Instituto Politécnico de Castelo Branco
Escola Superior de Tecnologia

Reconhecimento de Imagens por Câmara de Smartphone em Tempo Real

Fernando Jorge Fernandes Geraldes

Relatório apresentado ao Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Desenvolvimento de Software e Sistemas Interativos, realizada sob a orientação científica do Prof. Doutor José Carlos M. M. Metrôlho, Professor Adjunto da Unidade Técnico-Científica de Informática da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

2012

Agradecimentos

Em primeiro lugar quero agradecer a minha esposa e ao meu filho, por terem compreendido a falta de tempo que passei com eles para a realização deste projeto, e pelo apoio que me deram de incentivo para chegar ao fim do mesmo.

Em segundo lugar quero agradecer ao meu orientador Prof. Doutor José Carlos M. M. Metrôlho pela sua disponibilidade para orientar o projeto, mesmo com o pouco tempo que tem por causa das funções que exerce na escola.

Quero agradecer também á Empresa TIMWE onde trabalho, por ter disponibilizado algum tempo para esta investigação no horário de trabalho. Ao meu chefe Paulo Salgado pelo incentivo e pela ajuda para a realização da investigação.

Agradeço também ao Prof. Doutor Hugo Proença da UBI (Universidade Da Beira Interior) pela ajuda e tempo que disponibilizou para que esta investigação pudesse ser concluída com êxito.

Por fim, agradeço a toda a minha família que sempre me incentivou para a realização do mestrado e para a realização deste projeto.

A todos o meu muito Obrigado!

RECONHECIMENTO DE IMAGENS POR CÂMARA DE SMARTPHONE EM TEMPO REAL

Fernando Jorge Fernandes Geraldes

Resumo - O reconhecimento de imagens nos dias de hoje é uma prática utilizada em vários âmbitos, tais como: reconhecimento de face, reconhecimento de código de barras e QR-Code, análise de imagens médicas, entre outras.

Existem também já trabalhos que pretendem explorar o reconhecimento de imagens em tempo real através da câmara de *smartphones*, existindo já alguns protótipos. No entanto ainda nada existe em termos de produto comercial, e o que existe neste âmbito são aplicações que reconhecem imagens mas não em tempo real, e sim a partir da imagem captada por fotografia.

Com este projeto pretende-se que o *smartphone* seja capaz de reconhecer imagens em tempo real, para que possa ser útil em várias aplicações destinadas ao público em geral, mas com o objetivo fundamental que é o de poder reconhecer as imagens e poder identificá-las através de voz para ajudar pessoas invisuais no reconhecimento de objetos, ou acrescentar informação às imagens para complementar a visão humana.

A implementação da aplicação consiste em desenvolver uma API para a segmentação e análise da imagem (desenvolvida em linguagem C e C++) usando bibliotecas nativas do NDK Android, OpenCV para processar imagens, que está explicada na primeira parte desta dissertação. A segunda parte é composta pela implementação de uma aplicação Android em JAVA usando o SDK, e a integração da API desenvolvida na primeira parte com a aplicação, para que o objetivo principal seja alcançado.

Para alcançar os objetivos, utilizamos várias técnicas de equalização, segmentação e reconhecimento de imagens, e também foi construída uma base de conhecimento inicial para o reconhecimento de objetos. Esse conhecimento é mais tarde alargado com a experiência e utilização do *software* como a recolha de imagens não conhecidas inicialmente. O objetivo é ter uma API que reconheça qualquer objeto previamente inserido na base de conhecimento.

Os resultados obtidos neste projeto foram muito satisfatórios, o *software* desenvolvido respondeu positivamente a 98% dos testes realizados em 100 imagens representativas dos objetos treinados. É de salientar que o tempo médio de resposta para o reconhecimento foi 1,5s o que correspondeu as nossas expectativas para este projeto. Os testes foram realizados em vários *smartphones* e em todos eles o *software* desenvolvido foi executado com sucesso e o reconhecimento dos objetos estudados foi conseguido em todos eles.

RECOGNITION OF IMAGES BY CAMERA SMARTPHONE IN REAL TIME

Fernando Jorge Fernandes Geraldes

Abstract - The image recognition nowadays is a practice widely used in various fields for face recognition, recognition of barcodes and QR-Code, image analysis means, among others.

There are also now works who want to explore the recognition of images in real time via the camera of smartphones, there are already some prototypes. However there is still nothing in terms of commercial product, and in this context there are applications that recognize images but not real time, but from the image captured by photography.

With this project it is intended that the smartphone is capable of recognizing images in real time, so it can be useful in many applications aimed at the general public, but with the ultimate goal which is to be able to recognize the images and be able to identify them through voice to help blind people in the recognition of objects, or append information to images to complement human vision.

The implementation of the application is to develop an API for image segmentation and analysis (developed in C and C + +) libraries using native Android NDK, OpenCV for image processing, which is explained in the first part of this dissertation. The second part consists of the implementation of an application in Java using the Android SDK and API integration developed in the first part with the application, so the main goal is achieved.

To achieve the objectives, we use various techniques of equalization, segmentation and recognition of images, and also built a knowledge base for the initial recognition of objects. This knowledge is later extended with the experience and use of software as a collection of images is not known initially. The goal is to have an API that recognizes any object previously inserted in the knowledge base.

The results of this project were very satisfactory, software developed positively responded to 98% of the tests on 100 images representing objects trained. It is noteworthy that the average response time for the recognition was 1.5 s which corresponded to our expectations for this project. The tests were performed on various smartphones and all the software they developed was successful and recognition of objects studied was achieved in all of them.

ÍNDICE

AGRADECIMENTOS II
RESUMO III
ABSTRACT IV
LISTA DE ACRÓNIMOS IX
CAPÍTULO 1 1
INTRODUÇÃO 1
CAPÍTULO 2 7
ESTADO DA ARTE 7
CAPÍTULO 3 12
MÉTODOS ESTUDADOS 12
3.1 MÉTODOS BASE E DE PRÉ-PROCESSAMENTO 12
3.1.1 Conversão da Imagem para Tons de Cinza 12
3.1.2 Histograma da Imagem em Tons de Cinza 13
3.1.3 Operador Gradiente 13
3.2 SEGMENTAÇÃO 14
3.2.1 Segmentação por K-means 15
3.2.2 Segmentação por Watershed 16
3.2.3 Segmentação por Region Growing 17
3.2.4 Deteção de Contornos por Operador de Roberts 18
3.2.5 Deteção de Contornos Pelo Operador de Prewitt 19
3.2.6 Deteção de Contornos Pelo Operador de Sobel 20
3.2.7 Deteção de Contornos Pelo Operador de Laplace 20
3.2.7 Deteção de Contornos Canny 21
3.2.7 Deteção de características 24
3.3 MÉTODOS DE TREINO E RECONHECIMENTO DE IMAGENS 26
3.3.1 Algoritmo AdaBoost 26
3.3.2 Framework Viola–Jones 29
3.3.3 Ficheiro “Haar Cascade Classifier” 32
CAPÍTULO 4 37
DESENVOLVIMENTO E IMPLEMENTAÇÃO 37
4.1 JNI, NDK E SDK ANDROID 37
4.2 SISTEMA PROPOSTO 39
4.2.1 Primeiro Método Estudado 39
4.2.2 Segundo Método Estudado 45
CAPÍTULO 5 51
5.1 DESEMPENHO DO SISTEMA 51
CAPÍTULO 6 59
CONCLUSÕES 59
DIFICULDADES ENCONTRADAS 60
TRABALHO FUTURO 61
REFERÊNCIAS 62
REFERÊNCIAS BIBLIOGRÁFICAS 63

Índice de Figuras

Figura 1 - Vista em corte do olho humano. 2
Figura 2 - Um Sistema de Visão Artificial (SVA) e suas principais etapas. 4
Figura 3 - Exemplo de etiquetas reconhecidas pela API ARToolKit. 9
Figura 4 – Exemplo de conversão da Imagem para Tons de Cinza 12
Figura 5 - Imagem com o respetivo Histograma 13
Figura 6 - Imagem original (esquerda) e após aplicação do algoritmo K-means (direita). 16
Figura 7 - Segmentação por Watershed. 17
Figura 8 - Segmentação por Region Growing. 18
Figura 9 - Deteção de contornos pelo Operador de Roberts. 19
Figura 10 - Deteção de contorno pelo operador Prewitt. 19
Figura 11 - Deteção de contornos pelo operador Sobel. 20
Figura 12 - Deteção de contorno pelo operador Laplace. 21
Figura 13 - Deteção de contorno pelo operador Canny. 23
Figura 14 - Deteção de características 25
Figura 15 - Representação do algoritmo AdaBoost. 27
Figura 16 - Resultado do algoritmo AdaBoost[Jiri Matas and JanSochman AdaBoost 2009][17] 28
Figura 17 - Gráfico temporal do algoritmo AdaBoost 29
Figura 18 – As Características Básicas do Tipo Haar. 30
Figura 19 - Arquitetura de cascada 31
Figura 20 - Pasta para o treino e criação de ficheiros *"Haar Cascade Classifier"*. 32
Figura 21 - Aspeto do ficheiro TXT das imagens positivas. 33
Figura 22 - Aspeto do ficheiro TXT das imagens negativas, com as referências para essas imagens .. 33
Figura 23 - Janela de treino. 35
Figura 24 - Exemplo de um ficheiro XML de treino de imagens 35
Figura 25 - Contribuição do algoritmo *AdaBoost*. 36
Figura 26- Arquitetura do sistema operativo Android. 37
Figura 27 - Arquitetura do JNI. 38
Figura 28 - Caso de uso para o método apresentado 41
Figura 29 - Imagem com as diferentes regiões após execução do algoritmo K-means. 42
Figura 30 - Deteção de contorno Canny 42
Figura 31 - Resultado da implementação do passo em cima descrito 43
Figura 32 - Resultado do passo anterior. 43
Figura 33 - Imagem resultante da implementação do algoritmo descrito em cima 44
Figura 34 - Distâncias que caracterizam a matriz do objeto ..... **Error! Bookmark not defined.**
Figura 35 - Deteção de características do objeto 44
Figura 36 - Reconhecimento do objeto 45

Figura 37 - Caso se uso para o método apresentado 46
Figura 38 - Exemplos de imagens para treino e respetivo reconhecimento (Imagens Positivas) 47
Figura 39 Exemplos de imagens fotografadas para treino e respetivo reconhecimento 47
Figura 40 - Exemplo de imagens no ficheiro vec 48
Figura 41 - Resultado do treino da imagem. 48
Figura 42 – Reconhecimento de múltiplos objetos 49
Figura 43 - Exemplos de imagens reconhecidas 51
Figura 44 - Exemplo de reconhecimento com variação da iluminação. 53
Figura 45 - Exemplo de reconhecimento em imagens virtuais. 54
Figura 46 - Exemplo de reconhecimento de uma mão real. 55
Figura 47 - Exemplo do reconhecimento de imagens num fundo aleatório. 55
Figura 48 – Reconhecimento da mão em imagem real. 56
Figura 49 – Reconhecimento da mão em imagem virtual 57
Figura 50 – Reconhecimento de corpo humano inteiro, parte de cima do corpo humano e parte de baixo do corpo humano 57
Figura 51 – Reconhecimento da face com varias expressões, por dois ficheiros de classificação deferentes. 57

Índice de Tabelas

Tabela 1 - Comparação entre o sistema visual humano e um sistema de visão artificial 3
Tabela 2 - Comparação entre soluções apresentadas em cima e a solução proposta neste projecto. ... 10
Tabela 3 - Tabela comparativa dos métodos estudados 49
Tabela 4 - Tabela com a variação temporal dos vários critérios escolhidos para a avaliação do sistema 52
Tabela 5 - Intervalos de distância de reconhecimento para objectos de vários tamanhos 53
Tabela 6 - Classificação da performance em relação aos critérios descritos em cima. 56

Lista de acrónimos

API – *Application Programming Interface*

NDK – *Native Development Kit*

SDK – *Software Development Kit*

SVA – Sistema de Visão Artificial

BI – Bilhete de Identidade

JDK – *Java Develop Enviorment*

JRE – *Java Runtime Enviorment*

JNI – *Java Native Interface*

ADT – *Android Development Tools*

IDE – *Integrated Development Environment*

OpenCV – *Open Source Computer Vision*

DLL - *Dynamic Linked Library*

IPL - *Image Processing Library*

AB - *AdaBoost*

Capítulo 1

Introdução

A área de processamento de imagens é alvo de um crescente interesse por permitir explorar um grande número de possíveis aplicações. Segundo *John C. Russ* [1] esta área pode ser dividida em duas categorias bem distintas: por um lado, aumentar a interpretação humana de uma imagem real e por outro lado, permitir a análise automática por computador de informações extraídas de uma imagem.

Uma das primeiras aplicações na primeira categoria data no começo deste século, onde se estudavam formas de melhorar a qualidade de impressão de imagens digitalizadas transmitidas através do sistema "*Bartlane*" de transmissão de imagens por cabo submarino entre Londres e Nova Iorque [1]. Os primeiros sistemas "*Bartlane*", no início da década de 20, codificavam uma imagem em cinco níveis de intensidade distintos. Esta capacidade seria aumentada, já em 1929, para 15 níveis, ao mesmo tempo em que era desenvolvido um método mais complexo de revelação de filmes através de feixes de luz modulados por uma fita que continha informações codificadas sobre a imagem. Mas o grande impulso para a área de Processamento de Imagem ocorreria cerca de três décadas mais tarde, com o surgir dos primeiros computadores digitais e o início do programa espacial norte-americano.

O uso de técnicas computacionais de aperfeiçoamento de imagens despoletou no "Jet Propulsion Laboratory (Pasadena, California - EUA) em 1964" [1], quando imagens da lua transmitidas por uma sonda "Ranger2" eram processadas por computador para corrigir vários tipos de distorção inerentes à câmara de TV presente na sonda [2]. Estas técnicas serviram de base para métodos elaborados de realce e restauração de imagens de outros programas espaciais posteriores, como as expedições tripuladas da série Apollo, por exemplo.

A área de processamento de imagens apresenta uma significativa evolução e as aplicações estão presentes em diversos ramos da atividade humana. Na medicina, o uso de imagens no diagnóstico médico tornou-se vulgar e os avanços em processamento de imagens vêm permitindo tanto o desenvolvimento de novos equipamentos quanto a maior facilidade de interpretação de imagens produzidas por equipamentos mais antigos, como por exemplo o de raio X. Na Biologia, a capacidade de processar automaticamente imagens obtidas de microscópios, para por exemplo contar o número de células de um certo tipo presentes em uma imagem, facilita a execução de tarefas laboratoriais com alto grau de precisão e repetibilidade [2]. O processamento e a interpretação automática de imagens captadas por satélites auxiliam os trabalhos nas áreas de Geografia, Sensoriamento Remoto, Geoprocessamento e Meteorologia, entre outras. Técnicas de

restauração de imagens ajudam arqueologistas a recuperar fotografias contendo artefactos. O uso de robôs com visão artificial em tarefas tais como controlo de qualidade em linhas de produção aumenta a cada ano que passa.
Inúmeras áreas tão distintas como a Astronomia ou a Segurança são beneficiadas com os avanços nas áreas de processamento de imagens e visão por computador.

Sendo este um projeto em que o processamento de imagem está presente, importa considerar também a visão humana, de resto algo que é comum quando se lê bibliografia desta área. Em [2] é possível encontrar uma descrição detalhada sobre visão humana elo que decidi não abordar profundamente esse aspeto neste relatório mas também não quis deixar de o referir. A Figura 1 expressa a complexidade do sistema de visão humana e é devidamente explicada na fonte que referi.

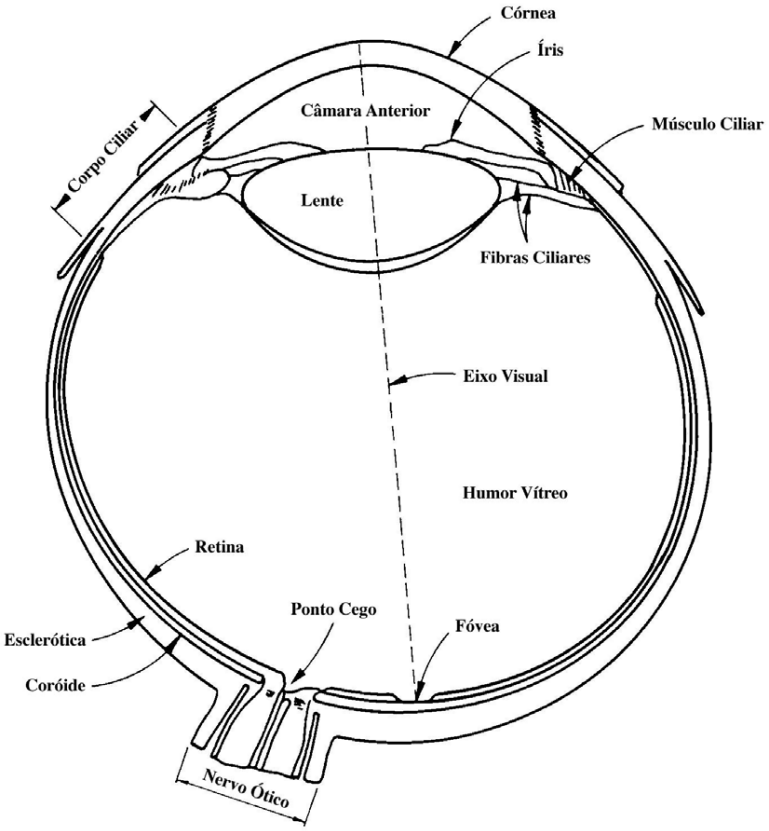

Figura 1 - Vista em corte do olho humano (adaptado de Gonzalez, R.C., Woods, R.E., Digital Image Processing, Addison-Wesley, 1992) [2].

Quando nos debruçamos sobre a complexidade da visão humana, podemos constatar três características do processo de perceção, que são [2]:

- Uma base de dados muito rica;
- Elevada velocidade de processamento;
- Capacidade de trabalhar sob condições diversas.

O avanço na tecnologia de dispositivos de armazenamento e o surgimento de novas CPUs e arquiteturas computacionais cada vez mais céleres, com elevado grau de paralelismo, permite-nos ter condições cada vez melhores para desenvolver as duas primeiras características. O grande desafio permanece, como fazer com que os sistemas de visão artificial trabalhem em diferentes condições de luminosidade, contraste, posicionamento relativo dos objetos em uma imagens sem perder a capacidade de interpretar a mesma de forma idêntica à capacidade que temos de reconhecer um amigo com relativa facilidade, independentemente de ele estar com ou sem óculos, ter deixado crescer a barba ou estar num local onde não dispomos de outra imagem senão a vista de perfil e onde as condições de luminosidade não são as melhores.

Tabela 1 - Comparação entre o sistema visual humano e um sistema de visão artificial.

	Sistema visual humano	Sistema de visão artificial
Espectro	Limitado à faixa de luz visível (300 nm a 700 nm) do espectro de ondas eletromagnéticas.	Pode operar em praticamente todo o espectro de radiações eletromagnéticas, dos raios X ao infravermelho.
Flexibilidade	Extremamente flexível, capaz de se adaptar a diferentes tarefas e condições de trabalho.	Normalmente inflexível apresenta bom desempenho somente na tarefa para a qual foi projetado.
Habilidade	Pode estabelecer estimativas relativamente precisas em assuntos subjetivos.	Pode efetuar medições exatas, baseadas em contagem de pixéis e, portanto, dependentes da resolução da imagem digitalizada.
Cor	Possui capacidade de interpretação subjetiva de cores.	Mede objetivamente os valores das componentes R, G e B para determinação de cor.
Sensibilidade	Capaz de se adaptar a diferentes condições de luminosidade, características físicas da superfície do objeto e distância ao objeto. Limitado na distinção de muitos níveis diferentes de cinza, simultaneamente.	Sensível ao nível e padrão de iluminação, bem como à distância em relação ao objeto e suas características físicas. Pode trabalhar com centenas de tons de cinza, conforme projeto do digitalizador.
Tempo de resposta	Elevado, da ordem de 0,1 s.	Dependente de aspetos de *hardware*, podendo ser tão baixo quanto 0,001 s.
2-D e 3-D	Pode executar tarefas 3-D e com múltiplos comprimentos de onda (dentro do espectro de luz visível) facilmente.	Executa tarefas 2-D com relativa facilidade, mas é lento e limitado em tarefas 3-D.
Perceção	Percebe variações de brilho em escala logarítmica. A interpretação subjetiva de brilho depende da área ao redor do objeto considerado.	Pode perceber brilho em escala linear ou logarítmica.

Vamos abordar agora a Estrutura de um Sistema de Visão Artificial. Definiremos um Sistema de Visão Artificial (SVA) como um sistema computadorizado capaz de adquirir, processar e interpretar imagens reais.

A figura 3 representa o diagrama de blocos de um SVA [2].

