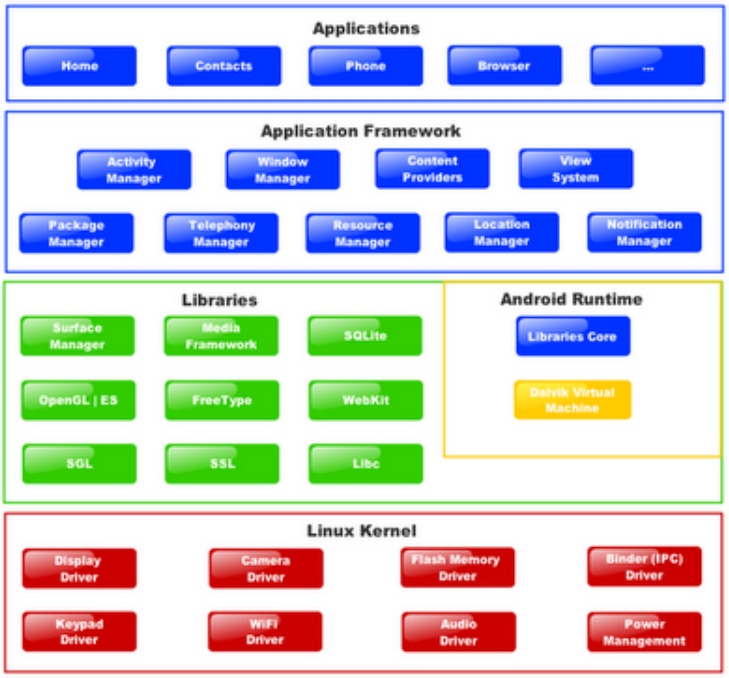


Figura A.1: Arquitetura de sistema android [3].

A.3 Ciclo de vida de uma *"activity"*

Internamente cada interface monitor possui a sua classe "activity", cada "activity"têm o seu próprio ciclo de vida. Uma aplicação é formada por uma ou mais "activity"para além dos processos linux que as contém. No android o ciclo de vida de uma "activity"não está totalmente dependente do ciclo de vida de um processo, os processos são apenas recipientes descartáveis de uma "activity"[3]. Um exemplo do esquema de funcionamento do ciclo de vida de uma "activity"é ilustrada na figura A.2.

Durante o ciclo de vida de uma aplicação, cada "activity"[3] pode estar num dos vários estados tal como é representado na A.2. Quando uma "activity"transita de estado o programador é notificado dessa mudança de estado por troca de mensagens com o sistema operativo. "Activities"que não estejam a correr podem ser paradas ou os processos linux que os contém podem ser eliminados pelo sistema operativo de modo a libertar memória para um novo processo a decorrer.

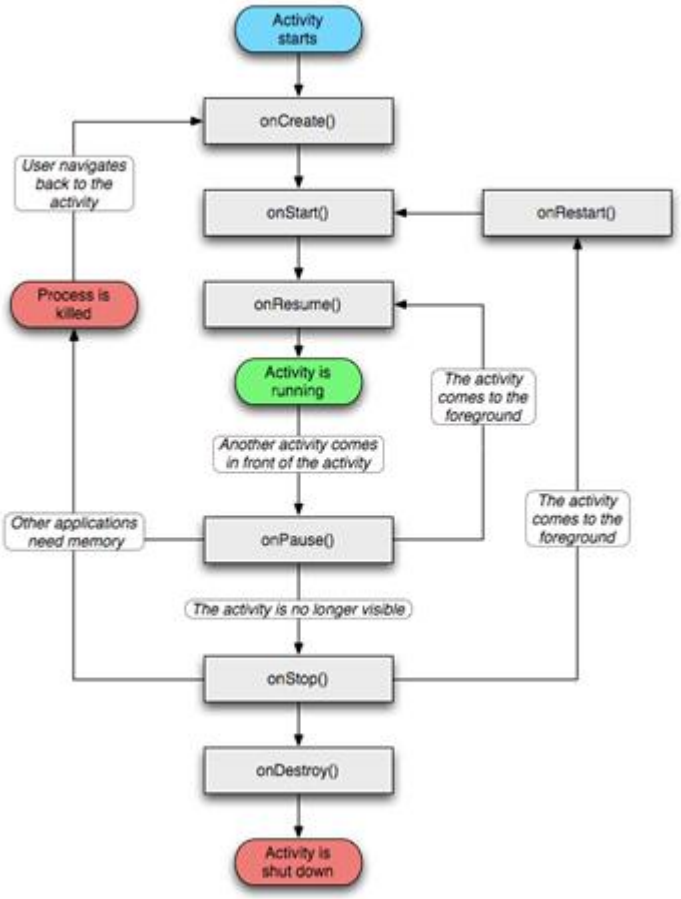


Figura A.2: Ciclo de vida de uma "activity"[3]

A.4 Sistema iOS

O sistema iOS foi desenvolvido originalmente para o iPhone, mas também é usado em iPod touch, iPad e Apple TV. A interface do usuário do iOS é baseado no conceito de manipulação direta, utilizando gestos em multi-toque. Acelerômetros internos são usados por alguns aplicativos para responder à agitação do aparelho ou rodá-la em três dimensões. O iOS é constituído quatro camadas de abstração: a camada Core OS, a camada Core Services, a camada Média, e a camada Cocoa Touch. O sistema operacional usa aproximadamente 960 MB de armazenamento do dispositivo, que varia para cada modelo [5].

O sistema iOS possui características únicas tais como [5]:

- Permite um conjunto de interações únicas com utilizador;

- Atualizações rápidas;
- Todas as aplicações possuem certificados de qualidade;
- Sistema com alto nível de desempenho ;
- Possui um número elevado de recursos: Safari, Siri, Maps, AppStore, FaceTime,Icloud, etc.

A.5 Arquitetura do sistema iOS

O kernel do iOS é baseado no *kernel Mach*. No topo deste kernel estão as camadas de serviços que são utilizados para implementar as aplicações. Na figura A.3 encontra-se um esquema representativo da arquitetura do sistema iOS [4].

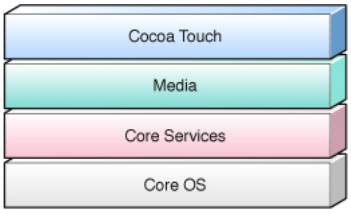


Figura A.3: Arquitetura do sistema iOS [4].

A camada *Core OS* e *Core Services* contêm as interfaces fundamentais do iOS, incluindo acesso a arquivos, tipos de dados de baixo nível, sockets de rede, etc. As interfaces são escritas em C e incluem tecnologias como a *Core Foundation*, *CFNetwork*, *SQLite*, etc [4] .

A camada de *Media* contém ferramentas de desenho 2D e 3D, áudio e vídeo. Esta camada inclui serviços como o *OpenGL*, *Quartz* ,*Core Audio* e *Core Animation* [4].

Na camada *Cocoa Touch* fornece um conjunto de frameworks essenciais para a construção de qualquer aplicação. Por exemplo:

- *Foundation framework* fornece suporte para coleções, gestão de arquivos, operações de rede, etc;
- *UIKit framework* fornece infra-estruturas para a construção das interfaces.

Padrão de Desnho do Sistema iOS

Uma aplicação em iOS segue o padrão de desenho *"Model-View-Controller"*. Neste modelo a componente de dados e a representação visual encontram-se separados e são utilizados controladores para estabelecer a comunicação entre as duas partes, tal como é ilustrado na figura A.4 [5].

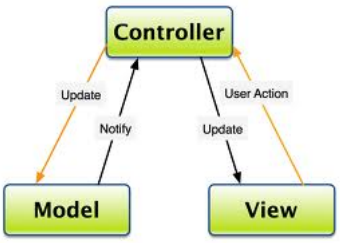


Figura A.4: Padrão de desenho Model-View-Controller [5].

Assim, todo o modelo fica salvaguardado sendo apenas acedido pelos controladores existentes. Através de um padrão de desenho *delegate* são permitidas trocas de informação entre objetos. Para implementar reações a eventos do utilizador é implementado o padrão de desenho *target-action*[5].

A.6 Ciclo de vida de uma aplicação iOS

Como qualquer programa em C, uma aplicação iOS têm uma função *main ()* que será a primeira função a ser invocada pela aplicação. Dentro desta função é invocada a função *UIApplicationMain ()*, responsável pela iniciação da aplicação. Cada *view* está associada uma *view controller* e são apenas carregadas quando a *view controller* é carregada [16] [17].

É também instanciado uma *application delegate* que será responsável pelo tratamento de eventos da própria aplicação. Após isto a aplicação entra num evento loop até que o utilizador interage com a aplicação, em cada iteração dentro do loop é criada uma nova autorelease pool, que será usado para gestão de memória durante esse ciclo [16] [17].

