Faculdade de Engenharia da Universidade do Porto

Interface baseada em objetos visuais usando dispositivos móveis
aplicação a serviço de cinema

Ivo Almeida Rodrigues

Dissertação

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Luís Filipe Pinto de Almeida Teixeira

19 de Fevereiro de 2014

Interface baseada em objetos visuais usando dispositivos móveis
aplicação a serviço de cinema

Ivo Almeida Rodrigues

Mestrado Integrado em Engenharia Informática e Computação

19 de Fevereiro de 2014

Resumo

Atualmente para saber algo sobre objetos que desconhecemos, podemos usar a Internet, mas sem o nome a tarefa torna-se difícil, ainda mais se for usado um diapositivo móvel como um "Smartphone" de pequenas dimensões, onde a ausência de teclado pode dificultar a entrada de dados. Com a crescente capacidade dos dispositivos móveis é possível aplicar o conceito de reconhecimento de objetos, que permite responder a estas necessidades.

Com uma imagem da câmara do dispositivo, consegue-se analisar e identificar os objetos relevantes. Desta forma o projeto que é proposto, tenta desenvolver um sistema capaz de reconhecer objetos ao usar apenas a câmara dos "Smartphones" ou outros dispositivos e uma ligação com um servidor, de forma a que este consiga identificar objetos na imagem e apresentar informação sobre estes. Como aplicação prática, um sistema capaz de reconhecer cartazes de cinema, vai ser desenvolvido em parceria com "Sapo Cinema", o sistema apresenta assim ao utilizador informação sobre o filme.

O sistema é baseado em ferramentas *open-source* de visão por computador, nomeadamente OpenCV. As características globais e locais das imagens capturadas são usadas para reconhecer os objetos. Este reconhecimento baseia-se na comparação de características em pontos de interesse.

Abstract

Currently to know something about an object that we do not know, we can use the Internet but without the name, the task becomes difficult, especially if we use a small Smartphone, where the lack of keyboard can hamper the input of data. With the growing capability of mobile devices it is possible to apply the concept of object recognition, which allows to meet these needs.

With a camera of a mobile device, it is possible to analyze and identify the relevant objects. Thus, this project attempts to develop a system capable of recognizing objects while using only a smartphone's camera and/or other devices and a connection to a server, so that it can identify objects in the image and display this information. As a practical application, a system capable of recognizing movie posters, will be developed in partnership with "Sapo Cinema", presenting the user with information about the identified movie.

The system is based on open-source tools for computer vision, namely OpenCV. The global and local features of the captured images are used to recognize objects. This recognition is based on comparing features in points of interest.

Agradecimentos

Ao Professor Luís Filipe Teixeira, pela sua orientação no decorrer deste semestre. A sua paciência, disponibilidade e grande capacidade de ensinar e motivar.

A todos os meus amigos, que me acompanharam este ano.

À minha família, que apesar dos difíceis sacrifícios, me ajudou e apoiou incessantemente.

Ao leitor, que desconhecido pela minha parte, espero que desfrute a leitura tanto como me deu prazer em escrevê-la.

Ivo Rodrigues

"Failure is success
if we learn from it."