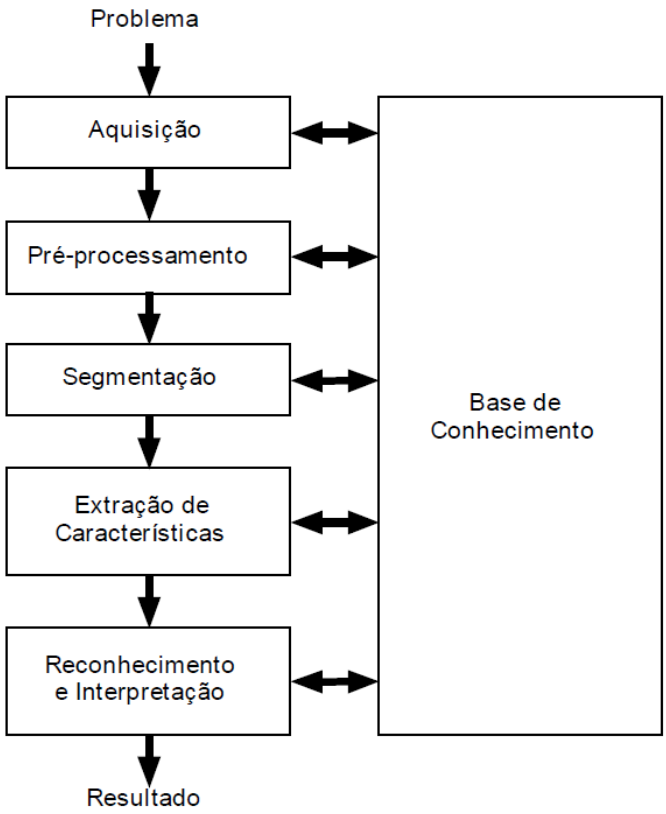

Figura 2 - Um Sistema de Visão Artificial (SVA) e suas principais etapas.

O domínio do problema, neste caso, consiste no reconhecimento de uma imagem com vários objetos e o objetivo do SVA é reconhecer cada um deles. Assim, o resultado esperado é uma imagem com os nomes / significado correspondentes a cada objeto da imagem.

O primeiro passo no processo consiste na aquisição de imagens dos objetos. Para isso, é necessário um dispositivo de captura de imagens como por exemplo uma câmara de *smartphone* para o caso em estudo. A câmara do *smartphone* transformará a imagem analógica em imagem digital. Nesta etapa, alguns dos itens importantes para o projeto são: o *smartphone* escolhido e o tipo de câmara incorporado, as condições de iluminação da cena, os requisitos de velocidade de aquisição, a resolução e o número de níveis de cinza da imagem digitalizada, entre outros. Esta etapa produz à saída uma imagem digitalizada dos objetos juntamente com o fundo.

O passo seguinte é denominado de pré-processamento. A imagem resultante do passo anterior pode apresentar imperfeições, tais como: presença de pixéis ruidosos, contraste e/ou brilho inadequado, etc. A função da etapa de pré-processamento é garantir a qualidade da imagem para as etapas seguintes. As operações feitas nesta etapa são ditas de baixo nível, porque lidam diretamente com os valores de intensidade dos pixels, sem nenhum conhecimento de quais pertencem aos

objetos e quais pertencem ao fundo da imagem. A imagem resultante desta etapa é uma imagem digitalizada de melhor qualidade que a original.

Posteriormente é segmentada a imagem. A etapa de segmentação consiste em dividir uma imagem em zonas significativas, ou seja, nos objetos de interesse que a compõem. Esta tarefa, apesar de simples de descrever, é das mais difíceis de implementar. No caso específico do problema apresentado, vamos dividi-lo em duas etapas: em um primeiro lugar, os algoritmos de segmentação tentarão localizar os objetos, do restante das informações, para depois trabalharmos sobre esta subimagem, segmentando cada objeto individualmente. Este bloco produzirá à saída subimagens, cada uma correspondendo a um objeto.

Avançamos depois para a etapa de extração de características. Esta etapa destina-se a extrair características das imagens resultantes da segmentação, através de descritores que permitam caracterizar com precisão cada objeto, e que apresentem bom poder de discriminação entre objetos parecidos. Estes descritores devem ser representados por uma estrutura de dados adequada ao algoritmo de reconhecimento. É importante observar que, nesta etapa, a entrada ainda é uma imagem, mas a saída é um conjunto de dados correspondentes a essa imagem. Para maior clareza, suponhamos que os descritores utilizados para descrever um objeto sejam as coordenadas normalizadas x e y do respetivo centro de gravidade e a razão entre a respetiva altura e largura. Neste caso, um vetor de três elementos é uma estrutura de dados adequada para armazenar estas informações sobre cada objeto processado por esta etapa. Designamos a estes dados de bilhete de identidade (BI) do objeto.

Nesta última etapa, chamada de reconhecimento e interpretação, o reconhecimento é o processo de atribuição de um BI a um objeto baseado nas respetivas características, traduzidas pelos respetivos descritores. A tarefa de interpretação, por outro lado, consiste em atribuir significado a um conjunto de objetos reconhecidos.

Para os passos anteriores a base de conhecimento é uma das componentes mais importantes. Todas as tarefas das etapas descritas acima pressupõem a existência de um conhecimento sobre o problema a ser resolvido, armazenado numa base de conhecimento, cujo tamanho e complexidade podem variar significativamente. Idealmente, esta base de conhecimento deveria guiar o funcionamento de cada etapa, e permitir a realimentação entre elas. Por exemplo, se a etapa de representação e descrição recebesse um objeto desconhecido, ela deveria ser capaz de identificar o problema na etapa de extração de características (responsável pela falha), para que esta procurasse o objeto não reconhecido e solicitasse informação do mesmo. Esta integração entre as várias etapas, através da base de conhecimento, ainda é um objetivo difícil de alcançar e não está presente na

maioria dos SVAs existentes atualmente. Ou seja, neste momento temos de percorrer todas as etapas para saber se o nosso objeto é reconhecido ou não.

Com o surgir da realidade aumentada, tornou-se uma prioridade o reconhecimento de imagens e padrões para que seja possível ter sobre a realidade uma visão computacional com a agregação de informação à vida real. Todos os algoritmos de realidade aumentada tem subjacente, uma forte componente de deteção e reconhecimento de imagens, sem a qual não seria possível apresentarem informação digital sobre a realidade.

Este projeto surge no âmbito da necessidade de uma API de reconhecimento de imagens para a empresa TIMWE, no âmbito do desenvolvimento de projetos de realidade aumentada, *marketing* entre outros, para possibilitar a entrada de vários produtos idealizados e com o propósito da inovação no mercado dos *smartphones*. Com o reconhecimento em tempo real de objetos podemos trabalhar em várias vertentes, como o *marketing* interativo, onde podemos ver com a câmara no nosso *smartphone* em realidade aumentada anúncios de publicidade entre outras coisas.

Neste projeto abordam-se várias técnicas de segmentação, deteção e reconhecimento de imagens para a construção de uma API que satisfaça as necessidades dos projetos referidos em cima. Para a concretização da API, utilizamos uma biblioteca de processamento de imagem OpenCV [3, 4] feita pela Intel, NDK e SDK do Android [5] para os acessos ao *hardware*, *software* e APIs do *smartphone*, Java JDK [6], Java JRE [6], JNI [7, 8], IDE eclipse com o ADT Plugin do Android [5].

Existem algumas soluções apresentadas por empresas, tais como por exemplo a Google com o Googles [9], Layar com Layar Vision [10], entre outras. Neste projeto, analisou-se estas soluções e analisaram-se as diferenças de comportamento relativamente à solução apresentada. No capítulo que se segue descreve-se com mais pormenor as soluções antes referidas.

Este relatório está dividido em 5 capítulos, introdução onde é descrito o propósito do projeto, um estado da arte onde se apresentam outros trabalhos nesta área, os métodos estudados para o propósito do projeto, a implementação e a criação da API e no capítulo são apresentados os resultados, verificação da performance da solução apresentada no reconhecimento dos objetos. Por último existe um capítulo onde são apresentadas algumas conclusões.

Capítulo 2

Estado da Arte

Existem várias aplicações para dispositivos móveis que permitem fazer o reconhecimento de imagens em tempo real, mas, como descrito anteriormente, apenas reconhecem etiquetas com uma estrutura específica com QR-Codes, não reconhecendo qualquer tipo de objeto.

Uma das que se assemelha ao projecto que apresentamos neste trabalho é o Google Googles [9] que reconhece fotografias tiradas com a câmara do nosso *smartphone* e faz também reconhecimento em tempo real através da câmara do mesmo, sendo este ultimo reconhecimento muito limitado, pois além de demorar muito tempo no reconhecimento, não reconhece muitos objetos porque para um bom reconhecimento precisaria de encontrar na imagem alguma palavra, ou uma estrutura, bem conhecida e posteriormente fazer uma pesquisa na base de dados do Google, para encontrar os resultados que nem sempre satisfazem a procura.

Para conhecer melhora o *software* Google Googles [9] respondemos as seguintes perguntas:

O que é o Google Googles?

- É uma aplicação para dispositivos Android e iPhone, que permite pesquisar no Google usando imagens em vez de palavras. Isso requer tirar uma foto ao objeto sobre o qual se deseja obter informações. Google verifica a imagem na sua base de dados e fornece resultados de pesquisa relacionados com a imagem fotografada.

Como funciona?

- Google Googles funciona através da decomposição de uma imagem em várias partes e, de seguida, compará-las com a sua base de dados. Assim, o Google retorna resultados relevantes para o conteúdo da imagem.

Informações em tempo real?

- Com o Google Googles podemos obter informações sobre um local específico. Não existe necessidade de obter qualquer fotografia, apenas focar o local de interesse com a câmara do *smartphone* e automaticamente ele exibe informações sobre esse local. Para isso, o serviço usa o GPS e a bússola do telefone assim como a sua localização e reconhecimento do local filmado pela câmara do *smartphone*.

A tecnologia?

- A tecnologia de reconhecimento visual que usa o Google Googles ainda está em desenvolvimento. Apesar da enorme base de dados que a Google já possui, a aplicação de

pesquisa por imagem ainda não é capaz de reconhecer com sucesso imagens de objetos como alimentos, carros, plantas ou animais.

De seguida apresento algumas aplicações que reconhecem imagens específicas.

Layar Vision [10]

- Layar Vision é uma API desenvolvida pela empresa Layar, para integração de uma aplicação. Layar Vision está disponível para várias plataformas.
  Esta plataforma disponibilizada a terceiros para integração de qualquer outra aplicação mediante um pagamento por fotografia armazenada na base de dados da própria Layar. Com o objetivo de reconhecer cartazes publicitários, *flyers*, *outdoors*, anúncios, etc.

Como funciona o Layar Vision?

- Uma empresa que compre os serviços Layar Vision deve enviar uma fotografia dos placares publicitários, *outdoors* entre outros, que façam parte do produto a apresentar. Por sua parte, a Layar processa as imagens e guarda o BI das mesmas na base de dados deles, associadas a informações, vídeos, imagens, etc., conforme o pedido da empresa que solicita o trabalho. Posteriormente, a Layar fornece a API para integração no software desejado pela empresa e quando os placares publicitários outdoors entre outros forem focados pela câmara do *smartphone* aparecerá a informação associada à imagem como realidade aumentada.

De seguida apresento algumas aplicações e APIs de realidade aumentada que permitem o reconhecimento de uma imagem específica (etiqueta ou *tag*).

Uma das mais conhecidas APIs de realidade aumentada é ARToolKit [11] da empresa ARToolWorks Inc [11]. Esta API está preparada para reconhecer uma etiqueta com a câmara de um *smartphone* e apresentar sobre ela o que nós quisermos (imagens, vídeos, informação em forma de texto, etc.). Ela reconhece a etiqueta como as aplicações que reconhecem *QR-Codes*. A etiqueta tem um padrão específico composta normalmente por uma imagem a preto e branco que contem apenas quadrados dentro de uma matriz, esses quadrados pertencem a um código interpretado pela aplicação.

Na figura que se segue podemos observar alguns exemplos de etiquetas reconhecidas pela API ARToolKit [11].

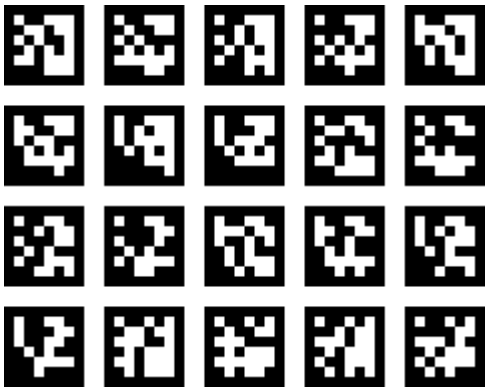

Figura 3 - Exemplo de etiquetas reconhecidas pela API ARToolKit [11].

Junaio [12] é outra API de realidade aumentada que tem uma plataforma de reconhecimento de imagens própria, mas o forte da API Junaio, é a realidade aumentada por localização, ou seja ele deteta a nossa localização e exibe informação dos locais reconhecidos assim como edifícios, etc., no reconhecimento de imagem é como a API anterior ele trabalha sobre imagens específicas como as etiquetas que falamos anteriormente, alguns padrões específicos ou sobre imagens previamente tratadas por eles e guardadas em base de dados próprias.

Finalmente vamos falar sobre a Marvel [13] que lançou recentemente uma aplicação para aumentar a realidade nos seus livros de aventuras. A aplicação reconhece imagens que não são mais do que etiquetas produzidas por eles com um padrão específico que sobrepõe a uma imagem de um herói de banda desenhada pode ser em 3D ou então um pequeno vídeo da história que estamos a ler.

Como podemos constatar, quase todas as aplicações e APIs existentes tem um padrão muito específico de reconhecimento de imagens, a única aqui apresentada que é mais geral é a Google Googles, só que ainda não está preparada para o reconhecimento de objetos simples como por exemplo utensílios de cozinha, ferramentas, partes do corpo humano, etc.

A solução apresentada neste projeto pretende reconhecer objetos de qualquer tipo mediante o treino do objeto em causa, para isso vamos utilizar como já referido algoritmos de deteção, segmentação e reconhecimento de imagens e também utilizar algoritmos de inteligência artificial para trabalhar o treino dos objetos.

No tratamento de imagens é utilizada a API OpenCV [3, 4] desenvolvida pela Intel.

OpenCV é uma biblioteca de programação, de código aberto, desenvolvida inicialmente pela Intel Corporation [3, 4]. O OpenCV implementa uma variedade de ferramentas de interpretação de imagens, indo desde operações simples como um filtro de ruído, até operações complexas, tais como

a análise de movimentos, reconhecimento de padrões e reconstrução em 3D. O pacote OpenCV está disponível gratuitamente na Internet bem como o manual de referência.

A biblioteca OpenCV possui mais de 500 funções. Foi idealizada com o objetivo de tornar a visão computacional acessível a utilizadores e programadores em áreas tais como a interação humano-computador em tempo real e a robótica. A biblioteca está disponível com o código fonte e os executáveis (binários) otimizados para os processadores Intel. Um programa OpenCV, ao ser executado, invoca automaticamente uma DLL (*Dynamic Linked Library*) que deteta o tipo de processador e carrega, por sua vez, a DLL otimizada para este. Juntamente com o pacote OpenCV é oferecida a biblioteca IPL (*Image Processing Library*), da qual a OpenCV depende parcialmente, além de documentação e um conjunto de códigos exemplos. A biblioteca está dividida em cinco grupos de funções: Processamento de imagens; Análise estrutural; Análise de movimento e rastreamento de objetos; Reconhecimento de padrões e Calibração de câmara e reconstrução 3D. As principais funções são apresentadas a seguir, juntamente com os conceitos de processamento de imagens e visão computacional [3, 14].

A tabela seguinte é um comparativo entre as várias APIs e a solução proposta.

Tabela 2 - Comparação entre a solução proposta neste projeto e outras.

	Solução Apresentada	Layar Vision	ARToolKit	Junaio
API de reconhecimento de imagem	OpenCV	OpenCV	OpenCV	OpenCV
Reconhecimento de imagem	Qualquer imagem que exista na base de dados de conhecimento, com aprendizagem	Imagens que existam na base de dados proprietárias	Etiquetas	Etiquetas e imagens que existam na base de dados proprietárias
Base de dados de conhecimento	Atualizada consoante as imagens treinadas	Atualizada mediante aprovação da empresa	Atualizada mediante pedido de etiquetas	Atualizada mediante aprovação da empresa
Realidade Aumentada	Sim	Sim	Sim	Sim
Realidade Aumentada por Localização	Não	Não	Não	Sim
Reconhecimento de vários objetos	Sim	Não	Não	Não
Custo	-	Custo por imagem armazenada	Grátis para a versão lite	Custo por imagem armazenada

Como foi descrito atrás, existem soluções no mercado que se assemelham ao objetivo proposto neste projeto mas nenhuma das soluções vai de encontro ao objetivo proposto que é o multi-reconhecimento em tempo real.

Neste capítulo apresentei o que existe no mercado para o reconhecimento de imagens e as empresas que estão a trabalhar em projetos de reconhecimento em tempo real.

No capítulo seguinte apresento os métodos estudados e implementados para que o objetivo proposto seja atingido, vamos abordar alguns métodos que se utilizam para o reconhecimento de imagens.

Capítulo 3

Métodos Estudados

Neste capítulo apresenta-se as técnicas e métodos usados para o desenvolvimento do sistema de segmentação e deteção de objetos. Inicialmente apresentam-se algumas técnicas de base que foram necessárias quer para o pré-processamento quer para a tomada de decisões em diferentes etapas do sistema. De seguida, apresentam-se os métodos de segmentação, métodos de deteção de contornos e métodos de treino e reconhecimento de imagem estudados e desenvolvidos.

3.1 Métodos Base e de Pré-Processamento

3.1.1 Conversão da Imagem para Tons de Cinza

A conversão da imagem RGB (*Red, Green, Blue*) para tons de cinza foi feita através da conhecida fórmula, comumente usada em aplicações científicas, que toma em consideração a capacidade de absorção dos diferentes comprimentos de onda pelo olho humano. É uma média ponderada das componentes de cores primárias, como mostra a seguinte formula [15, 16]:

$$GRAY = RED \times 0.3 + GREEN \times 0.59 + BLUE \times 0.11$$

Esta conversão é necessária para podermos implementar os algoritmos descritos em seguida, é normalmente muito utilizada em aplicações que trabalham com imagens.

A figura que se segue mostra a resultado da conversão de uma imagem (RGB) para tons de cinza.

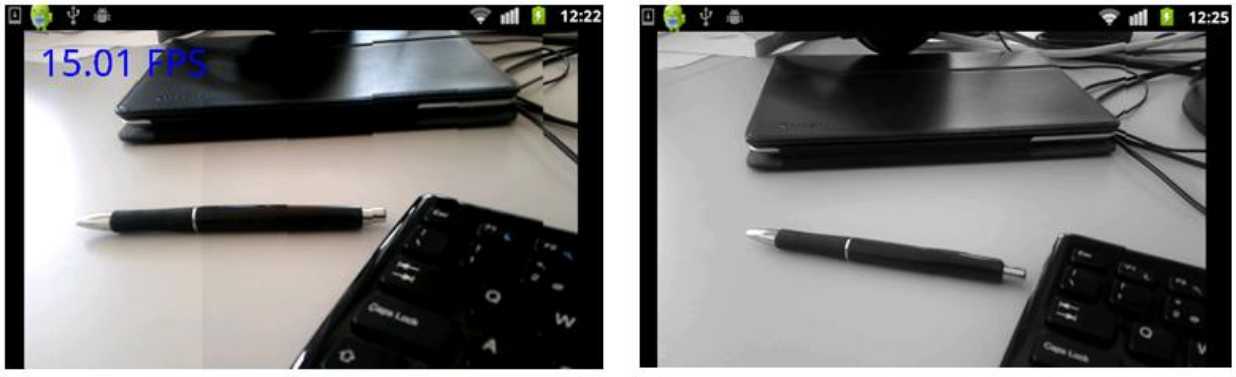

Figura 4 - Exemplo de conversão da Imagem para Tons de Cinza

3.1.2 Histograma da Imagem em Tons de Cinza

O histograma de uma imagem em tons de cinza é uma função h(L) que produz o número de ocorrências de cada nível de cinza, L, para 0 <= L >= max na imagem. No caso das imagens utilizadas o máximo é 255 (cor branca). O histograma representa, quando normalizado, a distribuição de probabilidade dos valores dos pixéis.

O histograma é importante para sabermos a concentração dos níveis de cinza na imagem e assim podermos aplicar os algoritmos de segmentação. Na imagem que se segue apresenta-se o exemplo de um histograma.