A invocação da *UIApplicationMain ()* dá início ao ciclo contínuo da vida da aplicação. Durante este ciclo todos os eventos originados pelo utilizador são colocados numa fila,

sendo posteriormente retirados desta e encaminhados pela aplicação para os objetos maís indicados para os tratar [16] [17].

A classe *AppDelegate* representado na figura A.5 adota o protocolo *UIApplicationDelegate* e terá a responsabilidade da gestão do ciclo de vida da aplicação. O protocolo *UIApplicationDelegate* redefine vários métodos que fazem parte do ciclo de vida de uma aplicação [16] [17].

```
#import <UIKit/UIKit.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@end
```

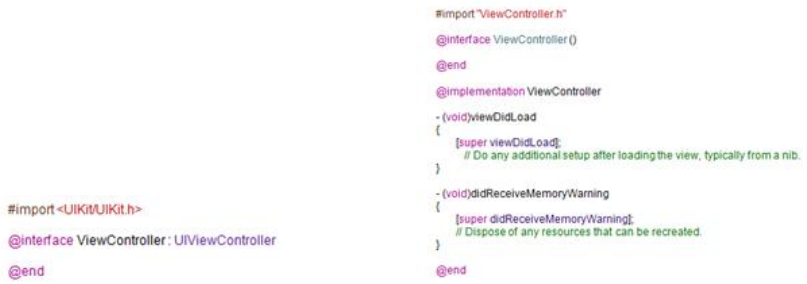
Figura A.5: AppDelegated.h.

O método *application DidFinishLaunching* representado na figura A.6 é invocado pelo objeto da aplicação e informa o *delegate* da aplicação que já terminou o seu processo. Este método é utilizado para inicializar a aplicação e criar a *user interface*, é criada uma janela principal da aplicação (alocada e inicializada) e por fim tornada visível [16] [17]. A *UIWindow* fica então disponível para apresentar os conteúdos dos diversos controladores e as suas respetivas *views*.

```
#import "AppDelegate.h"
#import "MainViewController.h"

@implementation AppDelegate

- (void)dealloc
{
    [_window release];
    [super dealloc];
}

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]
autorelease];

    self.window.rootViewController =
    [[[UINavigationController alloc] initWithRootViewController:
      [[[MainViewController alloc] initWithNibName:nil
                                            bundle:nil] autorelease]] autorelease];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Figura A.6: AppDelegated.m.

A *view Controller* representado na figura é responsável por controlar o conteúdo da *user interface* e lidar com eventos do utilizador [5].



(a) ViewController.h (b) ViewController.m

Figura A.7: ViewController.

A *ViewController* [5] é uma subclasse da *UIView Controller* e faz gestão de uma única view de topo.

- viewDidLoad: Método invocado após a apresentação da View;
- didReceiveMemoryWarming: Liberta todas as referências e dependências da View.

A.7 Gestão de memória

A gestão de memória é algo fulcral para o correto funcionamento de uma aplicação. Os objetos em Objetive-c têm um ciclo de vida dentro da aplicação. Quando os objetos são instanciados são colocados numa dada posição de memória e quando já não são mais necessários são dealocados a fim de libertar memória [5]. Em Objective-c existem três mecanismos para a gestão de memória[5]:

- Manual Reference Counting (MRC) - Neste mecanismo o utilizador têm a total responsabilidade do ciclo de vida de um objeto;
- Automatic Reference Counting (ARC) - Introduzido a partir das versões ios5
- Garbage Collection (GC) - Uma thread especial (Garbage Collector) fica responsável pela gestão de automática do ciclo de vida de um objeto. Este mecanismo apesar de estar disponível em Objective-C não é recomendado a sua utilização por questões de performance.

O mecanismo utilizado neste trabalho foi o ARC e MRC. Em MRC cada objeto possui um atributo chamado *retain count* sendo apenas eliminado quando o seu *retain count* se encontra a zero. O funcionamento deste mecanismo está centrado na noção de

posse. Quando um objeto é adquirido por uma função ou método o *retain count* é incrementado de 1 o inverso pode acontecer quando um método liberta a posse de um objeto e o *retain count*. Sempre que se inicializa o objeto com operações de *+alloc*, *allocWithZone* o contador do objeto é inicializado a 1. Sempre que se quer adquirir a posse de um objeto invoca-se a operação *retain*. Para libertar a posse do mesmo invoca-se a operação *release* [5].

Em torno destas operações que alteram o contador de retenções foi criado um conjunto de regras que permitem efetuar uma correta gestão do ciclo de vida de uma aplicação, impedindo assim fugas de memória e *Stack Overflow* [5]:

1. Adquirir um objeto através de um dos seguintes métodos: *+alloc*, *+allocWithZone*, *-copy*, *-mutableCopy* ou *-new*.
2. Adquirir um objeto ao invocar o método *retain*.
3. Guardar um objeto através dos métodos *retain* ou *-copy*.
4. Libertar um objeto com os métodos *release* ou *autorelease*. O método *autorelease* indica a pretensão de libertar o objeto no futuro.
5. Uma classe deve redefinir a seu método *dealloc* para libertar a posse das suas variáveis de instância (este método é invocado quando o *retain count* é 0).
6. Nunca invocar o método *dealloc* diretamente, exceto na implementação do mesmo.

Apêndice B

Aquirir uma foto Android

Para obter uma imagem digital utilizando a câmara do androide foi utilizado o seguinte procedimento:

- Abrir um objeto câmara;
- Criar uma classe CamaraPreview.java;
- Modificar as configurações da câmara;
- Tirar uma foto;
- Reiniciar a classe CamaraPreview.java;
- Libertar a câmara;
- Mostrar os resultados do processamento ao utilizador.

B.1 Abrir um objeto câmara

A primeira tarefa a ser realizada é criar uma instância do objeto câmara através do método *getCameraInstance()* tal como é ilustrado na figura B.1 [20].

```
/** A safe way to get an instance of the Camera object. */
public static Camera getCameraInstance() {
    Camera c = null;
    try {
        c = Camera.open(); // attempt to get a Camera instance

    } catch (Exception e) {
        // Camera is not available (in use or does not exist)
        Log.e("", "Camera is not available (in use or does not exist)", e);
    }
    return c; // returns null if camera is unavailable
}
```

Figura B.1: Método getCameraInstance()

Na figura B.1 o método Camera.open()lança uma exceção se a câmara já estiver a ser utilizada por outra aplicação. O método releaseCamera()(ilustrado na figura B.2) liberta a posse da câmara para outras aplicações [20].

```
private void releaseCamera() {
    if (mCamera != null) {
        mCamera.setPreviewCallback(null);
        mCamera.stopPreview();
        mCamera.release(); // release the camera for other applications
        mCamera = null;
    }
}
```

Figura B.2: Método releaseCamera().

B.2 Criar uma classe CamaraPreview.java

A imagem que é carregada para o screen do androide corresponde ao último frame que foi carregado para a memória do sistema. Para lidar com este tipo de entrada de dados provenientes da câmara e desenhar o seu conteúdo no screen é usado a classe *SurfaceView* [20]. A classe CamaraPreview.java (ilustrada na figura B.3) encontra-se disponível na aplicação dot-blot em SRC/CamaraPreview. Esta classe estende uma *SurfaceView* herdando todos os seus métodos e atributos, implementando a interface de serviço *android.view.SurfaceHolder.Callback*. Esta interface é usada para transmitir dados vindo do hardware da câmara para a nossa aplicação.

```
package pt.up.dcc.vcandroid;

import android.content.Context;

public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {
    private static final String TAG = "CameraPreview";
    private SurfaceHolder mHolder;
    private Camera mCamera;
    private PreviewAnalyzer prevAn;

    public CameraPreview(Context context, Camera camera) {
        super(context);
        setCamera(camera);
    }
    public void setCamera(Camera camera) {
        mCamera = camera;
        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
    }
public SurfaceHolder getholder(){return mHolder;}
public int  getPreviewSizeX( ) {return prevAn.getPreviewSizeX();}
public int  getPreviewSizeY( ) {return prevAn.getPreviewSizeY();}
```

Figura B.3: Classe CamaraPrewiew: Construtores e Getter's.

Após a activity ser criada é chamado o método onStart() que por sua vez invoca o método onResume() (ilustrado na figura B.4). Esta coloca a corrente activity no topo da pilha das activity's sendo esta posteriormente colocada em foreground podendo deste modo interagir com o utilizador [20].