George S. Patton

Conteúdo

1. Introdução **1**
- 1.1. Contexto/Enquadramento . . . 1
- 1.2. Objetivos . . . 2
- 1.3. Estrutura do Relatório . . . 2

2. Revisão Bibliográfica **3**
- 2.1. Técnicas de rastreio . . . 3
  - 2.1.1. Baseado em marcadores . . . 3
  - 2.1.2. Baseado em marcadores modelo/template . . . 3
  - 2.1.3. Código de barras . . . 4
  - 2.1.4. Marcadores Topológicos . . . 5
  - 2.1.5. Seguimento sem marcadores . . . 5
- 2.2. Comparação dos vários métodos de rastreio . . . 7
- 2.3. Algoritmos de deteção de pontos de interesse e extração de descritores . . . 8
- 2.4. Computação em nuvem . . . 9
- 2.5. Web services . . . 11
  - 2.5.1. SOAP . . . 11
  - 2.5.2. REST . . . 11
- 2.6. Ferramentas . . . 12
  - 2.6.1. OpenCV . . . 12
  - 2.6.2. Java . . . 12
  - 2.6.3. C/C++ . . . 13
  - 2.6.4. OpenMP . . . 13
  - 2.6.5. Ruby on Rails . . . 13
  - 2.6.6. SWIG - Simplified Wrapper and Interface Generator . . . 14
  - 2.6.7. Apache Cordova . . . 14
  - 2.6.8. Redis . . . 14
  - 2.6.9. Sidekiq . . . 15
  - 2.6.10. unixODBC . . . 15
  - 2.6.11. Resumo do uso das ferramentas/tecnologias . . . 16
- 2.7. Trabalhos Relacionados . . . 17
  - 2.7.1. Google Goggles . . . 17
  - 2.7.2. Augment - 3D Augmented Reality . . . 17
  - 2.7.3. Aplicação de reconhecimento de Livros em Coluna . . . 18
  - 2.7.4. Aplicação de reconhecimento de Ambientes exteriores . . . 19
  - 2.7.5. Layar . . . 19
  - 2.7.6. Word Lens Translator . . . 20
  - 2.7.7. Vivino Wine Scanner . . . 21
  - 2.7.8. Resumo de aplicações semelhantes . . . 22
- 2.8. Resumo . . . 23

3. Implementação **25**
3.1. Arquitetura do Sistema . . . 25
3.1.1. Componentes da API . . . 26
3.1.2. Mensagens da API (Interface Pública) . . . 27
3.1.3. Servidor . . . 30
3.1.4. Aplicação Cliente . . . 31
3.2. Resumo . . . 35

4. Testes e Resultados **37**
4.1. Especificações do hardware e Condições dos Testes . . . 37
4.2. Transferência da informação via Rede . . . 38
4.3. Formatos de armazenamento dos descritores . . . 39
4.4. Execução dos pedidos . . . 41
4.5. Resumo . . . 43

5. Conclusões e trabalho futuro **45**
5.1. Conclusões . . . 45
5.2. Trabalho futuro . . . 46

Referências **49**

A. Código JSON da API **51**

Lista de Figuras

1.1. Comparação de subscritores que usam "smartphones" e telemóveis convencionais. Fonte: Nielsen 1

2.1. Exemplo de marcador. 4
2.2. Exemplo de código de barras 1D. 4
2.3. Exemplo de código de barras 2D. 5
2.4. Exemplo de computação em nuvem. 10
2.5. Aplicação Google Goggles. Fonte: Google 17
2.6. Augment - 3D Augmented Reality. Fonte: augmentedev.com 18
2.7. Exemplo de reconhecimento de Livros em Coluna, Fonte: respetivo artigo [CSK+10]. . . 18
2.8. Exemplo de reconhecimento de ambientes exteriores, Fonte: respetivo artigo [TXG+08]. . 19
2.9. Aplicação Layar. Fonte: layar.com 20
2.10. Aplicação Word Lens Translator. Fonte: questvisual.com 20
2.11. Aplicação Vivino Wine. Fonte: Google Play (serviço de aplicações para Android) 21

3.1. Arquitetura da Solução Proposta 25
3.2. Esquema interno da API 26
3.3. Base de Dados da API 28
3.4. Algumas Mensagens da API 29
3.5. Esquema de uma possível configuração da rede interna 31
3.6. Aplicação cliente quando é aberta 32
3.7. Aplicação cliente quando é fotografado o objeto e enviado para o servidor 33
3.8. Aplicação cliente quando a pesquisa termina e suas funcionalidades 34
3.9. Aplicação cliente nas suas configurações 34

4.1. Imagens extraídas do dataset de fotografias tiradas com telemóvel 37

Lista de Tabelas

2.1. Comparação dos métodos de rastreio 7
2.2. Resumo do uso das ferramentas/tecnologias 16
2.3. Comparação de aplicações semelhantes 22

3.1. Exemplo de funções em libgor (C++) e suas equivalentes em libgor_wrap (Ruby) 26
3.2. Resumo de métodos do API Rest 30

4.1. Exemplo de resultados com diferentes configurações ao abrir a imagem no servidor . . . 39
4.2. Comparação da divisão da base de dados e formatos de armazenamento dos descritores. . 40
4.3. Tamanho da base de dados nos formatos XML, XML+ZLIB e PNG para 3570 cartazes. . . 41