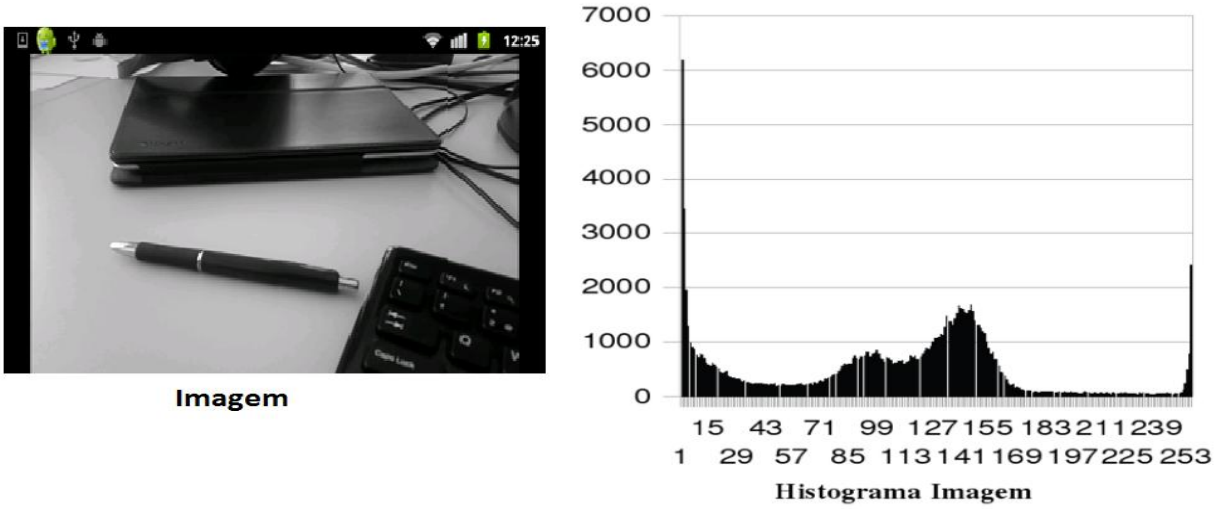

Figura 5 - Imagem com o respetivo Histograma

3.1.3 Operador Gradiente

Os operadores de gradiente são algoritmos simples e rápidos. O Cálculo do gradiente duma imagem é feito com a derivada para calcular as diferenças de valores de brilho na direção vertical e horizontal, esse cálculo e feito da seguinte forma [15, 16]:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \\ \frac{\delta f}{\delta y} \end{bmatrix}$$

O operador gradiente é simples mas computacionalmente difícil de implementar, por essa razão normalmente utiliza-se a aproximação da magnitude gradiente para valores absolutos.

$$\nabla f \quad = \quad |G_x| + |G_y|$$

Estimar o gradiente com recurso às aproximações numéricas unidimensionais, tem a grande desvantagem de se estar a calcular a derivada horizontal e a vertical em pontos diferentes.

$$G_x = [-1\ 1] \qquad G_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Uma alternativa é a utilização de janelas quadradas:

$$G_x = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

Após calculados os gradientes para os eixos, temos que fazer o cálculo para o gradiente geral que é feito com a raiz quadrada da soma dos quadrados dos gradientes anteriores, i.e., sendo G o gradiente,

$$G = \sqrt{(G_x)^2 + (G_y)^2}$$

Este operador é utilizado na grande parte das vezes nos métodos de deteção de contornos como os que estudamos para este projeto, como por exemplo no operador de Sobel [15, 16], Roberts [15, 16], Prewitt [15, 16] e de Laplacian [15, 16], neste ultimo não são usadas as janelas 2x2 mas sim janelas 3x3 porque o cálculo é feito pelo ponto central e pela segunda derivada e não pela primeira como nos anteriores.

3.2 Segmentação

Ao analisar uma imagem, necessitamos frequentemente de destacar uma determinada região, permitindo uma visualização mais precisa dos objetos que se procura analisar na imagem. Utilizam-se frequentemente para esta tarefa técnicas de segmentação de imagens.

Segmentar uma imagem consiste em dividi-la em diferentes regiões, de forma que os pixéis de cada uma delas possuam características específicas, por exemplo, nível de cinza com valor semelhante. A partir da imagem segmentada, é possível executar as medições das regiões ou mesmo estabelecer relações de vizinhança entre regiões adjacentes.

A segmentação de imagens são as técnicas mais utilizadas e muito úteis para a separação de objetos de uma imagem. Descrevemos de seguida algumas técnicas adequadas para o tipo de imagens que pretendemos segmentar. Os algoritmos gerais de segmentação são baseados em uma ou duas propriedades de valores de intensidade: descontinuidade e similaridade.

Na primeira parte do projeto foram utilizadas as técnicas de segmentação baseadas em regiões, que consideram pixéis com intensidade similar como pertencendo à mesma região da imagem. Na segunda parte foram utilizadas técnicas de segmentação baseadas na deteção de descontinuidades, para deteção dos contornos dos objetos da imagem.

A deteção de contornos é um procedimento para delinear os objetos, permitindo a separação dos mesmos. Na deteção de contornos, analisamos as descontinuidades nos níveis de cinza, sendo um contorno o limite entre duas regiões com propriedades relativamente distintas de nível de cinza. As linhas na imagem que caracterizam os contornos dos objetos são bastante úteis para segmentação e identificação de objetos na imagem.

Pontos de limite podem ser entendidos como as posições dos pixéis com variações abruptas de níveis de cinza. Os pontos da linha caracterizam as transições entre objetos diferentes.
Neste projeto foram estudados e implementados vários métodos de deteção de contorno, em que as linhas são detetadas pelos operadores gradiente, seguidos de uma linearização.

Todos os métodos estudados encontram-se na API OpenCV [3, 4] no fim foi escolhido o que se adequava mais à nossa solução.
De seguida vamos pormenorizar todos os métodos estudados.

3.2.1 Segmentação por K-means

O K-Means [15, 16] é um método de segmentação baseado em regiões que permite minimizar o número de tons de cinza de uma imagem de forma a se conseguir agrupar os objetos com mais semelhanças da imagem, conseguindo assim separar esses objetos.

O método depende de um fator K, o qual é número de regiões em que queremos que a imagem seja dividida, fazendo com que os pixéis mais próximos de uma determinada região sejam coloridos com a cor dessa região.

O método segmentação de K-Means efetua os seguintes passos [15, 16]:

1. Determinação do K a utilizar;
2. Cálculo do histograma e da respetiva quantidade de pixéis com uma determinada cor;
3. Divisão do histograma em K intervalos e cálculo do *centróide* das cores para cada intervalo;
4. Cálculo da distância da cor de cada pixel a todas as K cores. Neste caso foi utilizada a distância euclidiana [6]

$$D = \sqrt{(K - I)^2}$$

   Onde K corresponde à intensidade de cada cor calculada e *I* corresponde à intensidade da cor do pixel da imagem;
5. Calculo da menor distância.

Na imagem seguinte podemos ver a aplicação do algoritmo de K-Means com k=4.

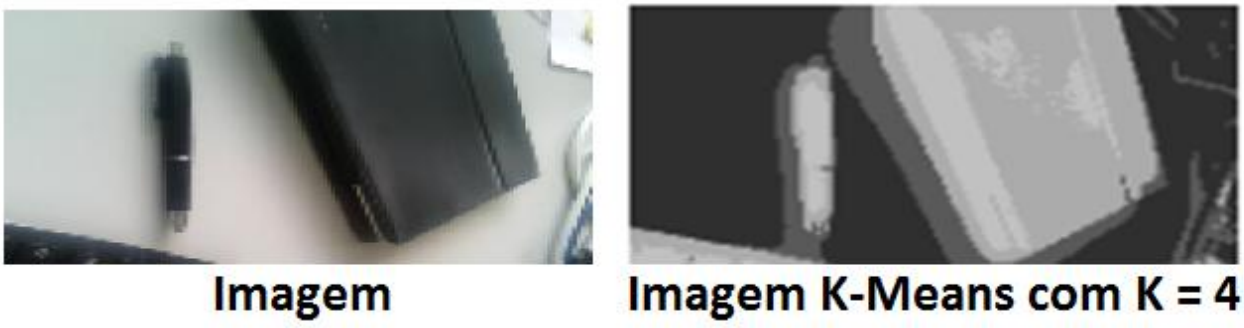

Figura 6 - Imagem original (esquerda) e após aplicação do algoritmo K-means (direita).

3.2.2 Segmentação por Watershed

O método de segmentação *Watershed* [15, 16] é um método de segmentação muito utilizado para a separação de objetos de uma imagem. Este método baseia-se no princípio de "inundação de relevos topográficos".

Essa abordagem vem do princípio de que uma imagem em níveis de cinza pode ser vista como um relevo topográfico, formando vales, que representam as regiões mais escuras da imagem, e formando montanhas, que representam as partes mais claras. É possível visualizar a "inundação" de duas maneiras distintas: a água vinda de cima, como se fosse chuva; ou vinda de baixo, como se o relevo estivesse perfurado nos pontos de altitude mínima e fosse imerso. Conforme as bacias vão sendo inundadas, águas provenientes de diferentes bacias se encontram, formando, nos pontos de encontro, represas ou linhas divisoras de águas, as chamadas "*watersheds*" [15, 16].

O método segue os seguintes passos [15, 16]:

1. Cálculo do operador gradiente da imagem;
2. Marcação das fronteiras primeira e última linha, primeira e última coluna, marcadas como mínimos;
3. Cálculo dos operadores gradientes mínimos para cada pixel e marcações das fronteiras, aumentando as partes mais claras "montanhas" e diminuindo as partes mais escuras "vales";
4. Se todos os objetos foram marcados termina, senão volta ao ponto 3.

A imagem seguinte mostra um exemplo da segmentação Watershed. Podemos observar também que este tipo de segmentação só pode ser aplicado após a aplicação do método de K-means [15, 16], pois sem ter a separação das cores e grupos mais pequenos de tons de cinza o algoritmo Watershed produz uma imagem com muito ruido e difícil perceção, neste caso o algoritmo de K-means serve de filtro para a imagem.

Figura 7 - Segmentação por Watershed.

3.2.3 Segmentação por Region Growing

A segmentação por Region Growing [15, 16], ou crescimento de regiões, é o método que agrupa pixéis ou sub-regiões em regiões maiores. A agregação de pixéis inicia-se com um conjunto de pontos "sementes", a partir dos quais crescem as regiões, aumentando a cada ponto "semente" aqueles pixéis que possuem propriedades similares (como níveis de cinza, textura ou cor).

Este método é efetua-se seguindo os seguintes passos [15, 16]:

1. Seleção de um pixel do objeto da imagem a estudar e marcação desse pixel com uma determinada cor;
2. Percorre-se a imagem do início para o fim fazendo crescer os pixéis a volta do pixel marcado anteriormente até encontrar uma linha de separação;
3. Repetição do passo 2 percorrendo a imagem do fim para o início para que os pixéis anteriores ao pixel marcado também possam crescer.
4. Procuram-se todos os vizinhos de pixéis coloridos até se encontrar uma linha, para que o objeto fique todo colorido.

Na figura que se segue temos a aplicação do algoritmo de Region Growing. Antes da implementação deste algoritmo tem que se implementar algoritmos de deteção de contornos, neste caso foi aplicado o método Watershed descrito em cima.

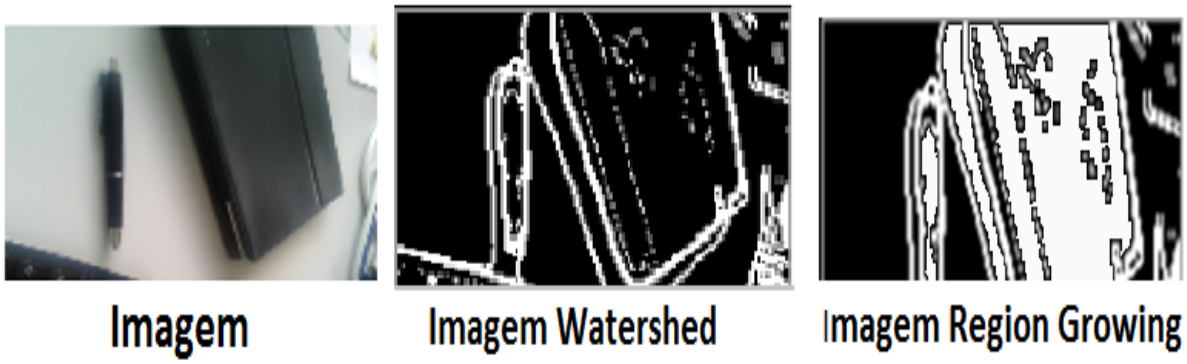

Figura 8 - Segmentação por Region Growing.

Vamos agora apresentar os algoritmos de segmentação para detetar contornos. Estes algoritmos são bastante utilizados no tratamento de imagens são umas das partes mais importantes na segmentação de imagens.

3.2.4 Deteção de Contornos por Operador de Roberts

É o mais antigo e simples algoritmo de deteção de contornos. Utiliza uma matriz 2x2 para encontrar as mudanças nas direções em x e em y.

O operador de Roberts [15, 16] faz o gradiente cruzado, isto é, em vez de calcular as diferenças de valores de brilho na direção vertical e horizontal, fá-lo numa direção rodada de 45º.
As janelas de convolução são as seguintes [15, 16]:

$$G_x = \begin{bmatrix} -1 & 0 \\ 0 & +1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Na figura seguinte podemos ver a implementação da segmentação pelo operador de Roberts. Nela podemos observar que este tipo de segmentação só resulta após a conversão para tons de cinza da imagem.

Figura 9 - Deteção de contornos pelo Operador de Roberts.

3.2.5 Deteção de Contornos Pelo Operador de Prewitt

O operador de Prewitt [15, 16], além de diferenciar, suaviza a imagem, atenuando os efeitos negativos do ruído e não tem o enviesamento do gradiente digital.
As janelas de convolução do Prewitt são:

$$G_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Na figura seguinte podemos ver a implementação da segmentação pelo operador de Prewitt, após a conversão para tons de cinza da imagem.

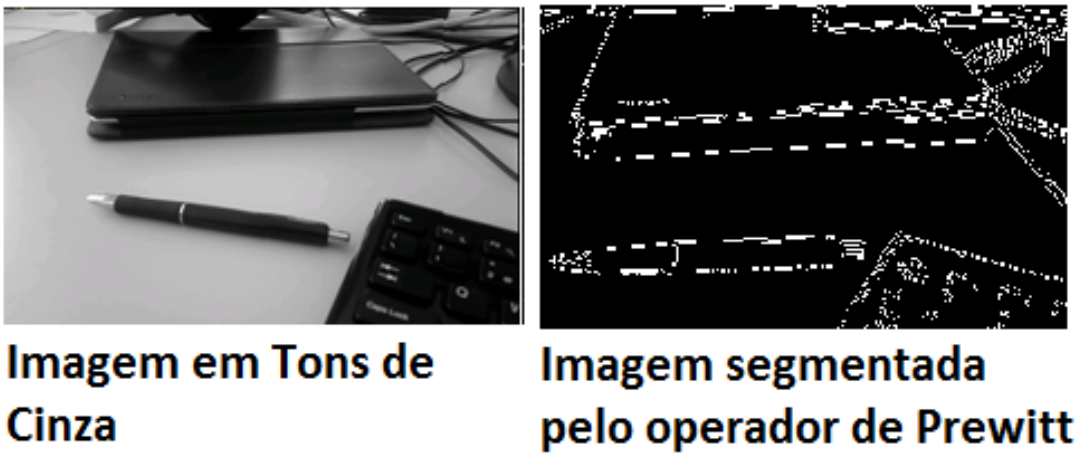

Figura 10 - Deteção de contorno pelo operador Prewitt.

3.2.6 Deteção de Contornos Pelo Operador de Sobel

O operador de Sobel [15, 16] é muito semelhante ao de Prewitt só que dá mais peso aos pontos mais próximos do pixel central. Relativamente ao operador Prewitt tem a vantagem de ter respostas, às linhas diagonais, menos atenuadas devido precisamente a dar mais peso aos pontos centrais.

As janelas de convolução de Sobel [15, 16] são:

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Na figura que se segue podemos ver a implementação da segmentação pelo operador de Sobel, após a conversão para tons de cinza da imagem.

Figura 11 - Deteção de contornos pelo operador Sobel.

3.2.7 Deteção de Contornos Pelo Operador de Laplace

O operador de Laplace difere dos anteriores porque utiliza o método da segunda derivada. Se as primeiras derivadas apresentam linha simples as segundas derivadas apresentam linhas duplas ou seja os objetos contidos nas imagens ficam mais realçados.

O operador de Laplace representa um método da segunda derivada, e é definido como [15, 16]:

$$\nabla^2 f = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2}$$

O operador de Laplace pode ser implementado de várias maneiras, sendo que, para uma região 3 X 3 pixéis, a forma mais frequentemente utilizada é:

$$\nabla^2 f = 4z_5 - (z_2 + z_4 + z_6 + z_8)$$

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Onde os z's são os níveis de cinza dos pixéis sobrepostos pela máscara numa parte da imagem.

Embora o operador de Laplace responda em intensidade, na prática ele é raramente utilizado para deteção de contorno por vezes é combinado com o Gaussiano, OperadorLoG. Como uma derivada de segunda ordem, Laplace é sensível ao ruído da imagem o que pode por vezes dar informações erradas da mesma. Além disso, produz bordas duplas e é incapaz de detetar a direção da linha. Por estas razões, normalmente faz o papel secundário de detetor para o estabelecimento se o pixel é do lado escuro ou claro da linha.

Na figura que se segue podemos ver a implementação da segmentação pelo operador de Laplace, após a conversão para tons de cinza da imagem.

Figura 12 - Deteção de contorno pelo operador Laplace.

3.2.7 Deteção de Contornos Canny

A deteção de contornos com o algoritmo de Canny [15, 16] é o mais utilizado para este fim, é utilizado na API OpenCV [3, 4] e neste projeto. É o mais utilizado porque utiliza um algoritmo multi-estágio para detetar uma vasta gama de arestas em imagens.

Etapas do algoritmo de Canny [15, 16]:

Uniformização da imagem

Porque o detetor de contornos Canny [15, 16] é suscetível ao ruído utiliza-se um filtro de suavização descrito pela seguinte fórmula.

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

Diferenciação

Um contorno de uma imagem pode estar numa variedade de direções, o algoritmo de Canny [15, 16] utiliza quatro filtros para detetar contornos horizontais, verticais e diagonais na imagem. O operador deteção de contornos (Roberts [15, 16], Prewitt [15, 16], Sobel [15, 16], por exemplo) retorna um valor para a primeira derivada na direção horizontal (Gx) e na direção vertical (Gy) como descrevemos em cima. A partir deste o gradiente de contorno e direção pode ser determinado [15, 16]:

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$
$$\mathbf{\Theta} = \arctan\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right).$$

Pelo ângulo de direção da margem arredondado para um de quatro ângulos representados vertical, horizontal e as duas diagonais (0, 45, 90 e 135 graus, por exemplo).

Omissão de pontos de mínima intensidade.

Encontradas as estimativas dos gradientes de imagem, realiza-se uma pesquisa para determinar se a magnitude gradiente assume um máximo local. Assim, por exemplo, se o ângulo de inclinação é arredondado para zero graus (ou seja, o contorno está na direção norte-sul), o ponto será considerado contorno, se a sua magnitude é maior do que as magnitudes no oeste e leste, se o ângulo do gradiente arredondado é de 90 graus (isto é, o contorno está na direção este-oeste), o ponto será considerado na contorno, se a sua magnitude é maior do que as magnitudes no norte e sul, se o ângulo de inclinação é arredondado para 135 graus (ou seja, o contorno está na direção noroeste a sudeste), o ponto será considerado contorno, se a sua magnitude do gradiente é maior do que as magnitudes no noroeste e sudeste, se o ângulo de inclinação arredondado é de 45 graus (ou

seja, o contorno está na direção sudoeste nordeste), o ponto será considerado contorno, se a sua magnitude do gradiente é maior do que as magnitudes no nordeste e sudoeste.

A partir desta fase, é obtido um conjunto de pontos, sob a forma de uma imagem binária.

***Limiarização* de contornos (*threshold*).**

A *limiarização* usado no algoritmo Canny usa o método chamado "*histerese*" [15, 16].

Considerando um segmento de contorno, para todo valor situado acima do limite superior de *limiarização*, ele é imediatamente aceite. Para todo valor situado abaixo do limite inferior de *limiarização*, ele é imediatamente rejeitado. Pontos situados entre os dois limites serão aceites se eles estiverem relacionados com pixéis que apresentem respostas fortes.

A figura abaixo mostra o resultado da implementação do operador Canny, após a conversão para tons de cinza da imagem.

Figura 13 - Deteção de contorno pelo operador Canny.