```
@Override
protected void onResume() {
    super.onResume();

    if (!initCamera(getApplicationContext())) {
        Toast.makeText(getApplicationContext(), "ERRO ao iniciar a camera",
                Toast.LENGTH_SHORT).show();
    }
}
```

Figura B.4: Invocação do método initCamera () em onResume ().

O método initCamera(), ilustrado nas figuras B.5 e B.6, será o ponto de origem onde se inicializa o sistema da câmara do androide e é invocado no método onResume(). O objecto Camâra é instanciado e inicializado, sendo posteriormente criado uma instância da CamaraPreview [20]. Como a CamaraPreview é uma extensão da SurfaceView esta pode ser adicionada como uma view a uma determinada FrameLayout.

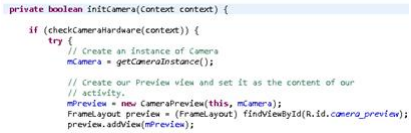


Figura B.5: método initCamera- parte I.

Figura B.6: método initCamera- parte II.

B.3 Configurações da câmara

O próximo passo será configurar certas características da câmara do Android como: diferentes níveis de zoom, tempo de exposição, etc. Ainda a partir da análise do código anterior (no método initCamera) guarda-se todas as resoluções possíveis da câmara numa HaspMap (tal como é ilustrado na figura B.8), que será posteriormente utilizado para criar um menu onde o utilizador poderá escolher o nível de zoom aplicado à câmara (ilustrado na figura B.7) e por fim instanciar e configurar uma classe Paint [6].

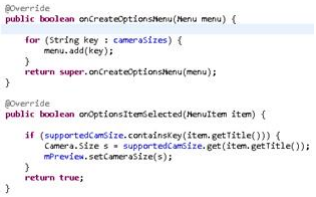


Figura B.7: Criação do menu no método onCreateOptionsMenu() e configuração do nível de zoom escolhido pelo utilizador no método onOptionsItemSelected() [6].

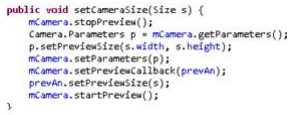


Figura B.8: método setCameraSize() na classe CamaraPreview() [6].

B.4 Tirar a foto

Usar o método Camera.takePicture()(referenciada na figura B.9) para tirar uma fotografia na chamada da classe CamaraPreview (podendo ser passado como argumento os objetos do tipo Camara.ShutterCallback() e Camara.PictureCallback() [6]).

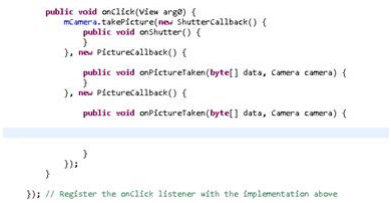


Figura B.9: Chamada ao método Camera.takePicture() dentro do método onCLick() [6].

O método onPictureTaken() é invocado quando uma imagem de dados está disponível após tirar a fotografia.

B.5 Reiniciar a classe CamaraPreview.java

Depois de a fotografia ser obtida é necessário reiniciar a CamaraPreview.java antes que o utilizador possa obter outra (tal como é ilustrado na figura B.10).

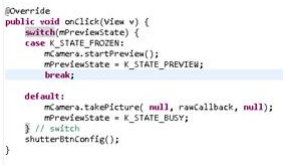


Figura B.10: Reiniciar CamaraPreview [6].

B.6 Libertar a posse da câmara

Quando a câmara deixa de ser necessária pela activity é necessário libertar a sua posse, tal como é ilustrado nas figuras B.11 e B.12. Tal acontece quando o utilizador deixa a presente activity tal como é ilustrado na figura A.2 do capítulo Tópicos de programação.

```
@Override
protected void onPause() {
    releaseCamera();
    super.onPause();
}
```

Figura B.11: Chamada do método releaseCamera() em onPause() [6].

```
private void releaseCamera() {
    if (mCamera != null) {
        mCamera.setPreviewCallback(null);
        mCamera.stopPreview();
        mCamera.release();
        mCamera = null;
    }
}
```

Figura B.12: Método releaseCamera().

B.7 Mostrar os resultados do processamento ao utilizado

Após processar os dados da imagem, a corrente activity termina o seu serviço mostrando os resultados ao utilizador através do método onActivityResult()(ver figura B.13).

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK) {
        if (requestCode == SELECT_PICTURE) {
            Uri selectedImageUri = data.getData();

            // OI FILE Manager
            filemanagerstring = selectedImageUri.getPath();

            // MEDIA GALLERY
            selectedImagePath = getPath(selectedImageUri);

            // DEBUG PURPOSE - you can delete this if you want
            if (selectedImagePath != null)
                System.out.println(selectedImagePath);
            else
                System.out.println("selectedImagePath is null");
            if (filemanagerstring != null)
                System.out.println(filemanagerstring);
            else
                System.out.println("filemanagerstring is null");

            // NOW WE HAVE OUR WANTED STRING
            if (selectedImagePath != null)
                System.out
                        .println("selectedImagePath is the right one for you!");
            else
                System.out
                        .println("filemanagerstring is the right one for you!");

            Intent myIntent = new Intent(this, ShowImg.class);

            myIntent.putExtra("imgPath", selectedImagePath);

            startActivity(myIntent);
        }
    }
}
```

Figura B.13: Método onActivityResult().

Este utiliza como método auxiliar o getPath(), ilustrado na figura B.14.

```
public String getPath(Uri uri) {
    String[] projection = { MediaStore.Images.Media.DATA };
    Cursor cursor = getContentResolver().query(uri, projection, null, null, null);
    if (cursor != null) {
        int column_index = cursor
                .getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
        cursor.moveToFirst();
        return cursor.getString(column_index);
    } else
        return null;
}
```

Figura B.14: Método getPath().

Apêndice C

SaveImage no sistema iOS

As classes Cocoa permitem guardar informações como NSString, UIImage e NSData. Estas funções permite guardar uma determinada imagem num diretório dado um caminho (path) tal como é ilustrado nas figuras C.1 e C.2. O número de componentes por cada pixel no *bitmap graphics contex* é especificado pelo espaço de cor em CGColorSpaceRef. Inicialmente é criado um contexto em CGBitmapContextCreate e criado *bitmap drawing environment*. Quando algo é desenho sobre este contexto, toda a informação é guardada sobre a forma de um *BitMap Data* num local específico na memória. Utilizando esse mesmo contexto criado anteriormente é criado um Objeto UIImage, que por fim será guardado usando os métodos de escrita presente na Classe Cocoa writeToFile [7].

```
BOOL saveImage(const char* filepath,
        const uint32_t* pixelsBuffer,
        int width,
        int height)
{
  BOOL retValue = YES;
  CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
  CGContextRef newContext = CGBitmapContextCreate((void *)pixelsBuffer,
                            width,      // width
                            height,     // height
                            8,          // bitsPerComponent
                            4 * width,  // bytesPerRow
                            colorspace,
                            kCGBitmapByteOrderDefault |
                            kCGImageAlphaPremultipliedLast);

  CGImageRef imgRef = CGBitmapContextCreateImage(newContext);
  UIImage* img = [UIImage imageWithCGImage:imgRef];
  CGImageRelease(imgRef);
  retValue = [UIImagePNGRepresentation(img) writeToFile:[NSString
stringWithCString:filepath encoding:NSUTF8StringEncoding]
                            atomically:NO];
```

Figura C.1: Função SaveImage no sistema iOS - parte I [7],

```
  CGImageRef imgRef = CGBitmapContextCreateImage(newContext);
  UIImage* img = [UIImage imageWithCGImage:imgRef];
  CGImageRelease(imgRef);
  retValue = [UIImagePNGRepresentation(img) writeToFile:[NSString
stringWithCString:filepath encoding:NSUTF8StringEncoding]
                            atomically:NO];

  CGContextRelease(newContext);
  CGColorSpaceRelease(colorspace);

  return retValue;
}
```

Figura C.2: Função SaveImage no sistema iOS - parte II [7].

Apêndice D

Carregar pixéis de uma imagem iOS

A função loadPixelsOfImage (ilustrada na figura D.1) tem como objetivo carregar os pixeis de um objeto UIImage para um bloco de memória numa estrutura de dados do tipo uint_32 (Na figura D.1 identificada pela variável m_pixels). Posteriormente todos os métodos da aplicação irão manipular este mesmo buffer onde eventualmente num futuro próximo será usado para carregar um outro objeto UIImage que por fim será mostrado ao utilizador [7].

```
- (void) loadPixelsOfImage:(UIImage*) image
{
   NSInteger width  = CGImageGetWidth(image.CGImage);
   NSInteger height  = CGImageGetHeight(image.CGImage);
   m_lines = height;
   m_columns = width;
   m_pixels = (uint32_t*)malloc(sizeof(uint32_t) * width * height);
   myresultados = (struct resultados *) malloc(sizeof(resultados));
   m_pixelsH = (uint64_t * ) malloc (sizeof(uint64_t)*width*height);
   m_colorspace = CGColorSpaceCreateDeviceRGB();
      m_context =
   CGBitmapContextCreate(m_pixels,
                         width,      // width
                         height,     // height
                         8,          // bitsPerComponent
                         4 * width,   // bytesPerRow
                         m_colorspace,
                         kCGBitmapByteOrderDefault |
                         kCGImageAlphaPremultipliedLast);
   CGContextSetBlendMode(m_context, kCGBlendModeCopy);
   CGContextDrawImage(m_context,
                      CGRectMake(0.0f, 0.0f, width, height),
                      image.CGImage);}
```

Figura D.1: Função loadPixelsOfImage para o sistema iOS [7]

Apêndice E

Manipular câmara iPhone/iPad

A captação de fotos é feita através da UIImagePickerController, que utiliza a framework MobileCoreServices [8]. Na interface de serviço são implementados vários protocolos entre os quais UIImagePickerControllerDelegate, UINavigationControllerDelegate e UIPopoverControllerDelegate tal como é ilustrado na figura E.1. É criada uma toolbar com dois botões que tem dois comportamentos distintos. Ao pressionar o botão **UseCamera** inicia-se a câmara do Smartphone enquanto que ao pressionar o botão **UseCameraRoll** inicia-se uma nova view onde o utilizador poderá escolher uma foto que foi previamente armazenada no disco.

```
@interface FotoViewController : UIViewController<UIImagePickerControllerDelegate,
UINavigationControllerDelegate, UIPopoverControllerDelegate>{
    UIPopoverController *popoverController;
    BOOL newMedia;
    IBOutlet UIToolbar *ToolBar;
    NSMutableArray *myStringArray;
    IBOutlet UIImageView *imageView;

}
- (IBAction)useCamera: (id)sender;
- (IBAction)useCameraRoll: (id)sender;
```

Figura E.1: FotoViewController.h.