Abreviaturas e Símbolos

API	*Application Programming Interface*
AR	*Augmented Reality*
BRIEF	*Binary Robust Independent Elementary Features*
BRISK	*Binary Robust Invariant Scalable Keypoints*
CPU	*Central Processing Unit*
CV	*Computer Vision*
FAST	*Features from Accelerated Segmrnt Test*
FREAK	*Fast Retina Keypoint*
FTP	*File Transfer Protocol*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
OCR	*Optical Character Recognition*
ORB	*Oriented FAST and Rotated BRIEF*
PNG	*Portable Network Graphics*
GFTT	*Good Features To Track*
GPS	*Global Positioning System*
REST	*Representational State Transfer*
SIFT	*Scale Invariant Feature Transform*
SOAP	*Simple Object Access Protocol*
SURF	*Speeded Up Robust Features*
URL	*Uniform Resource Locator*
WWW	*World Wide Web*
XML	*eXtensible Markup Language*
YAML	*YAML Ain't Markup Language*

1. Introdução

Neste capítulo é apresentado o contexto deste documento são explicados os seus objetivos. Conclui com uma descrição da estrutura deste documento.

1.1. Contexto/Enquadramento

Nos últimos anos os "smartphones" têm substituído os telemóveis convencionais, como mostrado na Figura 1.1, grande parte devido ao seu vasto leque de capacidades a custos acessíveis: GPS, "touchscreen", Internet móvel e Wi-Fi, câmara fotográfica e de vídeo, etc.

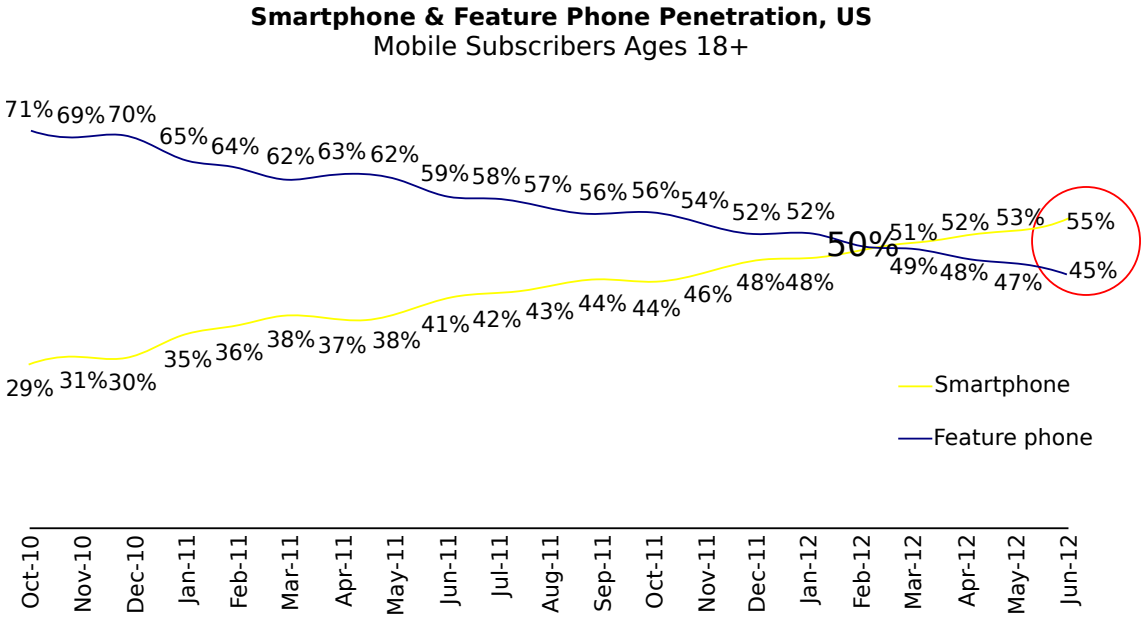

Figura 1.1.: Comparação de subscritores que usam "smartphones" e telemóveis convencionais. Fonte: Nielsen

Com as potencialidades atuais dos "smartphones", agora é possível explorar o mundo da visão por computador (CV) em dispositivos móveis. Desta forma podem ser criados sistemas de apoio ao utilizador com realidade aumentada (AR) e CV, que permitem

facilitar tarefas como por exemplo: dar informações sobre determinados objetos, ao sobrepondo-os a uma imagem da câmara temos uma perceção melhor de como um objeto ficaria se existisse na realidade, detetar pontos de interesse próximos com auxílio do GPS e da câmara, entre outros.