Algoritmo de deteção de contornos de Canny é muito usado para a deteção de contornos no reconhecimento de imagens. Este exemplo mostra como implementar a deteção de contornos em OpenCV [3, 4].

```
#include "stdafx.h"
#include "cv.h"
#include "highgui.h"

int main(void)
{
 IplImage* src=cvLoadImage("d:/cannypic.png");// load the image
 IplImage* graysrc;// pointer to store the gray scale image
```

```
IplImage* cannyedge;// pointer to store the output image
cvNamedWindow("Source Image", CV_WINDOW_AUTOSIZE );
cvNamedWindow( "Canny Edge", CV_WINDOW_AUTOSIZE );
      cannyedge=cvCreateImage( cvGetSize(src),8,1);
graysrc=cvCreateImage( cvGetSize(src),8,1);
cvCvtColor(src, graysrc,CV_RGB2GRAY );// convert RGB image to Gray scale image
cvCanny( graysrc, cannyedge, 200, 200, 3);// implement canny function
cvShowImage("Source Image",src);
cvShowImage( "Canny Edge",cannyedge );
   cvWaitKey(0);
   cvReleaseImage( &cannyedge );// release image from memory
   cvDestroyWindow( "Canny Edge" );// destroy the window
   cvReleaseImage( &src );
   cvDestroyWindow("Source Image");
 return 0;
}
```

IplImage é um ponteiro para a imagem. Função *cvLoadImage ()* carrega a imagem.

```
IplImage* src=cvLoadImage("d:/cannypic.png");// load the image
IplImage* graysrc;// pointer to store the gray scale image
IplImage* cannyedge;// pointer to store the output image
```

Função *cvCreateImage ()* cria cabeçalho imagem e aloca os dados de imagem.

```
cannyedge=cvCreateImage( cvGetSize(src),8,1);
graysrc=cvCreateImage( cvGetSize(src),8,1);
```

Função *cvCvtColor ()* converte a imagem RGB (*"src"*) para escala de cinza (*"graysrc"*). A função *cvCanny ()* implementa o algoritmo Canny em OpenCV. Executa o algoritmo em *"graysrc"* e armazena os dados da imagem resultantes em *"cannyedge"*.

```
cvCvtColor(src, graysrc,CV_RGB2GRAY );// convert RGB image to Gray scale image
cvCanny( graysrc, cannyedge, 200, 200, 3);// implement canny function
```

3.2.7 Deteção de características

Na deteção de características usamos um método presente na API OpenCV [3, 4], que nos devolve o número de características encontradas num determinado objeto, "*FastFeatureDetector()*", este método depois de feito todo o tratamento e segmentação descrito em cima, calcula todas as características do objeto que posteriormente serve para comparação e

respetivo conhecimento, estas características normalmente são detetadas pelas diferenças abruptas nas cores das imagens.

Na implementação do método *"FastFeatureDetector()"* existem dois objetos do tipo MAT *"PMarRgb"* e *"pMatGr"* que representam as imagens RBG e tons de cinza respetivamente, um vetor de pontos "v", um objeto do tipo "*FastFeatureDetector*" com um parâmetro que indica o número máximo de características a encontrar, aplicamos a imagem *"pMatGr"* o método *"detect"*, que devolve para o vetor "v" o número de característica encontradas, em seguida percorremos o vetor e desenhamos na imagem *"pMatRgb"* os pontos encontrados.

```
Mat* pMatGr=(Mat*)addrGray;
Mat* pMatRgb=(Mat*)addrRgba;
vector<KeyPoint> v;

FastFeatureDetector detector(50);
detector.detect(*pMatGr, v);

for( size_t i = 0; i < v.size(); i++ )
      circle(*pMatRgb, Point(v[i].pt.x, v[i].pt.y), 10, Scalar(255,0,0,255));
```

Na figura seguinte temos um exemplo da deteção de características.

Figura 14 - Deteção de características

3.3 Métodos de Treino e Reconhecimento de Imagens

Existem alguns métodos de treino e reconhecimento de imagens, estes métodos servem para ajudar um sistema de visão artificial (SVA) a reconhecer e distinguir imagens.
O método que vai ser apresentado é o mais utilizado no reconhecimento de imagens, tem uma componente de inteligência artificial e utiliza algoritmos como *AdaBoost (AB)* [17, 18, 19], e *Viola-Jones object detection framework* [20].

OpenCV já oferece alguns métodos para a implementação destes algoritmos que abordaremos, ver mais à frente.

O resultado desta implementação é um ficheiro XML, chamado de "*haar cascade classifier*" [21, 22, 23], que possui a informação do objeto treinado.
Para a construção deste ficheiro de treino são precisas várias imagens do objeto e várias imagens do não-objeto, para ele poder distinguir o objeto do não-objeto mais à frente apresentamos as técnicas para a criação de um ficheiro de conhecimento "*haar cascade classifier*" [21, 22, 23].

3.3.1 Algoritmo AdaBoost

AdaBoost [17, 18, 19], abreviatura de *Adaptive Boosting*, é um algoritmo de aprendizagem, criado por Yoav Freund e Schapire Robert [17, 18, 19] usado em classificadores em cascata, normalmente tem duas fases: treino e teste. É um meta-algoritmo, e pode ser usado em conjunto com muitos outros algoritmos de aprendizagem para melhorar seu desempenho. *AdaBoost* é adaptativo no sentido de que classificadores subsequentemente construídos são alterados em favor daqueles casos erroneamente classificados pelos classificadores anteriores. *AdaBoost* é sensível a dados ruidosos. Em alguns problemas, no entanto, pode ser menos suscetível à "*overfitting*", problema de maioria dos algoritmos de aprendizagem. Os classificadores que ele usa podem ser fracos (ou seja, apresentar uma taxa de erro substancial), mas desde que seu desempenho não seja aleatório (resultando com uma taxa de erro de 0,5 para classificação binária), eles vão melhorar o modelo final. Mesmo classificadores com uma taxa de erro mais elevada do que seria esperado a partir de um classificador aleatório será útil, uma vez que terá coeficientes negativos na combinação final linear de classificadores e, portanto, se comportam como suas inversas.

AdaBoost cria e chama um novo classificador fraco (apresenta uma taxa de erro substancial) em cada um de uma série de círculos. Para cada chamada, uma distribuição de pesos é atualizado e indica a importância de exemplos no conjunto de dados para a classificação. Em cada chamada, os pesos de cada exemplo classificado incorretamente são aumentados, e os pesos de cada exemplo corretamente classificados são reduzidos, de modo que o novo classificador concentra-se nos exemplos que até agora escapavam à classificação correta [17, 18, 19].

Como podemos observar na figura seguinte, cada estado é composto por dois nós de decisão, o primeiro é chamado nó de raiz ou de nó de entrada da árvore. Quando uma árvore tem apenas um nó de decisão (ou seja, uma característica única a ser testada), é chamada de *"Stump"*. Cada nó de

decisão corresponde à utilização de uma característica de "*Haar*" com avaliação limiar correspondente. Dependendo do resultado da comparação entre a avaliação limiar e a resposta dada pela característica, é possível continuar a ou não para o nível seguinte da árvore, em qualquer dos casos o ultimo nó a ser testado retorna um valor numérico que representa a resposta do classificador fraco aplicada à região em questão [17, 18, 19].

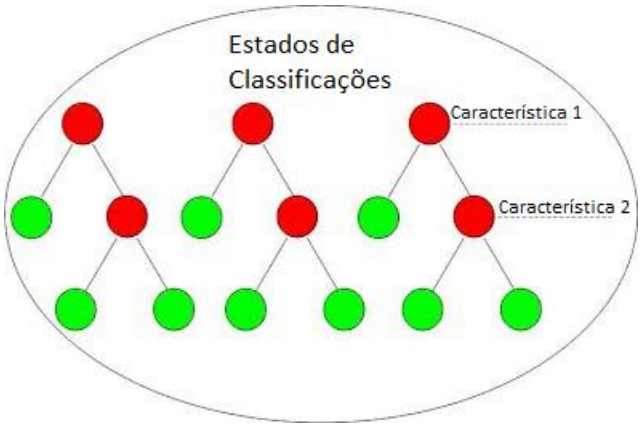

Figura 15 - Representação do algoritmo AdaBoost [17, 18, 19].

O algoritmo *Adaboost* utiliza como entrada um conjunto de treino (x1, y1),...,(xm, ym) onde cada xi é um vetor de características, pertencente a algum espaço X e yi pertencem ao conjunto Y = {-1, +1}, ou seja, o algoritmo é utilizado para problemas de classificação com duas classes. O *Adaboost* chama um classificador base, repetidamente, num conjunto de t execuções, t = 1,..., T. Uma das principais ideias do algoritmo é modificar a distribuição, ou o conjunto de pesos, sobre o conjunto de treino. O peso desta distribuição no exemplo de treino i na execução t é denominado por Dt(i). Inicialmente os pesos são todos iguais, para cada execução, os pesos dos classificadores incorretos são incrementados deforma que o classificador "base" seja forçado a atuar com maior intensidade sobre estes exemplos, no conjunto de treino, a taxa de erro é calculada contando o número de classificações incorretas [17, 18, 19].

AdaBoost é um algoritmo para a construção de um classificador "forte" com combinação linear de classificadores "simples", "fracos" hₜ(x) [17, 18, 19].

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Terminologia

- hₜ(x) ... "fraco" ou base de classificação, hipótese, "recurso"
- H(x) = sign(f(x)) ... "forte "ou classificador final / hipótese

Onde

- O hɬ(x) pode ser pensado como características da imagem
- O conjunto H = {h(x)} é infinito.

Representação do algoritmo [17, 18, 19]:

Dado: (x1, y1), ..., (xm, ym); xi Є X, yi Є Y={−1, 1}

Inicializar pesos Dt(i) = 1/m

Para t = 1, ..., T:

1. Retorna o classificador fraco ht: X → {−1, 1} com erro mínimo de distribuição Dt;
2. Escolhe αt Є R,
3. Atualiza

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

onde Zt é um fator de normalização escolhido de modo que Dt+1 seja uma distribuição

Saída do classificador forte:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

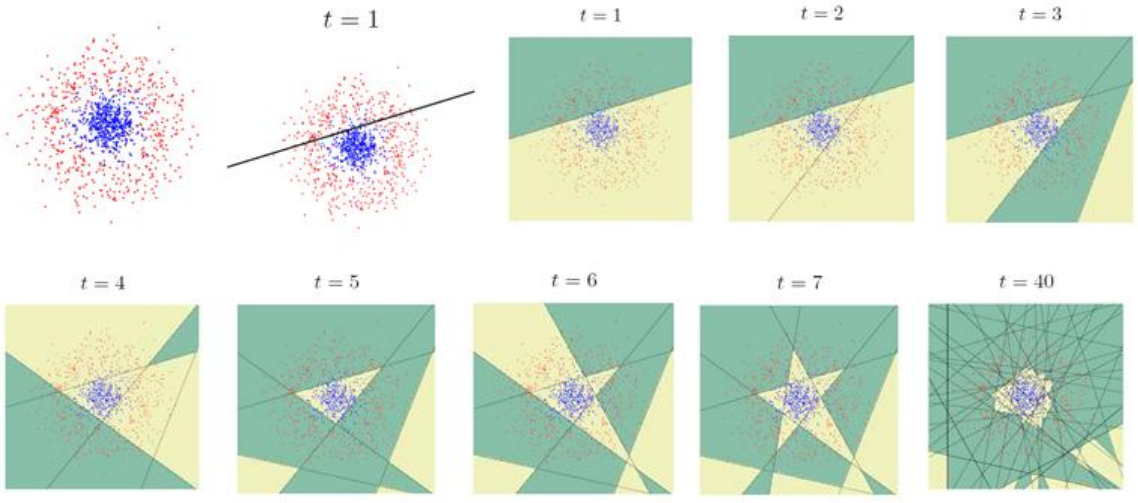

Figura 16 - Resultado do algoritmo AdaBoost[Jiri Matas and JanSochman AdaBoost 2009][17]

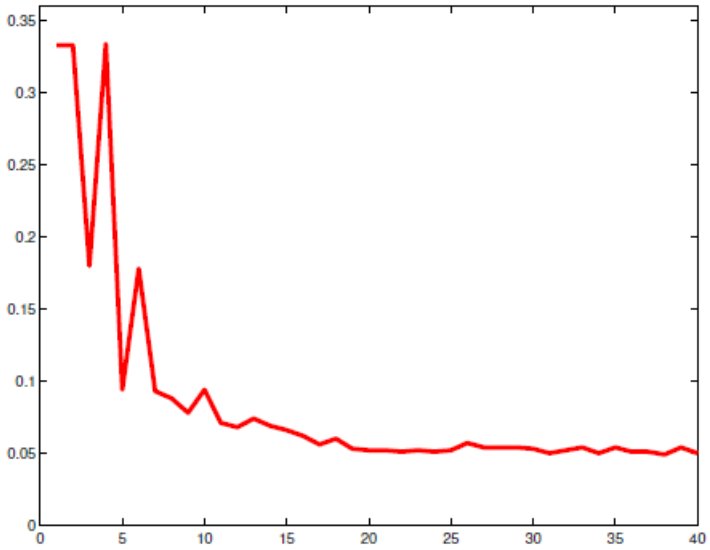

Figura 17 - Gráfico temporal do algoritmo AdaBoost [Jiri Matas and JanSochman AdaBoost 2009][17]

Como podemos observar nos resultados do algoritmo AdaBoost representados pelas figuras anteriores, retiradas do artigo de Jiri Matas and JanSochman AdaBoost 2009 [17], observamos que o algoritmo vai dividindo o conjunto de características, até ter um conjunto homogéneo das características "fortes" e "fracas". Podemos também observar que à medida em que o t aumenta, as divisões tendem para 0 o que quer dizer a separação de características ou pontos "fortes" e "fracos" ficam mais homogenias com o aumento do t.

Concluí-se também que ao aumentar o conjunto de treino temos mais probabilidades de reconhecimento do objeto pretendido.

3.3.2 *Framework* Viola-Jones

Viola-Jones [20] foi a primeira *framework* para detetar objetos em tempo real proposta em 2001 por Paul Viola e Jones Michael. Embora possa ser treinado para detetar uma variedade de classes de objetos, inicialmente foi projetado para a deteção de faces. Esta *framework* utiliza o algoritmo de *Adaboost* para fazer o treino e a classificação das imagens e esta presente na API OpenCV [3, 4] pela função *cvHaarDetectObjects()*.

Tipos de recursos e avaliação

São utilizados recursos de deteção que envolvem as somas de pixéis da imagem dentro de áreas retangulares, muito semelhante com funções de base *Haar* [21, 22, 23], que foram usadas anteriormente para a deteções de objetos, no entanto as características utilizadas por Viola e Jones não depende apenas de áreas retangulares, sendo geralmente mais complexas.

A figura que se segue ilustra os quatro tipos diferentes de recursos usados na imagem. O valor de qualquer característica é simplesmente a soma dos pixéis dentro retângulos claros

subtraídos da soma dos pixéis dentro retângulos sombreados. Apesar de serem sensíveis às características verticais e horizontais, o que retornam não é geralmente rigoroso. Mas com a utilização de uma representação da imagem denominada imagem integrante, as características retangulares podem ser avaliadas em tempo real, o que lhes confere uma vantagem de velocidade considerável sobre os seus opositores mais sofisticados, o que no nosso caso a velocidade é fundamental pois a nossa solução vai correr em *smartphones* onde o *hardware* é limitado. Para a extração de características a *framework* Viola–Jones calcula as referencias das matrizes para cada retângulo logo quando passa para a matriz seguinte já existe conhecimento prévio dos retângulos anteriores não sendo necessário fazer novamente todos os cálculos.

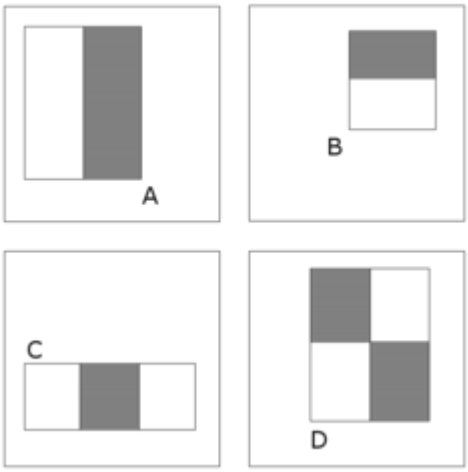

Figura 18 – As Características Básicas do Tipo Haar: (a) Divisão horizontal, (b) Divisão vertical, (c) Duas divisões verticais e (d) Divisões horizontais e verticais [Jones and Viola 2003].

Algoritmo de aprendizagem

A velocidade com que os recursos podem ser avaliados não compensam adequadamente o número, no entanto, por exemplo, num pixel padrão numa sub-matriz de 24x24, há um total de 45,396 características possíveis, o que tornaria muito dispendioso avaliar todos. Assim, o quadro de deteção de objetos emprega uma variante do algoritmo de aprendizagem AdaBoost tanto para selecionar os melhores recursos como para formar os classificadores.

Arquitetura em cascata

A avaliação dos classificadores fortes (ou seja, apresentam uma taxa de erro mínima) gerados pelo processo de aprendizagem pode ser feita rapidamente, mas não é suficientemente rápida para ser executada em tempo real. Por esta razão, os classificadores fortes são dispostos em cascata, a complexidade dos sucessivos classificadores é treinado apenas nas amostras selecionadas

que passam através dos classificadores anteriores. Se em qualquer fase da cascata um classificador rejeita a sub-matriz de inspeção, não é realizado nenhum processamento adicional e continua a procurar a próxima sub-matriz (ver figura 20). A cascata, por conseguinte, tem a forma de uma árvore degenerada [17, 18, 19]. No caso das faces, o primeiro classificador da cascata é chamado por operador inicial e utiliza apenas duas características para alcançar uma taxa de falsos negativos de aproximadamente 0% e uma taxa de falsos positivos de 40%. O efeito deste classificador único é reduzir em para metade o número de vezes que a cascata é avaliada.

A arquitetura em cascata tem implicações para o desempenho dos classificadores individuais. Como a ativação de cada classificador depende inteiramente do comportamento de seu antecessor, a taxa de falso positivo para uma cascata inteira é [17, 18, 19]:

$$F = \prod_{i=1}^{K} f_i.$$

Da mesma forma, a taxa de deteção é a seguinte:

$$D = \prod_{i=1}^{K} d_i.$$

Para coincidir com as taxas de falsos positivos que normalmente são alcançados por outros algoritmos, cada classificador pode começar por se afastar e ter um desempenho surpreendentemente pobre. Por exemplo, para uma cascata de 32 estados a taxa de falsos positivos é de 10^{-6}, cada classificador só precisa de ter uma taxa de falsos positivos de cerca de 65%, no entanto, cada classificador só é excecionalmente capaz, se atingir taxas de deteção adequadas, se por exemplo, queremos conseguir uma taxa de deteção de aproximadamente 90% cada classificação na cascata precisa de conseguir um coeficiente de deteção de aproximadamente 99,7%.

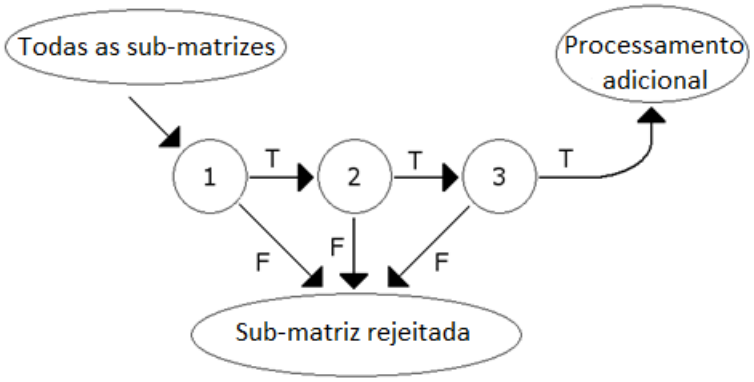

Figura 19 - Arquitetura de cascada

Sendo assim, ao escolher um objeto para a deteção, o algoritmo consiste basicamente em aplicar as características retangulares nas imagens, encontrando um conjunto de características que representam o objeto desejado. Através da aplicação do algoritmo de aprendizagem AdaBoost, filtra-se o conjunto diminuindo o número de características geradas. Ao final, com essas características constrói-se um classificador para aquele objeto.

Para desenvolver um classificador, é necessário usar dois tipos de imagens: positivas e negativas. As imagens positivas devem conter o objeto alvo a ser detetado, já as imagens negativas são imagens de fundo, que não possuem o objeto de interesse. Com esses dois conjuntos de imagens o treino é aplicado.

3.3.3 Ficheiro *"Haar Cascade Classifier"*

Os *"Haar Cascade Classifier"* [21, 22, 23] são ficheiros XML construídos com base nos classificadores de características dos objetos estudados. Para a construção destes ficheiros utiliza-se software fornecido pele API OpenCV [3, 4]

Para a criação destes ficheiros temos que seguir os seguintes passos:

1. API OpenCV disponibiliza ferramentas necessárias para a criação dos ficheiros *"Haar Cascade Classifier"*.

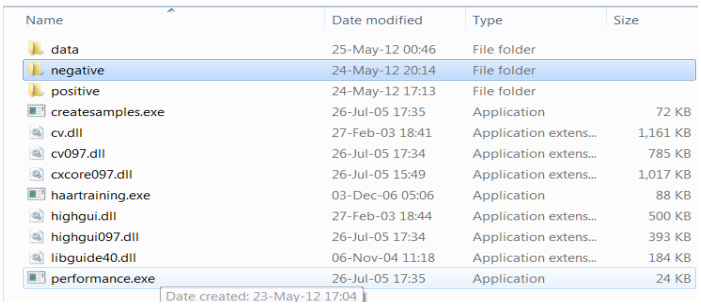

Figura 20 - Pasta para o treino e criação de ficheiros *"Haar Cascade Classifier"*.