Primeiro é verificado se o dispositivo onde a aplicação está a correr tem câmara (tal como é ilustrado na figura E.2), caso esta possua é criada uma instância do UIImagePickerController que delega qual é a classe responsável por tratar de eventos desta classe. Seguidamente definimos como tipo de fonte a câmara do dispositivo e por fim a última tarefa a ser implementar é colocar a flag *newMedia* a Yes (indicando deste modo que esta é uma nova imagem e não uma guardada na Galeria de Fotos)[8].

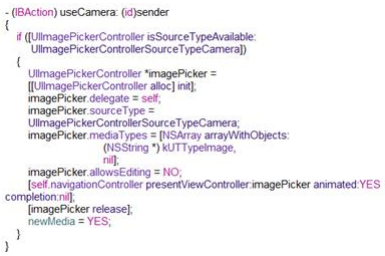


Figura E.2: Função useCamera iOS [8]

O método UseCameraRoll (ilustrado nas figuras E.3 e E.4) é extremamente semelhante com o método definido anteriormente, excetuando que a fonte da UIImagePickerController será a UIImagePickerControllerSourceTypePhotoLibrary e a flag *newMedia* a No [8].

Figura E.3: Função useCameraRoll parte I [8].

Figura E.4: Função useCameraRoll parte II [8].

Depois do utilizador tirar a foto ou ter escolhido uma que se encontrava na sua galeria, a Image Picker termina o seu trabalho e invocará o método didFinishPickingMediaWithInfo (tal como é ilustrado nas figuras E.5 e E.6) [8].

Figura E.5: Função didFinishPickingMediaWithInfo parte I [8].

```
    CGSize tp = CGSizeMake(CGImageGetWidth((image.CGImage))/2.5,
CGImageGetHeight(image.CGImage)/2.5);
 ManipularImagem *myImageProce = [ManipularImagem createWithImage:image
scaledToSize:tp];
     [self.navigationController pushViewController:myImageProce animated:YES];
 }
```

Figura E.6: Função didFinishPickingMediaWithInfo parte II [8].

Com a invocação da puhViewController o sistema transita para a próxima view ManipularImagemView.

Apêndice F

ManipularImagemView

Na classe ManipularImagemView o utilizador irá selecionar a área da imagem que será utilizada para o processamento. A principal libraria utilizada foi a CoreGraphics [21], esta libraria tem uma API denominada Quartz 2D que é um motor de desenho bidimensional para a plataforma iOS, esta possui um conjunto de funcionalidades tais como: path-based-drawing, pinturas com transparência, layers, etc [17] [22]. O segredo desta view é a criação de uma classe gráfica própria que permita lidar com iterações touch [23] do utilizador, desenhar layers sobre a imagem (consoante um determinado percurso escolhido pelo utilizador) e que possua alguma capacidade de feedback (mostrar a seleção do path na imagem aquando da terminação de eventos touch) [24] [25]. No ficheiro ManipularImagem.h é definido a interface de serviço da classe gráfica MyImageView2 (ilustrada na figura F.1) que estende uma classe mais genérica UIImageView. A classe MyImageView herda todos os atributos e métodos da sua superclasse UIImageView.

Em **@property** são definidos duas propriedades: uma imagem imutável (img) e outra imagem mutável que indica a atual seleção da imagem (currentSeletion).

```
@interface MyImageView2 : UIImageView
{
    CGMutablePathRef mypath;
    CAShapeLayer* feedbackLayer;
    UIImage *img;
}
@property (nonatomic, readonly) UIImage* currentSelection;
@property (nonatomic, readonly) UIImage* img;
@end
@class MyImageView2;
@interface ManipularImagem : UIViewController{
    IBOutlet MyImageView2 *myImageView2;
    IBOutlet UIButton *Avancar;
    CGMutablePathRef mypath;
    CGContextRef ctx;
}
```

Figura F.1: Interface MyImageView2 iOS

Dentro da definição da interface ManipularImagem é definido como um dos seus atri-

butos a nova classe gráfica MyImageView2 com uma ligação do tipo outlets permitindo deste modo que as componentes gráficas sejam tratadas como propriedades.

O próximo passo consiste em modificar a classe padrão no **Identity Inspector** (ilustrado na figura F.2) para nova classe criada anteriormente Class MyImageView2.

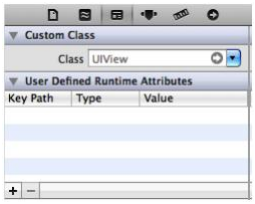


Figura F.2: Identity Inspector iOS.

No ficheiro ManipularImagem.m são implementados os métodos da nova classe MyImageView2. Quando todos os objetos na Interface Builder tiverem sido carregados para a memória e inicializados é invocada o método - **awakeFromNib** (ilustrado na figura F.3). Dentro desta função é autorizado que a classe gráfica MyImageView2 possa interagir com o utilizador e configurado a CAShapeLayer [23] para obter os efeitos de preenchimento desejado.

```
- (void) awakeFromNib
{
    [self setUserInteractionEnabled:YES];

    feedbackLayer = [CAShapeLayer layer];
    feedbackLayer.hidden = YES;
    feedbackLayer.backgroundColor = [[UIColor greenColor]
colorWithAlphaComponent:0.9f].CGColor;
    feedbackLayer.strokeColor = [UIColor redColor].CGColor;
    feedbackLayer.fillColor = [UIColor yellowColor].CGColor;
    feedbackLayer.lineWidth = 2.0f;
    [self.layer addSublayer:feedbackLayer];
}
```

Figura F.3: AwakeFromNib iOS.

Quando o utilizador interage com a aplicação (tocando com algum dedo no monitor do dispositivo) é despoletado a função touchesBegan (ilustrada na figura F.4) [25].

```
/*****************************************************
 * touchesBegan
 *****************************************************/

- (void) touchesBegan:(NSSet *)touches
        withEvent:(UIEvent *)event
{
    [self createPath];

    CGPoint loc = [self touchLocation:touches];
    CGPathMoveToPoint(mypath, nil, loc.x, loc.y);

    feedbackLayer.hidden = NO;
    self.layer.mask = nil;
}
```

Figura F.4: touchesBegan iOS.

Dentro desta função é criado um path depois é localizado o ponto no monitor (representado na figura F.5) onde o utilizador pressionou , adicionado ao caminho atual

(mypath) e finalmente desenhado no monitor do dispositivo(ilustrado na figura F.6) [25].

Figura F.5: touchesLocation iOS.

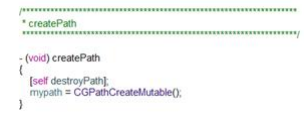


Figura F.6: createPath iOS.

Quando o utilizador arrasta o seu dedo ao longo do monitor é despoletada uma outra função touchesMoved (ilustrado na figura F.7) [25].

Figura F.7: touchesMoved iOS.

Nesta função são localizados todos os conjuntos de pontos selecionados, adicionados ao path corrente e por fim desenhado no monitor do dispositivo criando desta forma um feedback com o utililzador. Quando o utilizador liberta todos os seus dedos do monitor do dispositivo é despoletado a função touchesEnded (representado nas figuras F.8 e F.9). Nesta função é fechado o caminho (mypath) depois é criado uma máscara para filtrar a imagem, sendo o resultado final a imagem selecionada dentro do caminho (myPath). Posteriormente essa mesma imagem é utilizada para carregar uma próxima view ImageProcessingView [25].

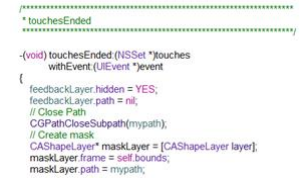


Figura F.8: touchesEnded parteI iOS.

```
    self.layer.mask = maskLayer;
    UIImage* img = [self currentSelection];
    if(img)
    {
        [UIImageJPEGRepresentation(img, 0.9f) writeToFile:[self tempImagePath:
@"selection"]
                                  atomically:NO];
    }

    // Close Path
    [self destroyPath];
}
```

Figura F.9: touchesEnded parteII iOS.

Apêndice G

Estrutura da aplicação APIdb

Uma lista ligada é uma estrutura de dados linear e dinâmica (tal como é ilustrado na figura G.1), composto por células com um campo de dados e um apontador para o próximo elemento da lista [19].

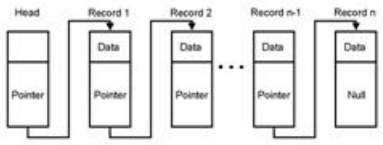


Figura G.1: Estrutura de uma lista ligada.

Uma lista ligada possui as seguintes vantagens [19]:

- Fácil de inserir ou remover no cabeçalho da lista;
- Não é necessário definir (no momento da criação da lista) o número máximo de elementos.

Uma lista ligada possui o seguinte conjunto de desvantagens [19]:

- Para aceder ao elemento na posição n da lista deve-se percorrer os $n-1$ anteriores;
- Difícil de manter a consistência em listas dinâmicas;
- Torna-se desvantajoso utilizar a estrutura à medida que a esta aumenta de dimensões.

A lista ligada contêm nós do tipo **NodeList** (representados na figura G.2), em cada NodeList existem informações relativos a um dot-blot em particular na imagem nomeadamente:

- As coordenadas na imagem (x, y);
- O tipo de dot;
- O número de pontos em cada sub-região $[0, \ldots, 8]$, onde 0 corresponde a região central do dot e as restantes numerações a cada sub-região tal como é ilustrado na figura 3.9 do capítulo 3;
- Intensidade de cada sub-região $[0, \ldots, 8]$;
- Ruído em cada sub-região;
- Confiança na Classificação, Treino, Ruído e Total;
- Apontador para o próximo NodeList da lista ligada.

Figura G.2: Nós utilizados na lista ligada.