2. Temos de ter uma base de dados de imagens positivas e negativas, normalmente as imagens negativas devem ser o dobro das imagens positivas pois os algoritmos mencionados em cima trabalha sobre resultados negativos.
    a. Imagens positivas são todas as que possam identificar o objeto inequivocamente
    b. Imagens negativas são todas a que não identificam o objeto, normalmente são constituídas por fundos onde o objeto possa estar.
3. De seguida colocamos todas as imagens positivas na pasta "positive" e com uma ferramenta para trabalhar as imagens vamos contornar o objeto, as coordenadas do objeto contornado são geradas e guardadas num ficheiro TXT.

```
rawdata/B-train001.bmp 1 7 15 43 44
rawdata/B-train002.bmp 1 10 19 48 48
rawdata/B-train003.bmp 1 14 15 35 47
rawdata/B-train004.bmp 1 11 17 35 44
rawdata/B-train005.bmp 1 15 14 37 48
rawdata/B-train006.bmp 1 15 16 35 47
rawdata/B-train007.bmp 1 14 16 41 46
rawdata/B-train008.bmp 1 17 19 36 44
rawdata/B-train009.bmp 1 14 16 38 52
```

Figura 21 - Aspeto do ficheiro TXT das imagens positivas depois de tratadas, com as coordenadas do objeto.

4. Depois de termos todas as imagens positivas trabalhadas vamos colocar a imagens negativas na pasta chamada *"negative"*, a seguir executamos o ficheiro *"create_list.bat"* que se encontra nessa pasta, ele vai criar um ficheiro TXT com a referência para as imagens negativas.

```
0.bmp
01novel.1.600.bmp
124020.bmp
1256290319-191853525.bmp
1256290431-1354793674.bmp
1256290756-869792524.bmp
```

Figura 22 - Aspeto do ficheiro TXT das imagens negativas, com as referências para essas imagens

5. Para criar as amostras de treino *"Haar"* temos de executar o seguinte comando [21, 22, 23]: ***"createsamples.exe -info positive/info.txt -vec data/vector.vec -num 527 -w 24 -h 24 -maxxangle 0.6 -maxyangle 0 -maxzangle 0.3 -maxidev 100 -bgcolor 0 -bgthresh 0"***
    - -info – parâmetro para identificar o ficheiro TXT das imagens positivas;
    - -vec – parâmetro para criar um ficheiro de amostras VEC;
    - -num – parâmetro para indicar o número de imagens positivas trabalhadas e guardadas no ficheiro TXT;
    - -w – parâmetro que identifica a largura da sub-matriz;
    - -h – parâmetro que identifica a altura da sub-matriz;
    - -maxxangle – parâmetro que indica a amplitude máxima do angulo no eixo dos XX. Varia entre 0 e 1;
    - -maxyangle – parâmetro que indica a amplitude máxima do angulo no eixo dos YY. Varia entre 0 e 1;
    - -maxzangle – parâmetro que indica a amplitude máxima do angulo no eixo dos ZZ. Varia entre 0 e 0.5;
    - -maxidev – parâmetro que indica a máxima intensidade do desvio. Varia entre 0 e 40;

- -bgcolor – parâmetro que indica a cor de fundo, 0 para tons de cinza e 1 para imagens RGB;
- -bgthresh – parâmetro que indica o "*threshold*". Varia entre 0 e 80.

Além destes parâmetros podemos utilizar ainda os seguintes:

- -img – parâmetro que indica o caminho para a imagens a ser treinada, quando estamos na presença de uma única imagem;
- -bg – parâmetro que indica o caminho para o ficheiro que tem as imagens negativas configuradas;
- -show – parâmetro que indica a escala para visualizarmos as imagens;

6. Depois de criadas as amostra podemos começar a treinar o nosso *"Haar"*, para isso executamos o seguinte comando [21, 22, 23]:
   "haartraining.exe -data data/cascade -vec data/vector.vec -bg negative/infofile.txt -npos 527 -nneg 1142 -nstages 30 -mem 1000 -mode ALL -w 24 -h 24 -nonsym"
   - -data – parâmetro para identificar a pasta onde ficam guardados os ficheiros de classificação;
   - -vec – parâmetro para identificar o local onde se encontra o ficheiro VEC;
   - -bg – parâmetro que identifica o local onde está o ficheiro das imagens negativas;
   - -npos – parâmetro para indicar o número de imagens positivas;
   - -nneg - parâmetro para indicar o número de imagens negativas;
   - -vec – parâmetro para criar um ficheiro de amostras VEC;
   - -nstages – parâmetro para indicar o número de estados que queremos para o nosso *"haar";*
   - -mem – parâmetro para indicar a quantidade de memória a disponibilizar para o treino;
   - -mode – parâmetro para identificar qual o modo de treino que desejamos no casso ALL utiliza o conjunto completo de rotação na vertical e 45º;
   - -w – parâmetro que identifica a largura da sub-matriz;
   - -h – parâmetro que identifica a altura da sub-matriz;
   - -sym(*default*)/-nonsym – parâmetro para especificar se a classe de objeto em estudo tem simetria vertical ou não.

   Além destes parâmetros podemos utilizar ainda os seguintes:

   - -nsplits – parâmetro que indica o número de divisões, normalmente escolhe-se o valor 1;

- -minhitrate – parâmetro que indica a mínima taxa de acerto, por defeito o valor é de 0.995000. Varia entre 0 e 1;
- -maxfalsealarm – parâmetro que indica máxima taxa de falsos alarmes, por defeito o valor é de 0.500000. Varia entre 0 e 0.5;
- - weighttrimming – parâmetro que indica peso de corte, por defeito o valor é de 0.950000. Varia entre 0 e 1;
- -minpos – parâmetro que indica o número mínimo de exemplos positivos;
- -maxtreesplits – parâmetro que indica o número máximo de divisões na árvore.

```
+----+----+-+---------+---------+---------+---------+
|  N |%SMP|F|  ST.THR |     HR  |     FA  | EXP. ERR|
+----+----+-+---------+---------+---------+---------+
|   1|100%|-|-0.857040| 1.000000| 1.000000| 0.082075|
+----+----+-+---------+---------+---------+---------+
|   2|100%|+|-1.702127| 1.000000| 1.000000| 0.102168|
+----+----+-+---------+---------+---------+---------+
```

Figura 23 - Janela de treino: após a ter executado o comando anterior N – núneno de iteração de treino, %SMP - percentagem de amostras originais, F - + Indica que o recurso é invertido e - se o recurso não é invertido, ST.THR - limiar de estágio, HR - taxa de acerto, FA - taxa de falso alarme. FYI: (FC, FA) = (1,0, 1,0), EXP.ERR - erro esperado (erro de classificação).

7. Para terminar, passamos o conteúdo da pasta *"data"* para a pasta *"cascade2xml"* e executamos o ficheiro *"convert.bat"* que se encontra nessa pasta. Este ficheiro depois de executado cria um ficheiro *"output.xml"* que é o nosso ficheiro *"Haar Cascade Classifier"* que tem todas as características para a identificação e reconhecimento do objeto em causa.

```
<?xml version="1.0"?>
<opencv_storage>
<haarcascade type_id="opencv-haar-classifier">
  <size>
    160 20</size>
  <stages>
    <_>
      <!-- stage 0 -->
      <trees>
        <_>
          <!-- tree 0 -->
          <_>
            <!-- root node -->
            <feature>
              <rects>
                <_>
                  78 8 5 6 -1.</_>
                <_>
                  78 11 5 3 2.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>-1.5556870494037867e-03</threshold>
            <left_node>1</left_node>
            <right_val>-1.0000059604644775e+00</right_val></_>
          <_>
            <!-- node 1 -->
            <feature>
              <rects>
                <_>
                  0 13 142 6 -1.</_>
                <_>
                  71 13 71 6 2.</_></rects>
              <tilted>0</tilted></feature>
            <threshold>-2.4272960424423218e-01</threshold>
            <left_val>-7.6678442955017090e-01</left_val>
            <right_val>1.</right_val></_></_></trees>
      <stage_threshold>-7.6678442955017090e-01</stage_threshold>
      <parent>-1</parent>
      <next>-1</next></_>
    <_>
      <!-- stage 1 -->
      <trees>
```

Figura 24 - Exemplo de um ficheiro XML de treino de imagens

Na imagem que se segue podemos ver como o algoritmo de aprendizagem AdaBoost contribui para a criação do ficheiro *"Haar Cascade Classifier"*, exigido pelo algoritmo de Viola e Jones [20].

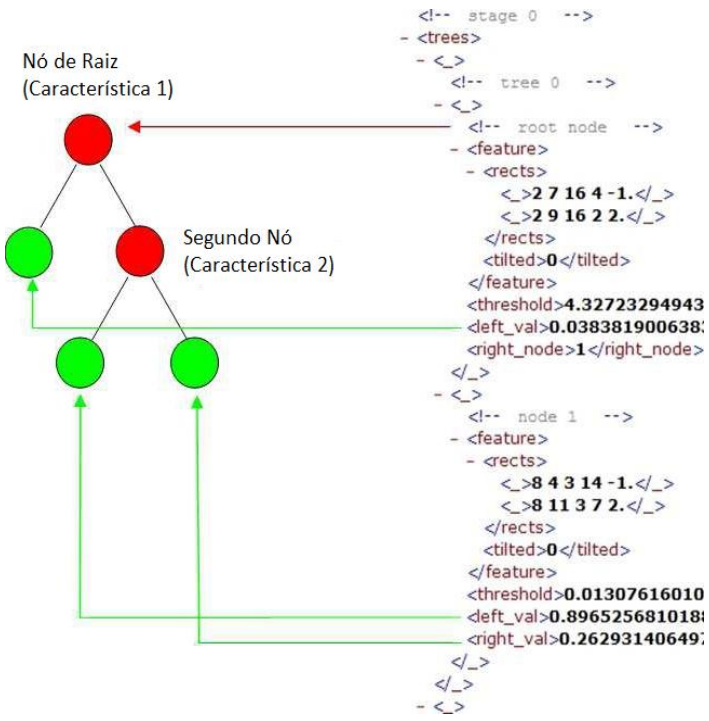

Figura 25 - Contribuição do algoritmo *AdaBoost* [17, 18, 19], para a criação do ficheiro *"Haar Cascade Classifier"* [21, 22, 23].

Neste capítulo foram abordados os métodos estudados para a realização deste trabalho, onde se pode salientar os métodos de segmentação e de deteção de bordas no primeiro método e o algoritmo AdaBoost e a *framework* Viola–Jones para o segundo método, pode-se ver também todo os passos a seguir, para a construção de algoritmos capazes para o reconhecimento de imagens.

No capítulo seguinte vamos ver a implementação e o desenvolvimento da solução, e ver como os métodos anteriores foram aplicados para o desenvolvimento do sistema de reconhecimento capaz.

Vamos ver também as linguagens de programação utilizadas e a forma como se complementam e interagem entre elas.

Capítulo 4

Desenvolvimento e Implementação

4.1 JNI, NDK e SDK Android

Antes de abordar a implementação do sistema, abordamos um pouco do *Java Native Interface* (JNI) [7, 8], NDK [5] e SDK Android [5] pois todos os algoritmos sobre imagens foram implementados em C/C++ com NDK Android [5] e ligação com o SDK do Android [5] que é programado em Java.

O Android é um sistema operativo baseado em Linux construído pela Google para *smartphones*. A primeira versão foi comercializada em Setembro de 2008, versão 1.0 do sistema operativo, e atualmente encontra-se já na versão 4.1. A figura seguinte representa a arquitetura do sistema operativo Android[5].

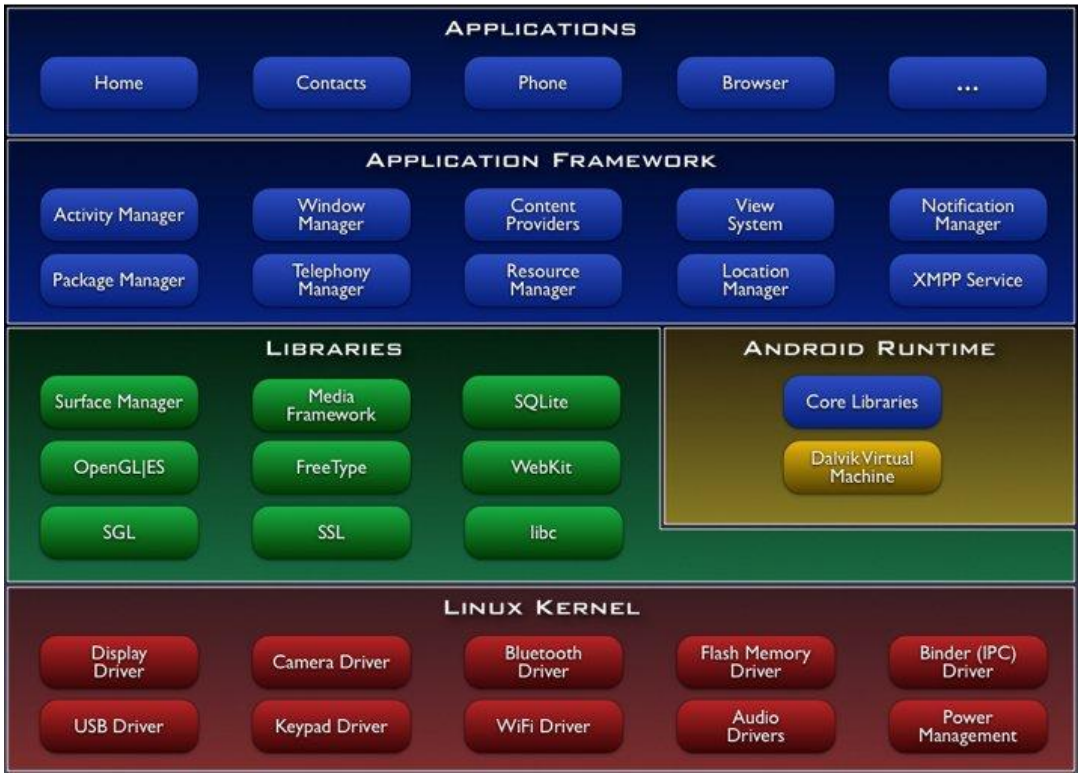

Figura 26- Arquitetura do sistema operativo Android[5].

Neste projeto todo o desenvolvimento da API foi feito em C/C++ que se enquadra na biblioteca ***libc***. O JNI serviu de ligação entre a biblioteca ***libc*** e a ***framework***, ou seja a biblioteca ***libc*** corresponde ao NDK Android e a ***framework*** corresponde ao SDK Android.

SDK Android - Normalmente os SDKs são disponibilizados pelas empresas para que programadores externos tenham uma melhor integração com o *software* proposto. Exemplo de um

SDK é o Android SDK que inclui documentação, código e utilitários para que programadores consigam desenvolver as suas aplicações de acordo com um padrão de desenvolvimento para o sistema operativo em questão.

NDK Android - O *Android NDK* [5] é uma ferramenta complementar para o Android SDK que permite construir partes das aplicações em código nativo. Enquanto o SDK Android fornece um ambiente de programação muito rica, o *Android NDK* amplia os horizontes e pode acelerar o desempenho das nossas aplicações pois como é escrito em c e c++ esta mais próximo da linguagem máquina, o que lhe confere alguns privilégios a trabalhar com o *hardware*.

JNI – *Java Native Interface* (JNI) [7, 8] é uma *framework* de programação que permite que o código executado pela *Java Virtual Machine* possa chamar e ser chamado por aplicações nativas (aplicações específicas para um hardware e sistema operativo) e por bibliotecas escritas em outras linguagens, tais como C, C++, Delphi e Assembly.

Na figura que se segue apresenta-se a arquitetura de um programa onde foi implementada a *framework* JNI [7, 8], esta *framework* utiliza uma nomenclatura muito especifica para fazer a ligação entre o código nativo e o java o método em linguagem de programação nativa tem que começar sempre por "JNIEXPORT" seguido do tipo de resultado que retorna e da sigla "JNICALL", o nome do método tem que fazer referencia ao Java e ao package do projeto Java por exemplo "Java_HelloWorld_print", os objetos passados para o método podem ser de qualquer tipo desde que suportado pela linguagem nativa e o nome do objeto tem que começar por "j" por exemplo "jint" para um inteiro, existe um objeto obrigatório que se denomina por "JNIEnv *" este objeto tem que existir em todos os métodos chamados posteriormente pelo projeto JAVA.

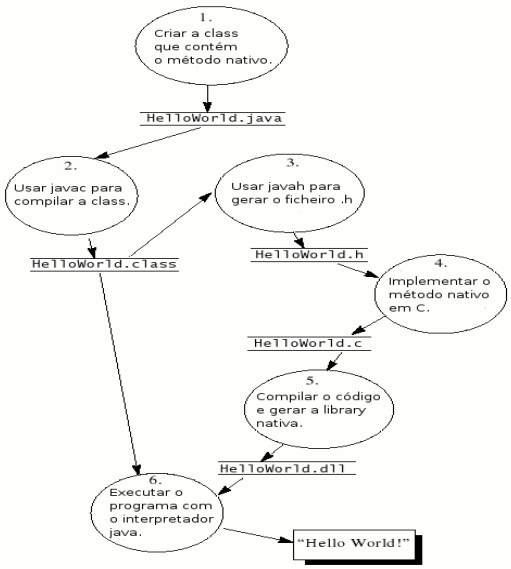

Figura 27 - Arquitetura do JNI [7, 8].

Exemplo de uma class em código nativo para ser suportado pelo JNI [7, 8]:

```
JNIEXPORT void JNICALL Java_HelloWorld_print (JNIEnv *, jobject);
```

Como podemos observar, todos os objetos começam por "j" identificam o objeto pertence a *framework*.

No JAVA a chamada é feita da seguinte forma:

```
static {
        System.loadLibrary("camera");
}
public native void print();
```

Neste projeto os métodos de pré-processamento, segmentação e deteção de objetos presentes na API desenvolvida foram implementados em C++ com o NDK Android. A aplicação para o *smartphone* foi implementada em JAVA com o SDK Android. A ligação entre o NDK e o SDK foi conseguida com a utilização da *framework* JNI descrita em cima.

4.2 Sistema Proposto

No desenvolvimento do sistema foram usadas as técnicas mencionadas atrás, em seguida vamos explicar como e quais técnicas foram utilizadas, a ordem como foram aplicadas e como ajudaram a obter o resultado final.

Neste projeto foram desenvolvidos dois métodos diferentes de reconhecimento, um por segmentação e o respetivo reconhecimento, este ficou aquém das espectativas visto que o ambiente onde foi feito o reconhecimento tinha que ser muito restrito e só reconhecia um objeto de cada vez, logo ficou um pouco longe do pretendido neste projeto. O outro método implementado foi o que se adequou mais ao objetivo proposto que é o de treino do objeto, este último satisfaz as nossas pretensões em relação ao estudo proposto.

4.2.1 Primeiro Método Estudado

O primeiro método analisado e testado foi o de segmentação da imagem e respetivo reconhecimento.

Os passos para a implementação deste método são descritos a seguir:

1. Pré-processamento: Conversão RGB para cinza segundo apresentado na secção 3.1.1. Este passo permite simplificar os passos que se seguem. O resultado deste passo é apresentado na figura 5.

2. Segmentação por K-means para separação do objeto de estudo: Podemos ver que no resultado apresentado na figura 6 já conseguimos ter tons de cinza mais homogéneos para os objetos.

3. Deteção de contornos pelo operador Canny: Escolhemos o operador de Canny porque é o mais completo e o mais utilizado para o tipo de estudo realizado, pareceu-nos também a melhor deteção de linhas nos objetos estudados.

4. Separação do objeto em estudo: Como para este método de estudo preparamos um cenário específico uma mesa clara com o objeto por cima, não foi necessário passar pelo algoritmo de "Region Growing", foi feita a separação do objeto recorrendo a um algoritmo implementado neste projeto que explicamos mais à frente.

5. Deteção de características: Após a separação do objeto aplicamos o algoritmo de deteção de características e tiramos as medidas descritas em cima no ponto 3.2.7.

6. Guardar resultados referentes ao objeto: De seguida guardamos todas as informações necessárias para o reconhecimento do objeto numa pequena base de dados com apenas uma tabela com os campos:
    - Nome
    - Número de características
    - Altura
    - Largura
    - Medida Superior
    - Medida Inferior
    - Medida direita
    - Medida esquerda

7. Reconhecimento: Com o objeto e as características guardadas na base de dados, quando apontamos a câmara do *smartphone* para o objeto este vai aparecer marcado com um retângulo e respetiva identificação.