Apêndice H

Pseudocódigo dos métodos API

Neste capítulo são apresentados os pseudocódigos dos métodos de Análise e Processamento de Imagem (API) desenvolvidos no capítulo 2.

```
1
    /**
3       INPUT: Vector Histograma
        OUTPUT : Vector Histograma Normalizado
5   **/

7
    int [] Normazlizar_Histograma(int [] hist){
9
            int count = soma(hist); // somar todos os elementos do vector
11          for(int i= 0 ; i<count ; i++){
                hist[i]= hist[i]/count;
13
            }
15      return hist;
    }
17
    /**
19      INPUT: vector de Histograma Normalizado
        OUTPUT: valor limiar K
21  **/

23  int Calcular_Otsu(int [] histNormal){
```

```
25          int Resultados_Otsu[histNormal.size] // guardar os Resultados

27          // calcular o muiT - intensidade média do conjunto
            int muiT=0;
29          calcular muiT(histNormal);

31          // Dentro do ciclo calcular os valores de omega0 , omega1, mui0 e
     mui1 e sigma
            // depois adicionar na posição k o valor de sigma calculado
33          for(int k = 0; k<histNormal.size; i++){
                calcular Omega0(k,histNormal);
35              calcular Omega1(k,histNormal);
                calcular mui0(k,histNormal,Omega0);
37              calcular mui1(k,histNormal,Omega1);
                calcular sigma(k, Omega0,Omega1,muiT,mui0,mui1);
39              Resultado_Otsu[k]=sigma;
            }
41          // retornar o índice onde ocorre o valor máximo de variância
    inter-classes
        return max_índice(Resultado_Otsu);
43  }

45


47  /**
        INPUT: Vector Histograma
49      OUTPUT: Void - Sem Retorno

51      Contrast Stretching altera os pixeis iniciais de input na imagem
    segundo uma função de transformação
        enunciada na parte Teórica do trabalho
53  **/

55  void Contrast_Stretching( int [] histNormal){

57      calcular max(histNormal);
        calcular min(histNormal);
59      for (int x = 0; x<sizex; x++){  // percorrer a imagem
            for(int y = 0 ; y<sizey ; y++){
```

```
61              int cinzento = matrix[x+y*sizex];
                matrix[x+y*sizex]= 255*((cinzento-min)/(max-min));
63
                }
65          }
        }
67

69      /**
        erosão
71          INPUT: dimensões da máscara
            OUTPUT : nova matriz , que será usada para carregar uma nova imagem
73
        **/
75
    void erosão(int janela){
77          //Criar uma Matriz com o mesmo Tamanho da Imagem , onde cada entrada
        corresponde
            // a um pixel na imagem
79      int [] Novamatriz = new int[sizex*sizey];

81          //Criar uma máscara com as dimensões janela*janela
            int[]mascara = new int[janela*janela];
83          //inicializar a máscara
            for(int i =0 , i<mascara.size; i++){
85          mascara[i]=1;
            }
87          for (int x = 0; x<sizex; x++){  // percorrer a imagem
                for(int y = 0 ; y<sizey ; y++){
89
                   if( sendo matrix[x+y*sizex] ponto central da máscara,
91                 se a máscara não estiver totalmente contida na imagem){
                        Novamatrix[x+y*sizex]=255; // passa a um ponto fundo
93                 }
                   else{
95                     // percorrer o máscara e realizar uma operação de
        convulsão
                       for (int mx =-janela/2; mx<janela/2; mx++){
97                        for(int my=-janela/2; my<janela/2; my++){
```

```
                        somar(mascara[mx+my*janela]*matrix[x+mx+(y+my)*sizex
    ])
99                      }
                    }
101                 if(soma < janela*janela){
                        Novamatrix[x+y*sizex]=255; // Classificado como ponto
    fundo
103                 }
                    else{
105                     Novamatrix[x+y*sizex]=0; // Classificado como ponto
    Corpo
                    }
107
        }
109


111     }
       }
113
            Criar uma nova imagem a partir da matriz criada
115         Guardar a referência desta nova imagem , que poderá passar por
    criar/modificar um      objeto Image ou alterar uma instância global tipo
     Image

117        /** É ACONSELHADO A UTILIZAÇÃO DO PADRÃO SINGLETON PARA ESTE
    PROCESSO
               OBJETIVO: TER APENAS UM OBJETO DO TIPO IMAGE **/
119


121 }

123 /**
    dilatação
125     INPUT: dimensões da máscara
        OUTPUT : nova matriz  que será usada para carregar uma nova imagem
127 **/


129
void dilatação(int janela){
```

```
131
133         //Criar uma Matriz com o mesmo Tamanho da Imagem , onde cada entrada
    corresponde
            // a um pixel na imagem
135     int [] Novamatriz = new int[sizex*sizey];

137         //Criar uma máscara com as dimensões janela*janela
            int[]mascara = new int[janela*janela];
139         //inicializar a máscara
            for(int i =0 , i<mascara.size; i++){
141         mascara[i]=1;
            }
143         for (int x = 0; x<sizex; x++){  // percorrer a imagem
                for(int y = 0 ; y<sizey ; y++){
145
                  if( sendo matrix[x+y*sizex] ponto central da máscara,
147               se a máscara não estiver totalmente contida na imagem){

149                   Novamatrix[x+y*sizex]=255; // passa a um ponto fundo
                      // uma possível modificação a este algoritmo é não alterar
    a natureza do pixel
151               }

153               else{
                      // percorrer o máscara e realizar uma operação de convulsão
155                   for (int mx =-janela/2; mx<janela/2; mx++){
                       for(int my=-janela/2; my<janela/2; my++){
157       somar(mascara[mx+my*janela]*matrix[x+mx+(y+my)*sizex])
                    }
159                 }
                    if(soma <1){
161                       Novamatrix[x+y*sizex]=255; // Classificado como ponto
    fundo
                    }
163                 else{
                          Novamatrix[x+y*sizex]=0; // Classificado como ponto
    corpo
165                 }
```

```
            }
167         }
        }
169
    }
171

173 /**
    Abertura
175     INPUT: dimensões da máscara
        OUTPUT : nova matriz , que será usada para carregar uma nova imagem
177 **/

179 void Abertura(int janela){

181     erosão(janela);
    dilatação(janela);
183
}
185
 /**
187     Fecho
        INPUT: dimensões da máscara
189     OUTPUT : nova matriz , que será usada para carregar uma nova image

191     **/

193 void Fecho(int janela){

195     dilatação(janela);
    erosão(janela);
197
}
199

201 /**
    Detetar a Fronteira de um Objeto
203     INPUT: dimensões da máscara de Sobel - as dimensões da máscara
   determinaram o grau de precisão nos cálculos
```

```
        OUTPUT : Nova matriz , que será usada para carregar uma nova imagem
205     CARACTERISTICAS: obter uma imagem apenas com as fronteiras dos
    Objetos

207 **/

209 void Detetar_Fronteira(int janela){

211 //Criar uma Matriz com o mesmo Tamanho da Imagem , onde cada entrada
    corresponde
        // a um pixel na imagem
213 int [] Novamatriz = new int[sizex*sizey];

215 //Criar duas máscaras Gx e Gy com as dimensões janela*janela
    int[] Gx = new int[janela*janela];
217 int[] Gy = new int[janela*janela];
    // inicializar as máscara
219 initialize(Gx);
    initialize(Gy);
221
    int Resultados [dimensão da imagem];
223 para todos os pontos da imagem{

225     derivar em ordem a x com a máscara Gx
        derivar em ordem a y com a máscara Gy
227     calcular o gradiente  no ponto |G| = |Gx| + |Gy|
    guardar um resultado numa estrutura auxiliar como Resultado[ponto] = |G|
229
    }
231
    calcular o valor médio em Resultados
233 calcular o desvio padrão com o valor médio calculado anteriormente;
    utilizar o vms = valor medio +2*desvio padrao;
235


237 para todos os pontos da imagem{

239     derivar em ordem a x com a máscara Gx
        derivar em ordem a y com a máscara Gy
```

```
241         calcular o gradiente  no ponto |G| = |Gx| + |Gy|
            if(|G|>vms){
243             novaMatriz[ponto]= 0;   // classificar o ponto como ponto
        Fronteira
            }
245         else{
                novaMatriz[ponto]=255; // classificar o ponto como ponto Fundo
247
            }
249     }

251 }

253


255


257 /**
        Algoritmo de etiquetagem
259         INPUT: Imagem binária
            OUTPUT : Mapeamento dos Corpos na Imagem
261         CARACTERÍSTICAS: Algoritmo para identificar os corpos presentes na
        imagem e as suas correspondentes coordenadas
            Solução iterativa
263
        **/
265


267 bool [] Labeling{

269     int [] novaMatriz = new int[sizex*sizey];
        /**criar uma matriz booleana Mapeamento[sizex*sizey] que será utilizada
271     para verificar se um determinado ponto já foi visitado e/ou processado**/
        bool Mapeamento = new bool[sizex * sizey];
273     int cor;

275     //instanciar uma Stack;

277     Stack myStack();
```

```
279    for(todos os pixeis na imagem){
           // se o pixel pertence ao corpo do objeto e não foi visitado
281        if(matrix[ponto]==0 && Mapeamento[ponto]){
               //colocar na pilha myStack(ponto);
283            myStack.add(ponto);
               //incrementar a cor
285            cor++;
               // começar o processo iterativo
287            while(myStack.isNotEmpty()){
                  pontoAnalise = myStack.RemoveFirstPosition();
289               /**Testar que tipo de ponto é: ponto no bordo da imagem ou no
    seu interior
                  Se pertence ao bordo da imagem, qual dos bordos pertence?
291               E agir de forma diferenciada dependendo caso**/
                  if(pontoAnalise esta na fronteira){
293                 switch(tipo de Fronteira){
     case(inferior):
295                 if(vizinhos não foram visitados e têm as mesmas
    características){
                        myStack.add(esq);
297                     myStack.add(dir);
                        myStack.add(sup);
299                 }
       case(superior):
301                  if(vizinhos não foram visitados e têm as mesmas
    características){
                        myStack.add(esq);
303                     myStack.add(dir);
                        myStack.add(inf);
305                 }
                    case(lateral esq):
307                   if(vizinhos não foram visitados e têm as mesmas
    características){
                        myStack.add(dir);
309                     myStack.add(sup);
                        myStack.add(inf);
311                 }
                    default:
```

```
313                 if(vizinhos não foram visitados e têm as mesmas
    características){
                        myStack.add(esq);
315                     myStack.add(sup);
                        myStack.add(inf);
317               }
              }
319           // colocar como visitado
              Mapeamento[pontoAnalisa]=false;
321           NovaMatriz[pontoAnalisa]= cor;

323           }
              else{
325     if(vizinhos não foram visitados e têm as mesmas características){
                     myStack.add(esq);
327                  myStack.add(sup);
                     myStack.add(inf);
329                  myStack.add(dir);
                    }
331                 Mapeamento[pontoAnalisa]= false;
                    NovaMatriz[pontoAnalisa]= cor;
333         }
    }
335        }
    }
337


339       /**  É ACONSELHADO A UTILIZAÇÃO DO PADRÃO SINGLETON PARA ESTE
    PROCESSO
             OBJETIVO: TER APENAS UM OBJETO DO TIPO IMAGE**/
341

    return: Mapeamento
343 }


345


347 /**
    INPUT: Matriz contendo as coordenadas de cada ponto do corpo
349 OUTPUT: Um vetor de Objetos do tipo Ponto, que contêm as coordenadas de
```

```
    todos os corpos ON na imagem

**/

(Ponto[]) Centro_de_Massa( int [] MatrizLocalizacao){

    criar um vetor de Pontos com dimensão grelhax*grelhay
    o Número máximo de dots presentes na imagem é grelhax*grelhay


    int coordenadaX;
    int coordenadaY;
    int Npontos=0;
    for (int i = 0; i<NúmeroMáximoDots; i++){

            coordenadaX=0;
            coordenadaY=0;

        for (int x = 0; x<sizex; x++){  // percorrer a imagem
            for(int y = 0 ; y<sizey ; y++){

                    if(MatrizLocalizacao[x+y*sizex]==i){
                        coordenadaX+=x;
                        coordenadaY+=y;
                        Npontos++;
                    }

            }
        }

        coodenadaX/=Npontos;
        coodenadaY/=Npontos;

        // CRIAR UM OBJETO DO TIPO PONTO
        Ponto pt = new ponto(coodenadaX,coodenadaY);
        VetorPontos[i]=pt;

    }
    return VetorPontos;
```

```
389 }
391
393 /**
    INPUT: Um vetor de Objetos do tipo Ponto, que contêm as coordenadas de
    todos os corpos ON na imagem
395     OUTPUT: uma lista ligada contêm objetos do tipo PontoCanto
397
    **/
399
    -(LinkedList PontoCanto)DetetarCantos(Pontos[] CentroidePontos){
401
            //Guardar INFORMAÇÕES relativos aos quatro cantos da imagem  numa
    estrutura auxiliar
403      Inf [4] CoordenadasCantos = [(0,0, "es"),(sizex,0,"ds"),(0,sizey, "ei")
    ,(sizex,sizey, "di")];
405
407         //CRIAR UMA LISTA LIGADA DE TAMANHO MÁXIMO COMPOSTA POR NÓS COM
    CAMPOS VAZIOS
            LinkedList<PontoCanto> listarPontos= new LinkedList<PontoCanto>(4);
409         // Procurar no vector de Pontos o elemento mais próximo de cada canto
     da imagem-medindo distância euclidianas
            // retornar o indice desse mesmo elemento
411
      for(int i = 0; i<4; i++){
413        int indexMaisProximo = maisProximoCanto(CoordenadasCantos[i],
    CentroidePontos);
           // Criar um Objeto PontoCant0
415        PontoCanto pj = new PontoCanto(CoordenadasCantos[i],CentroidePontos[
    indexMaisProximo]);
           //adicionar a lista ligada criada anteriormente
417        listarPontos.add(pj);
            }
419         return listarPontos;
```

```
}

/** Input: Uma LinkedList de Objetos tipo PontoCantos, as dimensões da grelha
     (x,y)
    Output: Coordenada dos dots na Imagem
    Projetar a grelha sobre a imagem

    UTILIZAR O MÉTODO DOS MÍNIMOS QUADRADOS
            COEFICIENTES DA TRANSFORMADA DADO POR : (At* A)-1 At*l
    **/