Na figura que se segue podemos observar o diagrama de sequência associado a esta implementação.

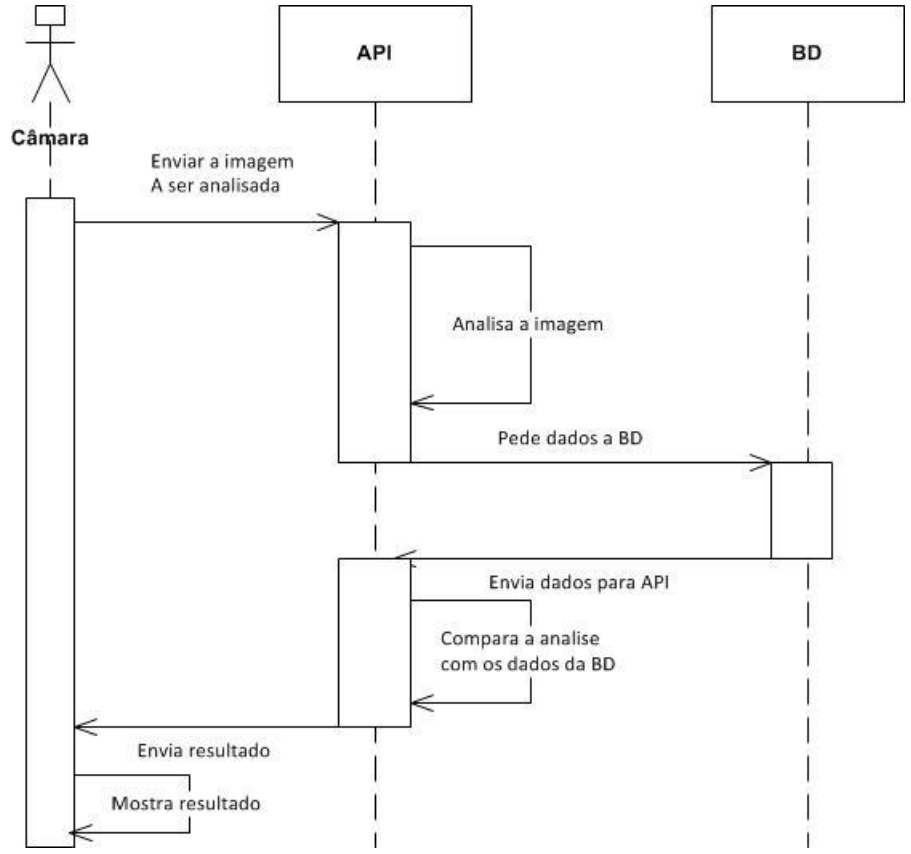

Figura 28 - Diagrama de sequência para o método apresentado

De seguida vamos detalhar cada uma destes passos e apresentar alguns detalhes da implementação que se mostrem exemplificativos.

Pré-processamento - A primeira fase do sistema foi converter a imagem para tons de cinza para a aplicação dos algoritmos que se seguem.

Segmentação por K-means – O primeiro algoritmo utilizado foi o K-means com um K = 2. Foi escolhido k=2 para podermos ficar apenas com uma cor para a mesa e outra para o objeto.

- Primeiro calculamos o histograma da imagem em tons de cinza
- De seguida dividimos os valores em dois intervalos a cor que tinha mais incidência que representava a mesa e as outras cores que representavam o objeto.

- Feita a divisão calculamos a média das cores do objeto e colorimos todo objeto com essa média de cor, ficando assim com as duas cores pretendidas. A figura a seguir mostra a implementação do algoritmo K-means com um K = 2:

A figura que se segue mostra um exemplo da implementação do algoritmo K-means com o K=2.

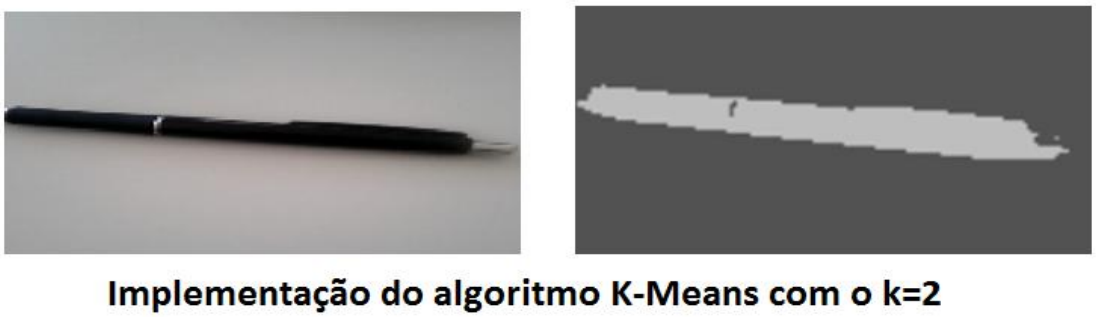

Figura 29 - Imagem com as diferentes regiões após execução do algoritmo K-means.

Deteção de contornos pelo operador Canny - Optamos pelo operador de Canny para a deteção de contornos por nos parecer o mais adequado para este estudo. A deteção de contornos pelo operador Canny foi implementada como descrito no ponto 3.2.7 e os resultados podem ser vistos na imagem que se segue.

A figura seguinte mostra a implementação dos algoritmos K-means com o K=2 e de seguida a implementação do operador de Canny.

Figura 30 - Deteção de contorno Canny

Separação do objeto em estudo – Com base na imagem resultante do operador de Canny foi implementado um algoritmo para retirar o objeto da imagem e conseguir verificar a sua posição e a matriz correspondente para a deteção do próprio. O algoritmo tem as seguintes etapas:

- Percorremos a matriz de entrada em quatro direções deferentes e guardamos as posições dos primeiros pixéis brancos que aparecem e traçamos linhas nessas direções logo ficamos com uma sub-matriz que contem o objeto.

A figura seguinte ilustra a implementação do ponto anterior. Como podemos observar temos os pontos necessários para passar ao ponto seguinte.

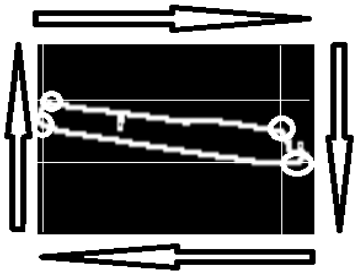

Figura 31 - Resultado da implementação do passo em cima descrito

- Após encontrar as linhas descritas em cima procuramos os pontos de interceção e retiramos todos os pontos que ficam para fora do objeto.

A figura que se segue ilustra a implementação do ponto anterior. Podemos também observar que neste ponto temos apenas o objeto assinalado.

Figura 32 - Resultado do passo anterior.

- De seguida retiramos a sub-matriz onde se encontra o objeto e ficamos assim só com a matriz do nosso objeto para posteriormente ser analisada.

A figura que se segue ilustra a implementação do ponto anterior. Observamos também que temos o objeto assinalado na imagem real.

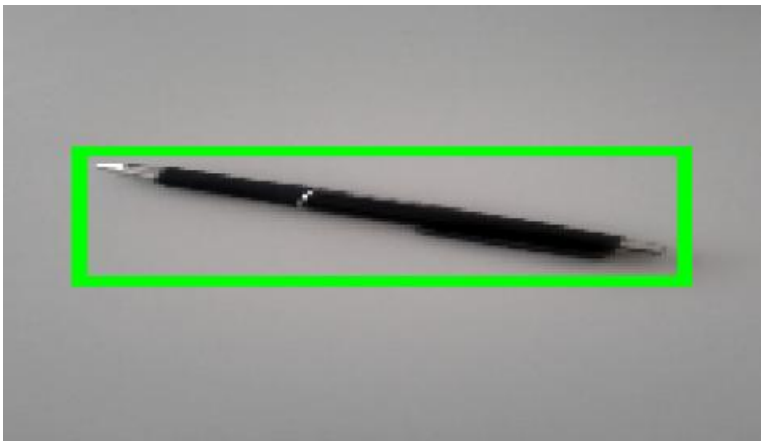

Figura 33 - Imagem resultante da implementação do algoritmo descrito em cima

Deteções de características – Neste passo são guardadas todas as informações referentes ao objeto do seguinte modo:

- Procurar o número de características encontradas no objeto em estudo como descrito anteriormente.

A figura que se segue ilustra a implementação do ponto anterior, com as características assinaladas.

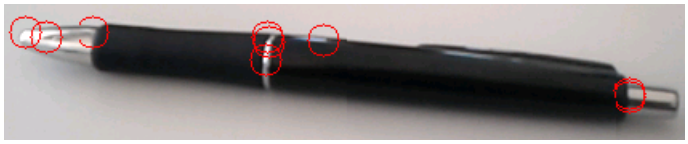

Figura 34 - Deteção de características do objeto

Guardar resultados referentes ao objeto – Neste passo guardamos os dados referentes ao objeto na nossa base de dados.

Reconhecimento – Esta é a fase final no nosso primeiro método, neste passo temos a câmara do *smartphone* a apontar para o nosso objeto, e a comparar todas a *frames* resultantes da câmara com os objetos guardados na base de dados, é de salientar que a câmara esta a filmar 15 *frames* por segundo o que pode levar a um pequeno atraso na visualização da imagem e respetivo conhecimento. Neste caso que apresentamos na imagem que se segue o tempo de reconhecimento foi de 0,345s, mas temos que ter em atenção ver que a base de dados tinha apenas as características de dois objetos. Se tivéssemos mais características de objetos na base de dados o tempo vai aumentar.

A figura que se segue ilustra a implementação do primeiro método estudado, com respetivo reconhecimento do objeto, observamos que o objeto está assinalado e respetivamente identificado.

Figura 35 - Reconhecimento do objeto

Como podemos observar, este método é um pouco restrito e requer um cenário específico, por esse motivo tivemos que encontrar um método alternativo de reconhecimento, e este método não foi mais atualizado nem utilizado ao longo do projeto. O método que se segue é o mais indicado para este tipo de projeto e por isso incidimos mais sobre ele.

4.2.2 Segundo Método Estudado

Neste método recorremos à implementação dos algoritmos de reconhecimento referidos no ponto 3.3.

1. Recolha de imagens: Foram recolhidas imagens dos objetos pretendidos em várias posições e a várias distâncias, chamamos as estas imagens "imagens positivas". Foram também recolhidas imagens do cenário onde o objeto pode se colocado (imagens de fundo), chamamos a estas imagens "imagens negativas".

2. Criar um Ficheiro *"Haar Cascade Classifier"*: Aplicamos todos os passos para a criação do ficheiro "Haar Cascade Classifier" descritas no ponto 3.3.3.

3. Passar as características presentes no ficheiro para a memória: Com o ficheiro *"Haar Cascade Classifier"* criamos um objeto do tipo *"CvHaarClassifierCascade()"* da API OpenCV e reservamos memória para esse objeto com a função *"CvMemStorage.create()"*.

4. Criar um *array* para os objetos encontrados: Criamos um *array* do tipo *"CvSeq()"* da API OpenCV[3, 4] para armazenar o objetos detetados com o ficheiro *"Haar Cascade Classifier"* a implementação deste ponto é feito da seguinte forma:

```
cvSeq.add(cvHaarDetectObjects(grayImage, classifier, storage, 1.1, 3, CV_HAAR_DO_CANNY_PRUNING));
```

   - grayImage – Parâmetro que representa a imagem em tons de cinza.
   - classifier – Parâmetro que representa o nosso objeto *"CvHaarClassifierCascade()"*.
   - storage – Parâmetro para a quantidade de memória reservada.
   - 1.1 – Fator de escala.
   - 3 – mínimo de vizinhos
   - CV_HAAR_DO_CANNY_PRUNING – Flag que indica o algoritmo de deteção de contornos a ser utilizado para reconhecimento do objeto.
5. Reconhecimento: Com os objetos e as características detetadas, quando apontamos a câmara do smartphone para o objeto ele vai aparecer marcado com um retângulo e respetiva identificação.

Na figura que se segue podemos ver o diagrama de sequência para este método.

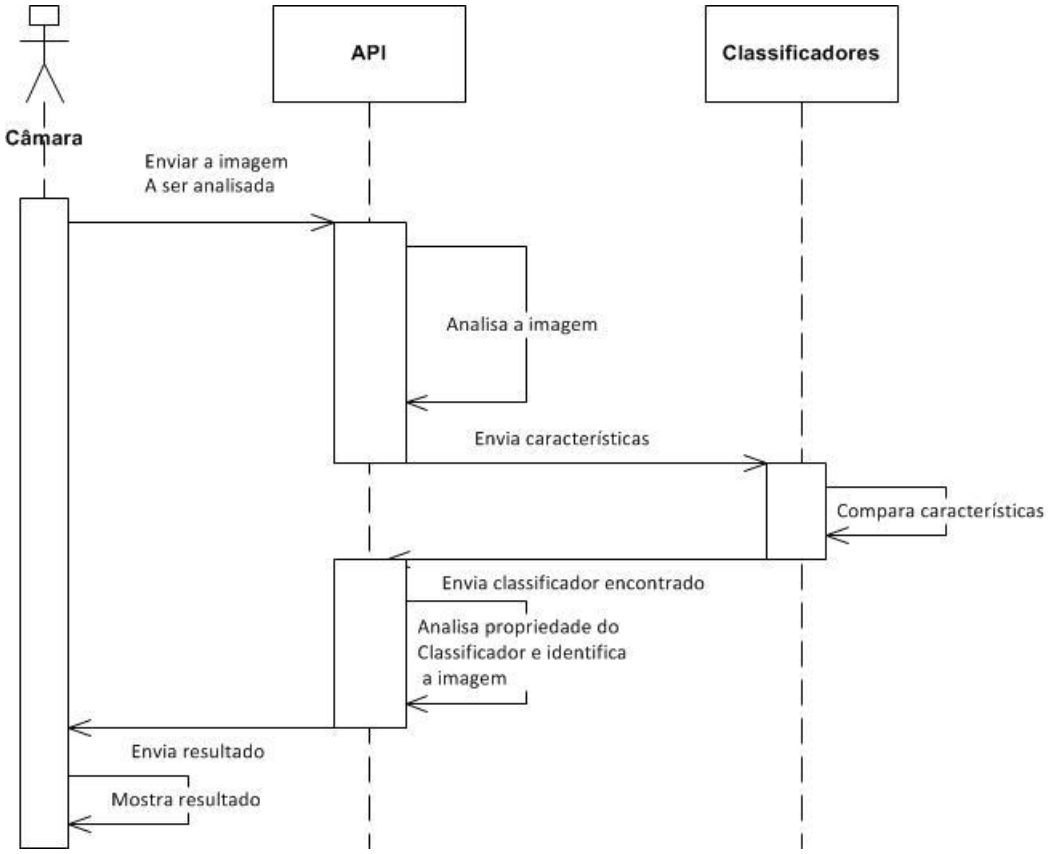

Figura 36 - Diagrama de sequência para o método apresentado

Recolhas de imagens - Imagens positivas foram recolhidas por um *smartphone* Android, com a versão 4.0.4 do sistema operativo, com uma resolução de 5 MPixies, e outras foram retiradas da página de referência da API OpenCV [3, 4], para treino das mesmas, na figura que se segue temos exemplos de imagens que servira para o treino e respetivo reconhecimento. Foram realizados treinos para imagens de mãos, face e corpo humano, neste último foram também treinadas imagens que continham apenas a parte de cima do corpo humano e outras que continham apenas a parte debaixo do corpo humano.

Figura 37 - Exemplos de imagens para treino e respetivo reconhecimento (Imagens Positivas)

As imagens negativas são compostas por fundos onde o objeto pode ser inserido, foram recolhidas por um *smartphone* Android, com a versão 4.0.4 do sistema operativo, com uma resolução de 5 MPixies, previamente convertidas para tons de cinza pelo software da câmara do *smartphone* e retiradas da pagina de referência da API OpenCV [3, 4], já convertidas em tons de cinza, como mostra a seguinte figura.

Figura 38 Exemplos de imagens fotografadas para treino e respetivo reconhecimento (imagens negativas)

Criar um Ficheiro *"Haar Cascade Classifier"* – Para a criação deste ficheiro foram efetuados os seguintes passos [3, 4]:

1 – Criação de um ficheiro "*vec"* com as imagens positivas com o seguinte comando,

"createsamples -img image.png -num 7000 -bg negatives/infofile.txt -vec data/samples.vec -maxxangle 0.6 -maxyangle 0 -maxzangle 0.3 -maxidev 100 -bgcolor 0 -bgthresh 0 -w 24 -h 24"

Figura 39 - Exemplo de imagens no ficheiro vec

Como podemos observar este comando insere imagens positivas em vários pontos aleatórios e em ângulos diferentes nas imagens negativas, esses ângulos variam conforme o escolhido quando escrevemos o comando "*createsamples*", neste caso foi escolhido uma variação máxima para o eixo dos XX de 0.6 e uma variação máxima para o eixo dos ZZ de 0.3, esta ultima para podermos ter reconhecimento 3D, para o eixo dos YY não foi escolhida nenhuma variação porque não fazia muito sentido o reconhecimento na posição longitudinal dos referidos objetos.

2- Passamos agora ao treino do ficheiro de classificação, para tal executamos o seguinte comando [3, 4]:

"haartraining -data cascade -vec data/samples.vec -bg negatives/infofile.txt -nstages 30 -nsplits 2 -minhitrate 0.999 -maxfalsealarm 0.5 -npos 7000 -nneg 3019 -w 24 -h 24 -nonsym -mem 1512 -mode ALL"

Na imagem seguinte está um exemplo do ficheiro resultante do treino.

```
<?xml version="1.0"?>
<opencv_storage>
<cascade type_id="opencv-haar-classifier">
  <size>
    24 24</size>
  <stages>
    <_>
      <!-- stage 0 -->
      <trees>
        <_>
          <!-- tree 0 -->
          <_>
            <!-- root node -->
            <feature>
              <rects>
                <_>
                  17 1 7 6 -1.</_>
                <_>
                  15 3 7 2 3.</_></rects>
              <tilted>1</tilted></feature>
            <threshold>-0.0530747510492802</threshold>
            <left_val>0.9390885829925537</left_val>
            <right_node>1</right_node></_>
          <_>
            <!-- node 1 -->
            <feature>
              <rects>
                <_>
                  5 2 2 5 -1.</_>
                <_>
                  5 2 1 5 2.</_></rects>
              <tilted>1</tilted></feature>
            <threshold>0.0220606196671724</threshold>
            <left_val>-0.9634469151496887</left_val>
            <right_val>0.8174802064895630</right_val></_></_>
        <_>
          <!-- tree 1 -->
          <_>
            <!-- root node -->
```

Figura 40 - Resultado do treino da imagem.

Passar as características presentes no ficheiro para a memória – Neste passo foi reservada memória para o ficheiro criado anteriormente.

Criar um *array* para os objetos encontrados – Neste passo foi criado um *array* do tipo *"CvSeq()"* da API OpenCV [3, 4] para armazenar o objetos detetados com o ficheiro *"Haar Cascade Classifier"*, ficamos assim com os nossos objetos nesse *array* para depois passarmos ao passo seguinte que é o reconhecimento.

Para os dois passos anteriores foi utilizada a linguagem de programação c++, para o passo seguinte foi utilizada a linguagem Java.

Reconhecimento – Neste último passo foi implementado um pequeno programa em java que recebe as coordenadas devolvidas pelos passos anteriores e desenha um retângulo em volta do objeto com o respetivo nome do objeto reconhecido.

A implementação deste método pode ser observada na seguinte figura:

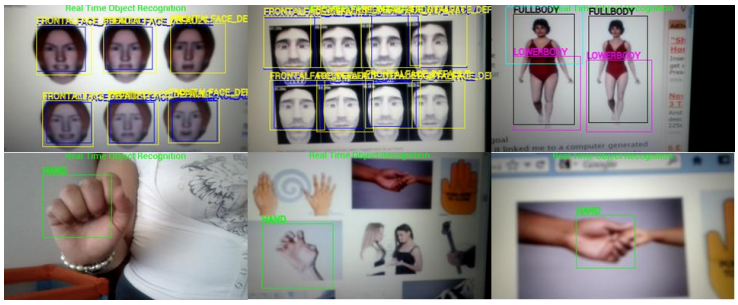

Figura 41 – Reconhecimento de múltiplos objetos

Os resultados e os testes vão incidir sobre este segundo método pois este está mais de acordo com o objetivo pretendido, visto que o primeiro só funciona bem em condições muito específicas e não se adequa ao objetivo proposto. Em baixo podemos ver uma tabela com as comparações dos dois métodos, onde podemos ver que o segundo se adequa mais ao objetivo do projeto.