-(double[]) ProjetarGrelha(LinkedList<PontoCanto> listarPontos, int nx, int
    ny){


 int [] A = new int [(1,1,1,0,0,0,0),(0,0,0,1,1,1),(1,nx,1,0,0,0),(0,0,0,1,nx
    ,1),(1,1,ny,0,0,0),(0,0,0,1,1,ny),(1,nx,ny,0,0,0),(0,0,0,1,nx,ny)];

    // transformar a lista lista na matriz das observações l
    double [] l = new double[8];

    ListNode first = listarPontos.getFirstNode();
    ListNode tmp = first;
    int index = 0;
    while(tmp!=NULL){
        Ponto ty =listarPontos.RemoveAtPosition(index);
        l.add(ty.px);
        l.add(ty.py);
        }

    Realizar o seguinte operação de matrizes (At* A)-1 At*l
    Onde At representa a transposta da matriz A
    -1  A operação de inversão
    Guardar o resultado numa matriz tipo:
    int [] MatrizCoeficientes = new int[8]

    //Com os coeficientes Calculados, construir uma matriz com o dobro das
    dimensões da grelha
    // em que cada entrada nesta matriz, representa as coordenadas de
```

```
    determinado ponto na grelha virtual.
455     // a sua estrutura segue o mesmo exemplo que A.
    double [][] CoordenadasGrelha = new double[ny*nx*2][nx];
457     for(int y=1 ; y<ny; y++){
        for/int x = 1 ; x<nx; x++){
459             CoordenadasGrelha[index][0]=1;
                CoordenadasGrelha[index][1]=x;
461             CoordenadasGrelha[index][2]=y;
                CoordenadasGrelha[index+1][3]=1;
463             CoordenadasGrelha[index+1][4]=x;
                CoordenadasGrelha[index+1][5]=y;
465             index+=2;
            }
467
    }
469
    double []ResultadosFinais = CoordenadasGrelha*MatrizCoficientes;
471
    return ResultadosFinais;
473 }


475
    /**
477         Input: Imagem original, um vetor com a posição dos dotblot na imagem

479     CARACTERISTICAS: Avaliar o Ruido presente numa Imagem
    **/
481


483 -(void) AvaliarRuido(double[]PosicaoDots){
        // criar uma LinkedList com objetos do tipo NodeList;
485     LinkedList<NodeList> lmp = new LinkedList<NodeList>();

487     // criar NodeList apenas com  as informações relativamente as
    coordenadas na imagem
        for(int i=0; i<PosicaoDots; i+=2){
489         NodeList xp = new NodeList(PosicaoDots[i],PosicaoDots[i+1]);
            lmp.add(xp);
491
```

```
493         }

495         para todos os pontos na imagem{

497           ver qual é o dot que se encontra mais próximo deste ponto
            para tal pode ser efetuado uma procura exaustivo na listaLigada ou
usar uma Matriz de Mapeamento
499           Cada entrada na matriz de mapeamento corresponde a um índice na
lista ligada/ ou vetor dependendo da estrutura utilizada
 Identificado qual é o dot,seguidamente será necessário identificar em
que região se encontra o ponto relativamente ao dot
501
    *****AVANÇAR A LISTA LIGADA ATÉ AO DOT EM QUESTÃO*****
503                 if(encontra-se no centro){
                        CURSOR.intensidadeZona[0]+=Intensidade(ponto);
505                     // Intensidade(ponto) corresponde a intensidade no
ponto
                        CURSOR.Número_Pontos_Zona[0]++;
507                 }
                    else if(no 1 quadrante){
509                     CURSOR.intensidadeZona[1]+=Intensidade(ponto);
                        // Intensidade(ponto) corresponde a intensidade no
ponto
511                     CURSOR.Número_Pontos_Zona[1]++;
                    }
513                 //AVALIAR PARA TODOS OS QUADRANTES
                    ::::::::::::
515                 ::::::::::::
                    ::::::::::::
517                 else{ //Caso se encontre no Último quadrante
                        CURSOR.intensidadeZona[8]+=Intensidade(ponto);
519                     // Intensidade(ponto) corresponde a intensidade no
ponto
                        CURSOR.Número_Pontos_Zona[8]++;
521                 }
        }
523     //CALCULAR A INTENSIDADE DE CADA ZONA
         for(cursor = first; cursor!=null;cursor =cursor.next){
```

```
525                 for(int i= 0; i<9; i++){
                        Cursor.intensidadeZona[i]/=Cursor.Número_Pontos_Zona;
527
                    }
529
        }
531     // por fim calcular o ruido em cada dot
        int a = 30;
533     int b = 100;


535
        for(cursor = first; cursor!=null;cursor =cursor.next){
537         // avaliar o ruido em cada sub-região
            // o ciclo não começa em 0, porque o 0 corresponde a região
    central
539         for(int i = 1; i<9; i++){
               // existem três niveis de ruido
541            if(abs(cursor.intensidadeZona[i]-cursor.intensidadeZona[0])<a)
    {
                 cursor.RuidoZona[i]=0;
543            }
               else if(abs(cursor.intensidadeZona[i]-cursor.intensidadeZona
    [0])>a
545             && abs(cursor.intensidadeZona[i]-cursor.intensidadeZona[0])<b
    ){

547                cursor.RuidoZona[i]=1;
               }
549            else{
                   cursor.RuidoZona[i]=2;
551                }
            }
553
        }
555
    PARA CALCULAR AGORA O RUÍDO TOTAL ASSOCIADO A CADA DOTS, BASTA COLOCAR O
    CURSOR NESSE DOT
557 E SOMAR TODOS OS ELEMENTOS DO SEU VETOR RUIDOZONA;
```

```
559
561
    NOVAMENTE DEVESSE TER APENAS UMA LINKEDLIST , QUE DEVERÃO SER GLOBAL À
    CLASSE, OU SEJA ACESSÍVEL EM QUALQUER PONTO DA CLASSE
563 ESTA LISTA-LIGADA CONTÊM INFORMAÇÕES GLOBAIS QUE DEVEM SER PRESERVADOS
    COM O DECORRER DA APLICAÇÃO
}
565
567
569 /** INPUT: Lista ligada criada anteriormente,e um array de String contendo o
    nome de cada dot
    Avaliar a Probabilidade das Marcas estarem ON
571
**/
573
575
-void Avaliar_Probabilidade(String [] nomeDot, int nx, int ny){
577
    Primeiro inserir o nome de cada dot na nossa lista ligada
579
581 double IntensidadeFundo;
    double [nx*ny] IntensidadeNormalizada;
583
    //percorrer a lista ligada e calcular a intensidade de fundo de cada
    marca
585
    for(int i = 0; i<nx*ny; i++){
587    IntensidadeFundo =somar(cursor.IntensidadeZona)/8; // não somar a
    posição 0
       IntensidadeNormalizada = 1-cursor.IntensidadeZona[0]/IntensidadeFundo;
589
591     cursor= cursor.next;
    }
```

```
593
    CARREGAR A BASE DE DADOS E CALCULAR médias das intensidades normalizadas
595 DE CADA MARCA e a média das intensidade Normalizadas das marcas OFF

    // por fim percorrer a lista ligada e calcular a Probabilidade de cada
597 marca estar On

    int i = 0;
599 for(cursor = first; cursor!=null;cursor =cursor.next){
        // chamada de uma função auxiliar para calcular a equação dada na
601 parte Teórica
        cursor.Probabilidade_ON = Calcular_Probabilidade(
    IntensidadeNormalizada[i],mediaOff,mediaMarca);
603     i++;
    }
605
}
607
/**
609 INPUT: Lista Ligada
    Caracteristicas: Avaliar a confiança associada a cada Marca
611 **/


613


615 -void Avaliar_confiança(){


617
    /**Para o calculo da confiança no treino percorrer a lista ligada
619 e para cada elemento da lista calcular a média das probabilidades de
    todos as marcas daquele tipo
    usadas na fase de treino**/
621


623 for(cursor = first; cursor!=null;cursor =cursor.next){
        cursor.confiança_Treino = Calcular_Media_Tipo(cursor.Tipo_No);
625 }


627
```

```
    //Para o calculo da confiança no Ruido usar a equação na parte Teórica
629
    for(cursor = first; cursor!=null;cursor =cursor.next){
631     cursor.confiança_Ruido = Calcular_Confianca_Ruido(soma(cursor.
    RuidoZona));
    }
633
    // para calcular a confiança na Classificação usar a equação na parte
    Teórica
635
    for(cursor = first; cursor!=null;cursor =cursor.next){
637     cursor.confiança_Classificação = Calcular_Confianca_Classificação(
    cursor.Probabilidade_ON);
    }
639
    //por fim para calcular a confiança total associada a cada Marca
641 for(cursor = first; cursor!=null;cursor =cursor.next){
        cursor.confiança_Total  = cursor.confiança_Classificação*(cursor.
    confiança_Treino*cursor.confiança_Ruido)/2;
643 }


645


647 /**
    INPUT: Imagem binária da folha
649
    Caracteristicas: Excentricidade pode ser vista como uma medida de quanto
    uma determinada forma se desvia da forma circular
651
    Output: Um valor compreendido entre [0,1], sendo 0 a forma circular pura
    e 1 a forma parabólica.
653


655 **/


657
-int Excentricidade (){
659 /**criar uma matriz booleana Mapeamento[sizex*sizey] que será utilizada
    para verificar se um determinado ponto já foi visitado e/ou processado**/
```

```
661     bool Mapeamento = new bool[sizex * sizey];

663     //Calcular o Centroide da imagem binária
        Centroide myCentroide = obter_Centroide_Folha();
665     //Com as coordenadas Centroide criar o vector mui
        MatrizXf mui(2,1);
667     mui(0,0) = myCentroide.x
        mui(1,0)=  myCentroide.y
669     //Inicializar a matrix das Covariancias.
        MatrizXf Matriz_Covariancias(2,2)=0;
671     // Uma matriz de Booleana como estrutura auxiliar no Processo de Vizita
        int count=0;
673

675     //instanciar uma Stack;

677     Stack myStack();

679     for(todos os pixeis na imagem){
            // se o pixel pertence ao corpo do objeto e não foi visitado
681         if(matrix[ponto]==0 && Mapeamento[ponto]){
                //colocar na pilha myStack(ponto);
683             myStack.add(ponto);

685             // começar o processo iterativo
                while(myStack.isNotEmpty()){
687                 pontoAnalise = myStack.RemoveFirstPosition();
                    /**Testar que tipo de ponto é: ponto no bordo da imagem ou no
    seu interior
689                 Se pertence ao bordo da imagem, qual dos bordos pertence?
                    E agir de forma diferenciada dependendo caso**/
691                 if(pontoAnalise está na fronteira){
                        switch(tipo de Fronteira){
693         case(inferior):
                        if(vizinhos não foram visitados e têm as mesmas
    características){
695                         myStack.add(esq);
                            myStack.add(dir);
697                         myStack.add(sup);
```

```
                    }
699                 case(superior):
                       if(vizinhos não foram visitados e têm as mesmas
    características){
701                      myStack.add(esq);
                         myStack.add(dir);
703                      myStack.add(inf);
                       }
705                 case(lateral esq):
                       if(vizinhos não foram visitados e têm as mesmas
    características){
707                       myStack.add(dir);
                          myStack.add(sup);
709                       myStack.add(inf);
                       }
711                 default:
                       if(vizinhos não foram visitados e têm as mesmas
    características){
713                       myStack.add(esq);
                          myStack.add(sup);
715                       myStack.add(inf);
                       }
717             }
                 // colocar como visitado
719              Mapeamento[pontoAnalisa]=false;
                 MatrixXf ponto(2,1);
721             ponto(0,0)=pontoAnalisa.x;
                ponto(1,0)=pontoAnalisa.y;
723             //REALIZAR A SEGUINTE OPERAÇÕES MATRICIAL onde T simboliza a
    operação Transposta
                MatrixXf Mf =(ponto-mui)*(ponto-mui)T
725             // Somar a Matriz_Covariancia
                Matriz_Covariancia +=Mf;
727             //incrementar o numero de pontos
                count++;
729           }
              else{
731              if(vizinhos não foram visitados e têm as mesmas
    características){
```

```
                myStack.add(esq);
733             myStack.add(sup);
                myStack.add(inf);
735             myStack.add(dir);
              }
737           Mapeamento[pontoAnalisa]= false;
              MatrixXf ponto(2,1);
739           ponto(0,0)=pontoAnalisa.x;
              ponto(1,0)=pontoAnalisa.y;
741           //REALIZAR A SEGUINTE OPERAÇÕES MATRICIAL onde T simboliza a
    operação Transposta
              MatrixXf Mf =(ponto-mui)*(ponto-mui)T
743           // Somar a Matriz_Covariancia
              Matriz_Covariancia +=Mf;
745           //incrementar o numero de pontos
              count++;
747                 }
          }
749     }

751     MatrizCovariancia/=count;
        //Calcular os Valores próprios da matriz Covariancia
753     VProprios = valor_Proprio(Matriz_Covariancia);
        //Calcular a excentricidade através dos valores próprios
755     int excentricidade = sqrt(1-(VProprio.a^2)/(VProprio.b^2));
        Guardar os valores Próprios numa estrutura auxiliar
757     que seja global na Classe a implementar este método

759     return excentrcidade;


761


763 }


765

/**
767     INPUT: Imagem binária da folha e os semi-eixos da Elipse Calculados da
    função Excentricidade
    Características: Calculo da distância de escape
```

```
769     Output: Um inteiro compreendido entre [0,1]. O máximo é atingido na
        região circular.
**/
771

-int  Elongation(){

775
    Construir uma Árvore binária de Pesquisa
777     onde cada nó desta árvore é um ponto pertencente ao Bordo da Folha
    cada nó têm uma chave única construída através da seguinte função:
779     double chave = (ponto.x^2+ponto.y^2)cos(atan2(y,x));

781     for(todos os pixeis na imagem){
            // se o pixel pertence ao corpo do objeto e não foi visitado
783         if(matrix[ponto]==0 && Mapeamento[ponto]){

785             Calcular a chave neste Ponto
                Usar a chave para descobrir qual é o ponto mais próximo no Bordo
        da Folha
787             Usando a árvore binária construída anteriormente
                Calcular a distancia daquele Ponto ao Bordo da Folha
789             Guardar numa estrutura auxiliar DistanciaEscape

791         }

793     }
    // Calcular o maior elemento no Vector DistanciaEscape
795     int distEscape = maiorElemento(DistanciaEscape);

797

    int Elongation = 1-2*distEscape/EixoMenorDaElipse;
799     return Elongation;

801 }

803

/**
805     INPUT: Semi-eixos da elipse calculados na função excentricidade
```

```
807
    Output: Valores perto de 0 indicam formas alongadas.
809


811


813 **/
-int AspectRatio(){
815


817     return EixoMenorDaElipse/EixoMaiorDaElipse;
819


821
}
```