Tabela 3 - Tabela comparativa dos métodos estudados

	Método 1	Método 2
Reconhecer objetos	Sim	Sim
Reconhecer vários objetos	Não	Sim
Tempo médio de reconhecimento	1,0s (com os objetos em estudo)	1,5s (com os objetos em estudo)
Reconhecer objeto independentemente onde ele se encontra	Não	Sim
Reconhecer os objetos em vários ângulos	Sim	Sim
Reconhecer objetos a várias distâncias	Sim	Sim
Reconhecer 50% ou mais do objeto incompleto	Não	Depende to treino que foi feito na imagem

Como podemos observar na tabela anterior estão representadas as principais diferenças dos métodos apresentados, como podemos ver o método 2 é o que se adequa aos objetivos propostos no projeto, no reconhecimento de vários objetos e no reconhecimento dos mesmos independentemente do fundo onde se encontra o objeto.

Capítulo 5

5.1 Desempenho do Sistema

No capítulo anterior foram apresentados os métodos estudados para a realização deste projeto e como podemos observar o segundo método é mais adequado ao objetivo pretendido neste projeto.

Neste capítulo apresentamos os resultados obtidos com o método referido anteriormente assim com a sua performance em relação aos seguintes critérios:

1. Tempo de reconhecimento dos objetos;
2. Variação da iluminação;
3. Distância em que o reconhecimento é efetuado;
4. Imagens virtuais (reconhecimento de imagens no ecrã do pc);
5. Imagens reais (reconhecimento de imagens em ambiente real);
6. Fundos onde o objeto está inserido.

As imagens utilizadas para o reconhecimento foram retiradas de uma pesquisa no "*Google*" com os objetos pretendidos e captadas em ambiente real, forma feitos testes a 100 imagens diferentes que continham os objetos treinados, em seguida apresento alguns exemplos e demonstrações em 6 imagens escolhidas aleatoriamente das 100 testadas.

Para o reconhecimento foram utilizados ficheiros de classificação fornecidos pela API OpenCV[3, 4], para os seguintes objetos: face frontal, perfil da face, olhos, nariz, boca, parte superior do corpo humano, parte inferior do corpo humano e corpo humano completo, foram também utilizados ficheiros treinados pelo método descrito anteriormente para o reconhecimento da mão.

Podemos ver alguns exemplos das imagens utilizadas na figura que se segue.

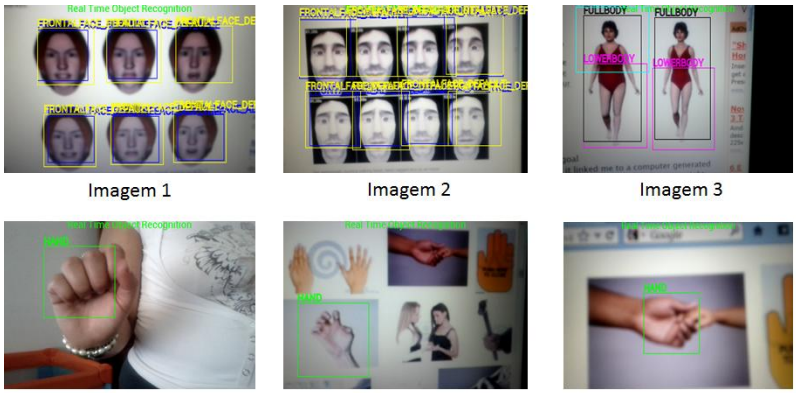

Figura 42 - Exemplos de imagens reconhecidas

- Imagem 1 – Representa imagem virtual de faces
- Imagem 2 – Representa imagem virtual de faces com um fundo deferente da anterior com imagens mais próximas umas das outras
- Imagem 3 – Representa imagem virtual do corpo humano
- Imagem 4 – Representa uma imagem real de uma mão fechada
- Imagem 5 – Representa uma imagem virtual de uma mão com os dedos fechados
- Imagem 6 – Representa uma imagem virtual de uma mão segurando outra

Os resultados obtidos com a implementação deste método foram ao encontro com o pretendido neste projeto e foi feito o reconhecimento de vários objetos ao mesmo tempo em tempo real. Foram feitos testes em vários cenários e o algoritmo respondeu com pretendido. Os cenários de teste foram os seguintes:

Tempo de reconhecimento dos objetos: O tempo de reconhecimento dependeu muito das variantes que se seguem, mas no geral e com um ambiente ideal para o reconhecimento (boa iluminação, uma distância aceitável entre 20cm e 2m dependendo do objeto) o sistema respondeu como esperado, conseguindo reconhecer todos os objetos estudados. O tempo foi medido da seguinte forma: foi inserido um botão no ecrã do *smartphone* para iniciar o reconhecimento, com esse botão era inicializada uma variável ***start*** com o tempo atual do sistema, e quando o sistema reconhecia o objeto registávamos o tempo atual do sistema noutra variável ***end***, ao subtrairmos a variável ***end*** pela variável ***start*** encontrávamos o tempo decorrido para o reconhecimento. O tempo variou como apresentado na seguinte tabela:

Tabela 4 - Tabela com a variação temporal dos vários critérios escolhidos para a avaliação do sistema

	Distância em que o reconhecimento é efetuado	**Imagens virtuais**	**Imagens reais**	**Fundos onde o objeto está inserido**	**Variação da iluminação**		
					Luminosidade Alta	Luminosidade Normal	Luminosidade Baixa
Imagem 1	1505 (ms)	1407 (ms)	-	1822 (ms)	2034 (ms)	1507 (ms)	2032 (ms)
Imagem 2	1756 (ms)	1733 (ms)	-	1976 (ms)	2056 (ms)	1758 (ms)	2114 (ms)
Imagem 3	1632 (ms)	1592 (ms)	-	1867 (ms)	2756 (ms)	1635 (ms)	2556 (ms)
Imagem 4	2087 (ms)	-	2056 (ms)	2058 (ms)	3009 (ms)	2090 (ms)	3706 (ms)
Imagem 5	2032 (ms)	2022 (ms)	-	2321 (ms)	2746 (ms)	2037 (ms)	2867 (ms)
Imagem 6	2114 (ms)	2109 (ms)	-	2404 (ms)	2558 (ms)	2119 (ms)	2756 (ms)

Como podemos observar na tabela anterior, onde o tempo mais oscila é com a iluminação, temos também uma demora um pouco maior no reconhecimento quando temos um fundo que não é liso (que contenha outros objetos ou paisagens), em todos os outros critérios a oscilação do tempo de reconhecimento é mínima.

Variação da iluminação: o sistema não revelou alterações de desempenho, apenas quando a iluminação era muito fraca o sistema demorava um pouco mais de tempo no reconhecimento, também se houvesse reflexo o sistema tinha dificuldades e reconhecer os objetos.

As seguintes figura mostram o comportamento do sistema com a variação da iluminação.

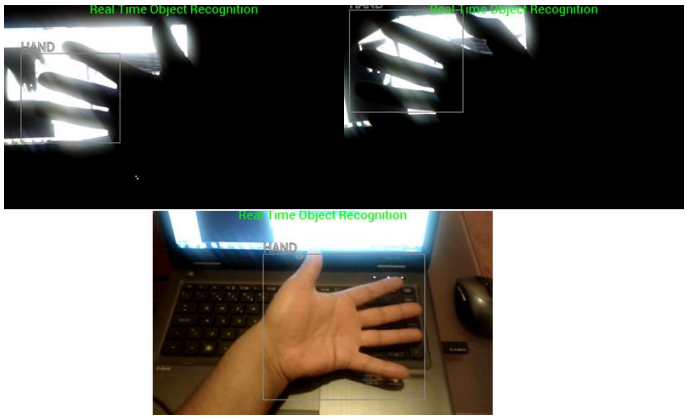

Figura 43 - Exemplo de reconhecimento com variação da iluminação.

Distância em que o reconhecimento é efetuado: A distância é um pouco relativa pois tem a ver com o objeto que se quer reconhecer, para objetos grandes requer uma distância maior do que para os objetos mais pequenos. A distância foi medida da seguinte forma: com *software* disponível na *play Google* com o nome de *"Telêmetro : Smart Measure"* [24] desenvolvida por "*Android boy's Lab*" [25] para medir distâncias com a câmara do *smartphone*, ele mede a distância do objeto à câmara do *smartphone*, através da altura do objeto, depois de medida a distância era feito o reconhecimento dos objetos em causa, assim conseguimos fazer os intervalos de reconhecimento. A tabela que se segue mostra os intervalos de distância para os vários objetos:

Tabela 5 - Intervalos de distância de reconhecimento para objetos de vários tamanhos

	Objetos Pequenos	Objetos Médios	Objetos Grandes
Distância Mínima	10 (cm)	10 (cm)	50 (cm)
Distância Ideal	15 (cm)	20 (cm)	100 (cm)
Distância Máxima	20 (cm)	30 (cm)	200 (cm)

Como podemos ver na tabela anterior, a distância varia com o tamanho dos objeto. Chamamos objetos pequenos aqueles que o seu reconhecimento tem que ser efetuado muito próximo dele como por exemplo a mão humana, nariz, boca e olhos, chamamos objetos médios onde o reconhecimento pode ser feito a uma distância maior que a anterior como por exemplo face, parte de cima do corpo humano, parte debaixo do corpo humano, e objetos grandes são aqueles em que o reconhecimento pode tem que ser feito a uma distância maior, como por exemplo o corpo humano inteiro, é de referir também que estas distâncias são aplicadas em imagens reais, porque ao contrario destas as virtuais dependem unicamente do tamanho da imagem em estudo.

Imagens virtuais: Para as imagens virtuais o reconhecimento foi feito com sucesso em 92% das imagens pesquisadas, foram feitos testes em 100 imagens pesquisadas no Google e o sistema respondeu com sucesso em 92 imagens e as que não reconheceu eram imagens em que os objetos não estavam representados a 100%:

A figura seguinte mostra o reconhecimento em imagens virtuais.

Figura 44 - Exemplo de reconhecimento em imagens virtuais.

Imagens reais: Para as imagens reais o reconhecimento teve sucesso em 95% das imagens, foram testadas em 150 visualizações de faces, mãos e corpo humano e o resultado foi muito satisfatório, nas 100 imagens testadas apenas não respondeu com sucesso quando estávamos a uma distância muito grande para o reconhecimento de mãos e face, e quando nos encontrávamos a uma distância muito curta para o reconhecimento do corpo.

Na figura que se segue podemos ver um exemplo de reconhecimento em imagens reais.

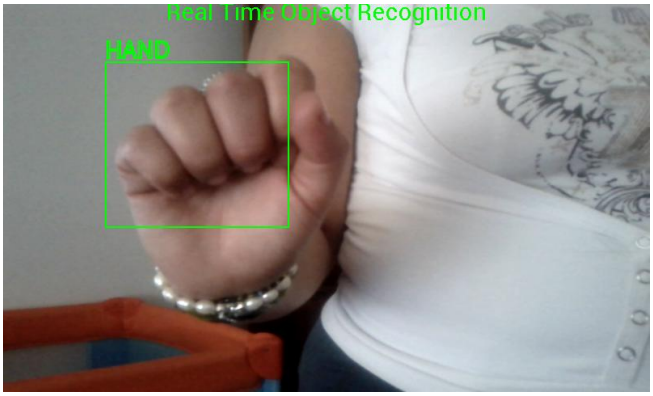

Figura 45 - Exemplo de reconhecimento de uma mão real.

Fundos onde o objeto está inserido: Foram efetuados teste em vários fundos, e o sistema respondeu satisfatoriamente a 98% dos testes. Para este teste foram utilizadas 100 imagens do objeto inseridas em imagens de fundo aleatórias, o reconhecimento só não teve sucesso quando a imagem estava num ângulo em que as propriedades efetuadas para a construção do ficheiro de classificação não eram cumpridas.

Na figura que se segue podemos observar um exemplo do reconhecimento de vários objetos num fundo aleatório, também se pode ver que as imagens que não obedecem ao padrão do ficheiro de classificação não são reconhecidas.

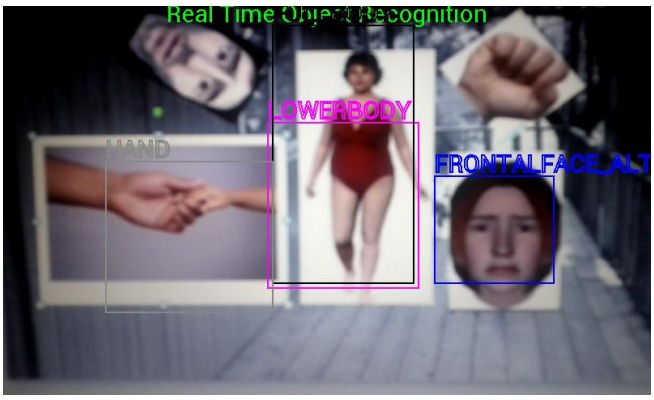

Figura 46 - Exemplo do reconhecimento de imagens num fundo aleatório.

A tabela que se segue é demostrativa dos critérios que usamos para classificar a performance do sistema, onde a classificação pode ser: bom, médio, fraco, dependendo do reconhecimento do objeto em cada um dos critérios, usou-se as imagens exemplo apresentadas em cima para caracterizar o sistema.

Tabela 6 - Classificação da performance em relação aos critérios descritos em cima.

	Distância em que o reconhecimento é efetuado	Imagens virtuais	Imagens reais	Fundos onde o objeto está inserido	Variação da iluminação		
					Luminosidade Alta	Luminosidade Normal	Luminosidade Baixa
Imagem 1	Bom	Bom	-	Médio	Fraco	Bom	Médio
Imagem 2	Bom	Bom	-	Médio	Fraco	Bom	Médio
Imagem 3	Bom	Bom	-	Médio	Fraco	Bom	Médio
Imagem 4	Bom	-	Bom	Bom	Médio	Bom	Bom
Imagem 5	Bom	Bom	-	Médio	Fraco	Bom	Médio
Imagem 6	Bom	Bom	-	Médio	Fraco	Bom	Médio

Ao analisar a tabela anterior podemos verificar que o sistema na maioria dos critérios porta-se como o esperado tendo um reconhecimento bom dos objetos, podemos ver também que só em condições limites é que o comportamento fica um pouco aquém das espectativas, principalmente com a iluminação e com alguns fundos em que existem objetos com características semelhantes aos estudados. Podemos afirmar que o sistema vai de encontro ao objetivo proposto para este projeto.

As figuras que se seguem contêm exemplos de reconhecimento de multiobjectos, nelas podemos ver o comportamento no sistema.

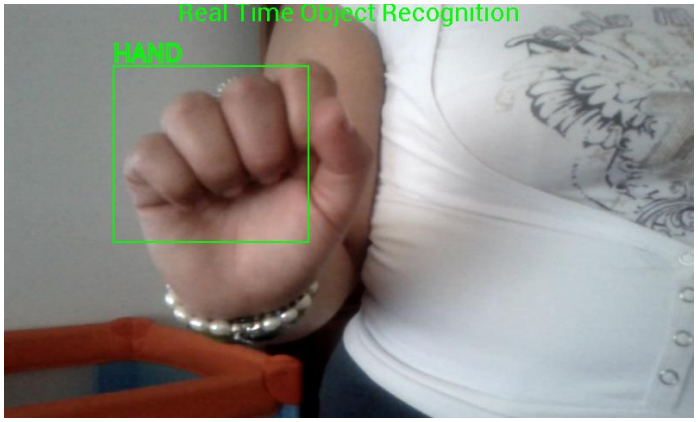

Figura 47 - Reconhecimento da mão em imagem real.

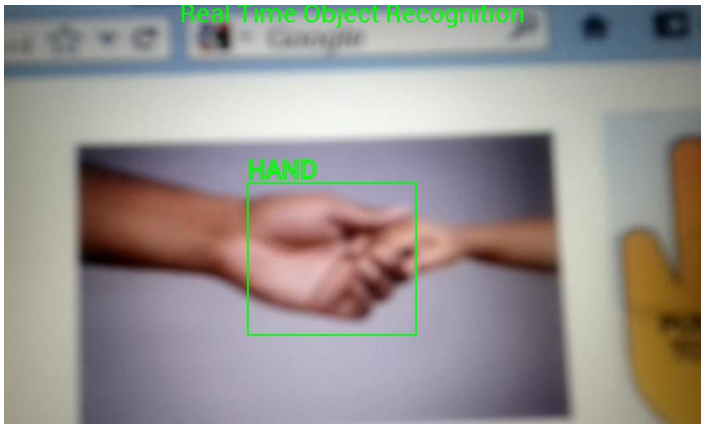

Figura 48 - Reconhecimento da mão em imagem virtual

Figura 49 - Reconhecimento de corpo humano inteiro, parte de cima do corpo humano e parte de baixo do corpo humano

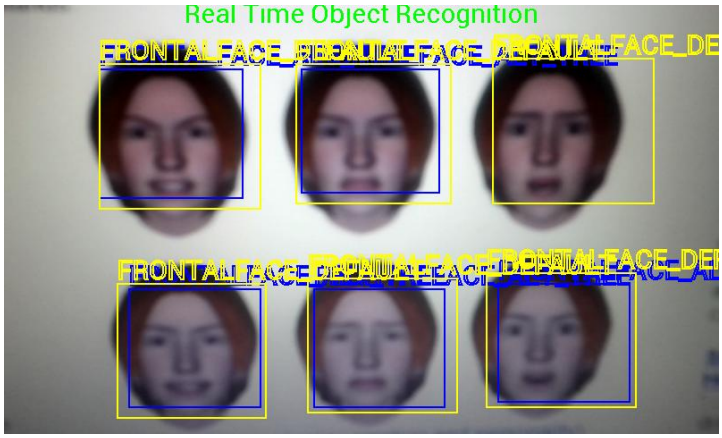

Figura 50 - Reconhecimento da face com várias expressões, por dois ficheiros de classificação deferentes.

Com podemos observar, este método supera em muito o primeiro método que apresentamos neste projeto, podemos verificar também que apesar de estarmos a trabalhar com *smartphones* e de estarmos limitados a capacidade do *hardware* que ele oferece tivemos muito bons resultados, este resultados devem-se à maneira como foi implementado o software e como foi construída a API por

nós proposta, com este método e com a nossa implementação o reconhecimento de objetos só é limitado pelo tamanho do ecrã e pela resolução do *smartphone*. Como descrito anteriormente, este software está preparado para reconhecer, face frontal, perfil da face, olhos, nariz, boca, parte superior do corpo humano, parte inferior do corpo humano, corpo humano completo e mãos, ou seja temos 10 elementos diferentes para reconhecimento, onde podemos ver que a quantidade de objetos que podemos reconhecer só é limitado pelos ficheiros classificativos que inserirmos no software. Este método foi testado com 25 ficheiros de classificação e a sua performance manteve-se o que podemos concluir que a quantidade de objetos que podemos reconhecer é muito satisfatória.

Capítulo 6

Conclusões

Com este estudo desenvolveu-se um sistema capaz de reconhecer vários objetos em tempo real com uma câmara de *smartphone*.

Este sistema de reconhecimento é composto por um conjunto de etapas apresentadas a seguir:

- Primeiro método:
    - Pré-processamento
    - Segmentação por K-means para separação do objeto de estudo
    - Deteção de contornos pelo operador Canny
    - Separação do objeto em estudo
    - Deteção de características
    - Guardar resultados referentes ao objeto
    - Reconhecimento
- Segundo método:
    - Recolha de imagens
    - Criar um Ficheiro *"Haar Cascade Classifier"*
    - Passar as características presentes no ficheiro para a memória
    - Criar um *array* para os objetos encontrados
    - Reconhecimento

Como podemos verificar ao longo deste projeto a etapa mais importante do mesmo é o segundo método, com ele conseguimos uma boa base de reconhecimento em tempo real, podemos também destacar os algoritmos Viola-Jones, e a construção de ficheiro de classificação *"Haar Cascade Classifier"* os passos mais importantes deste método, estes métodos são os mais utilizados no reconhecimento de imagem e não poderíamos prescindir deles. É de destacar também a forma como foi implementado este sistema de multi-reconhecimento em dispositivos móveis (*smartphones*), com as limitações que estes têm a nível de *hardware*, pouca memória e pouca capacidade de processamento em relação aos computadores tradicionais.

No primeiro método estudado não foi tão eficaz para o objetivo proposto, mas não é menos relevante, pois com ele conseguimos demostrar que os métodos de segmentação e análise de imagem são muito importantes para a separação de objetos, podemos destacar também que trabalhar sobre a matriz da imagem nos dá muitas vantagens neste tipo de projetos, com ele conseguimos verificar que algoritmos de filtragem de deteção de características ajudam-nos no reconhecimento.

Com este estudo ficamos a saber que para o reconhecimento de imagens é necessário uma grande componente de inteligência artificial, essa componente está bem presente no algoritmo de Viola-Jones e na criação dos ficheiros "*Haar Cascade Classifier*", que implementam algoritmos com árvores de decisão e redes neuronais para a separação de características e assim encontrar imagens positivas e negativas respetivamente.

Vantagens do método utilizado para o reconhecimento:

- Reconhecimento de qualquer objeto depois se ser treinando
- Reconhecimento do objeto independentemente do fundo onde ele se encontra
- Tempo de resposta de reconhecimento
- Adaptável a outros sistemas por ter sido desenvolvido em linguagem c/c++.

Desvantagens do método utilizado para o reconhecimento:

- Tempo de treino dos objetos
- Em objetos complexos requer mais do que um ficheiro de classificação ou então um número muito grande imagens positivas e negativas.

O sistema desenvolvido neste projeto foi testado com sucesso nos seguintes dipositivos móveis (*smartphones*):

- Nexus S – 1GHZ de processador e 512 de RAM, câmara de 5MP e Android 4.1 como sistema operativo:
- Sony Ericson – 1GHZ de processador e 512 de RAM, câmara de 8MP e Android 2.3.7 como sistema operativo.
- Huawei Idios x5 – 800MHZ de processador e 512RAM, câmara de 5MP e Android 4.0.3 como sistema operativo:
- T21 – 1GHZ de processador e 512RAM, câmara de 3MP e Android 4.0.3 como sistema operativo:
- HTC Desier – 1GHZ de processador e 512 de RAM, câmara de 5MP e Android 2.3.7 como sistema operativo:

Dificuldades Encontradas

Neste projeto uma das maiores dificuldades foi a criação dos ficheiros de classificação, esta etapa requer um computador dotado de grandes capacidades de processamento e de muita memória RAM, como o algoritmo de Viola-Jones corre muitas vezes as imagens para retirar as características das mesmas torna-se muito pesado em termos de processamento, lembramos aqui que para termos

uma boa classificação são necessárias uma média de 5000 imagens positivas e o dobro de imagens negativas. Por essa razão foram utilizados mais ficheiros fornecidos pela API OpenCV do que os ficheiros produzidos neste projeto, tendo em conta que o objetivo deste projeto era o reconhecimento de multiobjectos e não a produção de ficheiros classificativos a maior parte dos ficheiros utilizados foram os que a API fornecia.

Outra dificuldade encontrada foi a já mencionada em atrás, nas limitações do *hardware* do *smartphone*, quando tentávamos por na memória vários ficheiros de classificação no princípio tinha muitos erros de *MemoryOverFlow*, os quais foram resolvidos com a implementação *multi-thread* em código nativo sem passar pela máquina virtual do JAVA que faz esse controle automaticamente, o que não era satisfatório.

Trabalho Futuro

Para trabalho futuro estamos a pensar juntar os dois métodos estudados para fazer um reconhecimento mais eficaz, do primeiro método utilizamos a segmentação e os histogramas para separar as várias cores das imagens e depois com o segundo método construíamos um ficheiro de classificação com as propriedades vetoriais da imagem conseguindo assim uma forma mais eficaz de reconhecimento.

Pretende-se também implementar esta solução em imagens estáticas para termos um maior leque de utilização deste sistema, ou seja iria funcionar para imagens estáticas e imagens em tempo real.

E para diminuir o tempo de reconhecimento pretende-se melhorar a gestão de memória do *smartphone*, e assim aumentar o número de ficheiros de classificação inseridos no sistema. Foram realizados testes com 25 ficheiros e o sistema não perdeu performance, ou seja, não sabemos o número máximo de ficheiros que se podem utilizar, mas sabemos que é limitado. Por outro lado, pretende-se implementar um sistema que faça gestão do multiprocessamento que os *smartphones* hoje nos oferecem, é de salientar que hoje já temos dispositivos móveis com quatro processadores e com 2GB de memória RAM, o que para sistemas como este é o ideal.

Referências

[1] Russ, J. C., *The Image Processing Handbook - 2nd ed.*, CRC Press, 1995.

[2] Ogê Marques Filho, Hugo Vieira Neto, *Processamento Digital de Imagens, 1999*.

[3] OpenCV Reference Manual. V2.1. 2010

[4] Victor Eruhimov. OpenCV Tutorial. 2010

[5] Developer Android In: http://developer.android.com/index.html, acedido em 2 de Janeiro de 2012 até a data final do projeto.

[6] Java API Specifications In: http://www.oracle.com/technetwork/java/api-141528.html, acedido em 2 de Janeiro de 2012 até a data final do projeto.

[7] Sheng Liang. "The Java™ Native Interface" Programmer's Guide and Specification. 1999.

[8] Jurij Smakov. JNI Examples for Android. 2009.

[9] Google Goggles In: http://www.google.com/mobile/goggles/#text, acedido em 18 de Novembro de 2011.

[10] What is Layar? In: http://www.layar.com/what-is-layar/, acedido em 18 de Novembro de 2011.

[11] ARToolKit In: http://www.hitl.washington.edu/artoolkit/documentation/, acedido em 18 de Novembro de 2011.

[12] Junaio In: http://www.junaio.com/, acedido em 18 de Novembro de 2011.

[13] The Marvel ReEvolution Is Here In: http://marvel.com/news/story/18265/the_marvel_reevolution_is_here, acedido em 18 de Novembro de 2011.

[14] Maurício Marengoni, Denise Stringhini, Introdução à Visão Computacional usando OpenCV, 2009

[15] Rafael C. Gonzalez and Richard E. Woods. Digital Image Processing. Prentice Hall - Pearson Education International, second edition edition,2002.

[16] Roberto Hirata Jr. Segmentação de imagens por morfologia matemática, Março 1997.

[17] Jiri Matas and Jan Sochman. AdaBoost. 2009.

[18] Maria Dolores Valverde, Francesc J. Ferri. Experiments with Adaboost and linear programming. 2010.

[19] Ben Weber. Generic Object Detection using AdaBoost. 2008.

[20] Paul Viola, Michael Jones, Daniel Snow. Detecting Pedestrians Using Patterns of Motion and Appearance. 2003.

[21] Paul Viola and Michael Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. 2001.

[22] Paul Viola and Michael Jones. Robust Real-time Object Detection. 2001.

[23] Matthew L Weber. Android based Object Detection and Classification: Modeling a Child's learning of what's Hot/Cold. 2011.

[24] Google Play, Telêmetro - Smart Measure In: https://play.google.com/store/apps/details?id=kr.aboy.measure&feature=related_apps#?t=W251bGwsMSwxLDEwOSwia3luYWJveS5tZWFzdXJlll0, acedido em 12 de Julho de 2012

[25] Android boy's Lab In: http://androidboy1.blogspot.pt/, acedido em 12 de Julho de 2012

Referências Bibliográficas

Para este trabalho foram consultadas várias fontes bibliograficas, que apesar de não referenciadas no texto foram consultados os seguintes documentos que ajudaram na tomada de algumas decisões:

- Bradski, G. e Kaehler, A., Learning OpenCV, O'Reilly, 2008.
- Ricardo Sérgio Caetano Ferreira. Aplicações de processamento de imagem e sinal, Novembro 1999.
- Serkan Kiranyaz, Miguel Ferreira, and Moncef Gabbouj. Automatic object extraction over multiscale edge field for multimedia retrieval. IEEE TRANSACTIONS ON IMAGE PROCESSING, 15, DECEMBER 2006.
- Noyoon Kwak. Image processing and computer graphics, 1999.
- David MacKay. Information Theory, Inference and Learning Algorithms. 2003.
- Anuar L.H., Mashohor S., Mokhtar M. and Wan Adnan W.A.. Nose Tip Region Detection in 3D Facial Model across Large Pose Variation and Facial Expression. 2010.
- Nilcilene das Graças Medeiros, Erivaldo Antonio da Silva, José Roberto Nogueira. SEGMENTAÇÃO WATERSHED UTILIZANDO A TÉCNICA DO GRADIENTE MULTI-ESCALA E A TÉCNICA DE MÍNIMOS IMPOSTOS. 2010.
- Anand Panangadan Gaurav S. Sukhatme. Data Segmentation for Region Detection in a Sensor Network. 2005.
- Edinéia Aparecida dos Santos Galvanin, Erivaldo Antonio da Silva. EDGES DETECTION IN DIGITAL IMAGE BY A NON-LINEAR ANISOTROPIC DIFFUSION PROCESS. 2006.
- Douglas Messias Uba, Luciano Vieira Dutra. Deteção de objetos regulares em imagens de alta resolução utilizando casamento de modelos. 2009.
- A.Ashbrook and N.A.Thacker. "Tutorial" Algorithms For 2-Dimensional Object Recognition. 1998.

- Alvaro García Martín. people detection algorithms based on appearance and motion information. 2009.
- Vikram Goyal. "Pro Java ME MMAPI", Mobile Media API for Java Micro Edition. 2006.
- The OpenCV Tutorials, Release 2.3. 2011.
- Bernardo F. Reis, Jo˜ao Marcelo X. N. Teixeira, Veronica Teichrieb and Judith Kelner. Deteccão de Marcadores para Realidade Aumentada em FPGA. 2009.
- Radhakrishna Achanta, Francisco Estrada, Patricia Wils, and Sabine SÄusstrunk. Salient Region Detection and Segmentation. 2008.
- Alessandro Parolin. Segmentação de imagens de pessoas em tempo real para videoconferência. 2011.
- Karim Ali, Franc¸ois Fleuret, David Hasler, and Pascal Fua, Senior Member, IEEE. A Real-Time Deformable Detector. 2010.
- Stefan Varga, Michal Kostic. Android Sensors. 2010.
- Anush Mohan, Shyam Kumar. Object detection on the Android. 2010.
- Márcio Portes de Albuquerque e Marcelo Portes de Albuquerque. Processamento de Imagens: Métodos e Análises. 2000.
- Marlise Rosa. SEGMENTAÇÃO DE GRÃOS DE HEMATITA EM AMOSTRAS DE MINÉRIO DE FERRO POR ANÁLISE DE IMAGENS DE LUZ POLARIZADA. 2008.
- Adriano Breunig. ARQUITETURA FUZZY PARA RECONHECIMENTO, EM TEMPO-REAL, DE IMAGENS EM MOVIMENTO. 2005.
- Stephen Rose, Dale Potter, Matthew Newcombe. "Augmented Reality" A Review of available Augmented Reality packages and evaluation of their potential use in an educational context. 2010.
- Jérôme Berclaz, Franc¸ois Fleuret, Engin T¨ uretken, and Pascal Fua, Senior Member, IEEE. Multiple Object Tracking using K-Shortest Paths Optimization. 2010.
- Olga Barinova, Victor Lempitsky, Pushmeet Kohli. On Detection of Multiple Object Instances using Hough Transforms. 2010.
- Aldo von Wangenheim. Introdução à Visão Computacional. 1998.
- Michael Calonder, Vincent Lepetit, Mustafa Özuysal, Tomasz Trzcinski, Christoph Strecha, and Pascal Fua. BRIEF: Computing a local binary descriptor very fast. 2011.
- Dwayne Phillips. Image Processing in C. Second Edition. 1994.
- Alexander M. BRONSTEIN, Michael M. BRONSTEIN, Ron KIMMEL and Alon SPIRA. 3D Face Recognition without Facial Surface Reconstruction. 2003.
- David Riley. GPU Processing Methods for Machine Vision. 2007.
- Ming-Ming Cheng, Guo-Xin Zhang, Niloy J. Mitra, Xiaolei Huang, Shi-Min Hu. Global Contrast based Salient Region Detection. 2011.

- David Stavens. The OpenCV Library: Computing Optical Flow. 2011.
- Luís Domingues Tomé Jardim Tarrataca. A Gesture Recognition System using Smartphones. 2008.
- Duarte Manuel da Conceição Palma. Extracção Automática de Texto em Sequências de Vídeo. 2004.
- Rahul Choudhury. Recognizing pictures at an exhibition using SIFT. 2002.
- Murilo Fernandes Martins, Flavio Tonidandel e Reinaldo A.C. Bianchi. Reconhecimento de Objetos em Tempo Real para Futebol de Robôs. 2005.
- Andrew Davison. Java Prog. Techniques for Games. 2011.
- Phillip Ian Wilson, Dr. John Fernandez. FACIAL FEATURE DETECTION USING HAAR CLASSIFIERS. 2006.
- Gaurav Aggarwal, Amit K. Roy Chowdhury, Rama Chellappa. A System Identification Approach for Video-based Face Recognition. 2003.
- Yuko Roodt, Willem Visser, Willem A. Clarke, Member IEEE. Image Processing on the GPU: Implementing the Canny Edge Detection Algorithm. 2010.
- Isabela Santos Gonc¸alves, Antonio L. Apolin´ario Jr. Um Estudo Sobre o Uso de Classificadores Para o Reconhecimento Facial em Aplicações de Realidade Aumentada. 2010.
- Shyam Sunder Kumar, Min Sun, Silvio Savarese. Mobile Object Detection through Client-Server based Vote Transfer. 2011.
- Yi Yang, Sam Hallman, Deva Ramanan, Charless Fowlkes. Layered Object Detection for Multi-Class Segmentation. 2011.
- Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Expression-Invariant 3D Face Recognition. 2002
- Kevin Murphy, Antonio Torralba, Daniel Eaton, and William Freeman. Object detection and localization using local and global features. 2005.
- Maria João M. Vasconcelos e João Manuel R. S. Tavares. MÉTODOS DE SEGMENTAÇÃO DE IMAGEM PARA ANÁLISE DA MARCHA. 2006.
- Krystian Mikolajczyk, Bastian Leibe, Bernt Schiele. Multiple Object Class Detection with a Generative Model. 2006.
- Tatjana Zrimec, Andy Wyatt. Learning to Recognize Objects - Toward Automatic Calibration of Color Vision for Sony Robots. 2002
- Lenivaldo Ribeiro de Souza. ALGORITMO PARA RECONHECIMENTO E ACOMPANHAMENTO DE TRAJETÓRIA DE PADRÕES EM VÍDEOS. 2011.
- Jacinto Nascimento, Member IEEE, Jorge Marques, Performance evaluation of object detection algorithms for video surveillance. 2005
- Gorodnichy, D. Video-Based Framework for Face Recognition in Video. 2005.

- Liming Wang, Jianbo Shi, Gang Song, and I-fan Shen. Object Detection Combining Recognition and Segmentation. 2008.
- Ming-Hsuan Yang. Object Recognition. 2006.
- Yan Huang, Irfan Essa. Tracking Multiple Objects Through Occlusions. 2005.
- João Ascenso, João Valentim, Fernando Pereira. RECONHECIMENTO AUTOMÁTICO DE FACES USANDO INFORMAÇÃO DE TEXTURA E DE GEOMETRIA 3D.
- Vadim Pisarevsky, Senior Software Engineer Intel Corporation, Software and Solutions Group. OpenCV Object Detection: Theory and Practice.
- Florian Adolf. OpenCV's Rapid Object Detection. 2003
- Muhammad Zubair Malik. Rapid Object Detection using a Boosted Cascade of Simple Features (Viola and Jones CVPR 2001). 2007.
- The OpenCV User Guide. Release 2.4.2. 2012.
- Gary Bradski and Adrian Kaehler. Learning OpenCV. 2008.
- Jonathan E. Hunter. Object Recognition using Fuzzy Membership Rules. 2004.
- Frank Ableson. Reuse existing C code with the Android NDK. Learn how to use the Android Native Developer's Kit. 2011.
- Mustafa Özuysal, Michael Calonder, Vincent Lepetit and Pascal Fua. Fast Keypoint Recognition using Random Ferns. 2010.
- Mustafa Özuysal Pascal Fua Vincent Lepetit. Fast Keypoint Recognition in Ten Lines of Code. 2007.
- Mustafa Özuysal Vincent Lepetit Pascal Fua. Pose Estimation for Category Specific Multiview Object Localization. 2009.
- Christoph Strecha, Alexander M. Bronstein, Member, IEEE, Michael M. Bronstein Member, IEEE and Pascal Fua. LDAHash: Improved matching with smaller descriptors. 2011.
- V. Blanz, B. Scholkopf, H. Bulthoff, C. Burges, V. Vapnik, T. Vetter. Comparison of viewbased object recognition algorithms using realistic 3D models. 1996.
- Rafael Oliveira Lopes, Roberto Buaiz Simão. Implementação do algoritmo SIFT para deteção de objetos em imagens. 2011.
- Gemma Roig, Xavier Boix, Horesh Ben Shitrit, Pascal Fua. Conditional Random Fields for Multi-Camera Object Detection. 2011.
- Sergio Rodrigues Neves. ALGORITMOS PARA SEGMENTAÇÃO DE IMAGENS INFRAVERMELHAS. 2003.
- Eduard Serradell, Mustafa ¨Ozuysal, Vincent Lepetit, Pascal Fua, and Francesc Moreno-Noguer. Combining Geometric and Appearance Priors for Robust Homography Estimation. 2010.
- Chris Stauffer, W. Eric L. Grimson. Learning patterns of activity using real-time tracking. 2000.

- PAUL SUETENS, PASCAL FUA, ANDREW J. HANSON. Cornputatima! Strategies for Object Recognition. 1992.
- Tomasz J. Malisiewicz, Jonathan C. Huang, Alexei A. Efros. Detecting Objects via Multiple Segmentations and Latent Topic Models. 2007.
- Engin Tola, Christoph Strecha, Pascal Fua. Efficient Large Scale Multi-View Stereo for Ultra High Resolution Image Sets. 2011.
- Macintosh Version. Tutorial on Using OpenCV for Android Projects. 2012.
- Sen-Ching S. Cheung and Chandrika Kamath. Robust techniques for background subtraction in urban traffic video. 2007.
- Aydin Varol, Appu Shaji, Mathieu Salzmann, and Pascal Fua, Senior Member, IEEE. Monocular 3D Reconstruction of Locally Textured Surfaces. 2011.
- Jacinto Nascimento, Jorge S. Marques. New Performance Evaluation Metrics for Object Detection Algorithms. 2003.
- Daniel Weinland, Mustafa ¨Ozuysal, and Pascal Fua. Making Action Recognition Robust to Occlusions and Viewpoint Changes. 2010.
- Shaohua Zhou, Volker Krueger, and Rama Chellappa, Probabilistic recognition of human faces from video. 2003.
- Gary Bradski. Willow Garage, OpenCV, ROS, And Object Recognition. 2011.
- Luiz Nunes, Paulo Prado. RECONHECIMENTO DE OBJETOS CONTIDOS EM IMAGENS ATRAVÉS DE REDES NEURAIS. 2002.
- Maurício Marengoni, Denise Stringhini. Tutorial: Introdução à Visão Computacional usando OpenCV. 2011.
- Dongjae Lee, Wonchurl Jang, Doeksoo Park, Sung-Jae Kim. A Real-Time Image Selection Algorithm: Fingerprint Recognition using Mobile Devices with Embedded Camera. 2005.
- Bernd Girod, David Chen. Mobile Image Processing. 2003.
- K. S. Bae, K. K. Kim0, Y. G. Chung, W. P. Yu. Character Recognition System for Cellular Phone with Camera. 2005.
- Alexandre da Silva Simões. Segmentação de Imagens por Classificação de Cores: Uma Abordagem Neuronal. 2000.
- Fernando Castro. Reconhecimento e Localização de Padrões em Imagens Utilizando Redes Neuronais e Artificiais como Estimadores de Correlação Espectral. 1995.
- Takehito ABE, Tomonori TAKADA, Harumi KAWAMURA, Takayuki YASUNO and Noboru SONEHARA. Image-identification Methods for Camera-equipped Mobile Phones. 2007.
- Rafael Santos. Introdução ao Processamento de Imagens Digitais em Java / API JAI. 2010.
- Rafael Santos. Processamento de Imagens em Java.

- Guy Perelmuter, Enrique Vinicio Carrera E., Marley Vellasco e Marco Aurélio Pacheco. RECONHECIMENTO DE IMAGENS BIDIMENSIONAIS UTILIZANDO REDES NEURAIS ARTIFICIAIS. 1995.
- Alexandra Ribeiro e Miguel Figueiredo. Tutorial de OpenCV para Tótós. 2010.
- Sayed Hashimi, Satya Komatineni, Dave MacLean. Pro Android 2. 2010.
- W. FRANK ABLESON,CHARLIE COLLINS, ROBI SEN. Unlocking Android. 2009.
- W. FRANK ABLESON,CHARLIE COLLINS, ROBI SEN. Android in Action SECOND EDITION. 2011
- Antônio de Pádua Braga, André Ponce de Leon E de Carvalho, Teresa Bernarda Ludermir. Redes Neurais Artificiais: Teoria e Aplicações. 2000.
- Chris Simmonds. What else can you do with Android? Making use of Android. 2010.
- Open Source Computer Vision Library. Reference Manual. 2000.
- Amir hossein khalili. OpenCV Tutorial. 2007.
- Qing Chen. A Basic Introduction to OpenCV for Image Processing. 2007.
- Xuan Mo. OpenCV Tutorial I: Image Processing. 2011
- Umeshnarayanan. Introduction to OpenCv. 2010.
- Raúl Igual, Carlos Medrano. Tutorial de OpenCV. 2008
- Robert Laganière, VIVA lab, University of Ottawa. Programming computer vision applications: A step-by-step guide to the use of the Intel OpenCV library and the Microsoft DirectShow technology. 2008