H.1 Errata

Na página 32 a equação 3.6 está errada, e deveria ser:

$$P(\overline{I_\tau}) = \begin{cases} 0, & \text{if } \overline{I_\tau} < \overline{\mu_{OFF}}; \\ \frac{\arctan(u)-\arctan(-2)}{\arctan(8)-\arctan(-2}, & \text{if } \overline{\mu_{OFF}} \leq \overline{I_\tau} \leq \overline{\mu_{I_\tau}}; \\ 1 & \text{if} \overline{I_\tau} > \overline{\mu_{I_\tau}}. \end{cases} \tag{H.1}$$

Na página 33 a equação de u está errada e deveria estar:

$$u = -2 + \frac{10 \times (\overline{I_\tau} - \overline{\mu_{OFF}})}{\Delta}.$$

Na mesma página (1 parágrafo linha 5) deveria constatar $P(\overline{I_\tau})$ em vez de $P(\overline{\mu_{I_\tau}})$

Referências

[1] Cristina Maria Ribeiro Martins Pereira Caridade. *Desenvolvimento de algoritmos e técnicas de extracção de informação em imagens biomédicas*. PhD thesis, FCUP, January 2012. ix, xxi, xxii, 1, 23, 24, 25, 29, 32, 33, 34, 36, 37

[2] Pedro Filipe Barros Silva. Development of a system for automatic plant species recognition, 2013. x, xi, 1, 21, 22, 55, 56

[3] *Hello Android, Introducing Google?s Mobile Development Platform*. Pragmatic Bookshelf; Third Edition edition, August 4, 2010. xxiii, 46, 47, 67, 68, 69, 70

[4] MAC Developer Library. About the ios technologies, March 3, 2013. xxiii, 71

[5] *Desenvolvimento em ios iphone, ipad, e ipod Touch - Curso Completo*. FCA-Editora de Informática, Lda, January 13, 2013. xxiii, 70, 72, 73, 74, 75

[6] Android Developers. Camera.picturecallback, January 19, 2013. xxiv, 79, 80, 81

[7] *iOS6 Programming Cookbook*. O'Reilly Media; 1 edition, December 25, 2012. xxiv, 83, 85

[8] Indhuja. ios 6 iphone camera application, April 5, 2013. xxiv, 87, 88, 89

[9] *Digital Image Processing*. Prentice Hall; 3 edition, August 31, 2007. 5, 6, 8, 9, 10, 11, 12, 14, 15, 16, 19, 20, 21

[10] *The Image Processing Handbook*. CRC Press; 6 edition, April 7, 2011. 5, 9, 13, 14

[11] Miguel Coimbra. Visão computacional, October 4, 2012. 6, 14, 15, 16

[12] *Introduction to Algorithms*. The MIT Press; third edition edition, July 31, 2009. 17

[13] Wikipédia. Rgb, June 16, 2013. 20

[14] *Android Application for Dummies.* For Dummies; 2 edition, October 23, 2012. 67

[15] *Objective-C for Dummies.* For Dummies; 1 edition, October 23, 2012. 67

[16] *Beginning iOS6 Development Exploring the iOS SDK.* Apress; 1 edition, December 7, 2011. 72, 73

[17] *Sams Teach Yourself Ios 5 Application Development in 24 Hours.* Sams Publishing; 3 edition, January 9, 2012. 72, 73, 91

[18] Android Developers. Intent, January 17, 2013. 46

[19] *C Programming a modern Approach.* W. W. Norton and Company; 2 edition, April 19, 2008. 48, 57, 95

[20] Android Developers. Camerapreview, January 10, 2013. 77, 78, 79

[21] Ray Wenderlich. Core graphics tutorial: Patterns, April 19, 2013. 91

[22] stackoverflow. How to animate cashapelayer path and fillcolor, May 23,2013. 91

[23] MAC Developer Library. Cashapelayer class reference, May 24,2013. 91, 92

[24] Ole Begemann iOS Development. Animating the drawing of a cgpath with cashapelayer, May 29, 2013. 91

[25] Think Build. Playing around with core graphics, core animation and touch events (part 2), May 25,2013. 91, 92, 93