O problema é que sistemas baseados em CV exigem muito poder de processamento, o que limita as possibilidades. No entanto os "smartphones" são cada vez mais poderosos já que na atualidade existem "smartphones" com quatro núcleos de processamento, e os sistemas de rede nestes dispositivos são cada vez mais rápidos, o que permite o envio da informação para sistemas mais potentes e estes apenas retornam os resultados do processamento. Desta forma é possível eliminar as restrições anteriores.

Na atualidade com os fatores referidos e com aplicações de AR a aparecerem em estado embrionário, o mercado encontra-se muito favorável para o tipo de aplicação que nos propomos a desenvolver.

1.2. Objetivos

Pretende-se com esta dissertação continuar o trabalho desenvolvido pelo Hugo Teixeira [Tei13]. Neste sentido após um estudo aprofundado das tecnologias existentes, o objetivo no final será criar uma aplicação Android, um Servidor e uma API, capaz de reconhecer objetos genéricos.

A aplicação Android deverá ser capaz de reconhecer o cartaz de cinema presente na imagem obtida pela câmara do "smartphone" e apresentar ao utilizador a informação do filme indicado no cartaz. A fonte de dados dos cartazes existentes é obtido a partir do SAPO Cinema, um serviço online dedicado ao cinema.

O sistema tem de funcionar em tempo real. Desta forma, é prioritário que os algoritmos de "matching", de deteção e de extração de descritores tenham o melhor rácio performance e tempo de execução, que permita o seu uso em tempo real.

1.3. Estrutura do Relatório

Além da introdução, este documento contém mais 4 capítulos.

No Capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados.

No Capítulo 3, são apresentados possíveis implementações dos vários subsistemas, desde a aplicação Android à API e servidor.

No Capítulo 4, são realizados testes ao sistema e apresentados os resultados.

No Capítulo 5, as principais conclusões são referidas, serve assim como um resumo de todo o documento.

2. Revisão Bibliográfica

2.1. Técnicas de rastreio

Existem várias técnicas de rastreio, desde baseadas em marcadores a uso intensivo de algoritmos de CV. Os dispositivos móveis modernos "smartphones", têm hoje em dia muitos subsistemas, entre os quais: câmara de filmar/fotografar, flash, GPS, giroscópios, wireless, sensores de luz, entre outros. Isto permite a estes dispositivos obter uma informação do ambiente que os rodeia, de forma a reconhecer objetos.

O objetivo desta secção é rever vários métodos usados para rastreio/tracking em imagens, que possibilita entender a escolha de um método de reconhecimento de objetos sem marcadores conforme descrito na tese de Hugo Teixeira [Tei13].

2.1.1. Baseado em marcadores

É seguramente o método mais usado para rastrear objetos. Um marcador consiste num padrão conhecido de forma a que um algoritmo CV consiga reconhecer o objeto facilmente. Isto permite um custo muito baixo de complexidade computacional e tempo, e elevada taxa de sucesso.

2.1.2. Baseado em marcadores modelo/template

Um marcador consiste tipicamente num quadrado preto e branco com um padrão conhecido no interior. Desta forma é simples efetuar o reconhecimento, mas normalmente são usados vários padrões possíveis, o que torna o algoritmo CV mais complexo e mais dispendioso. Quanto mais elevado for o número de padrões, maior a complexidade e consequentemente o tempo despendido. Outro problema, é que a aplicação necessita de uma fase de treino para reconhecer os marcadores eficazmente.

A Figura 2.1 representa um exemplo de um marcador.

Figura 2.1.: Exemplo de marcador.

2.1.3. Código de barras

Um código de barras é geralmente preto e branco, tem várias disposições e formas dependendo do formatos. Existem vários de códigos de barras, e estes podem ter várias dimensões 1D ou 2D.

Desta forma o utilizador não necessita de indicar imagens de padrões nem treinar a aplicação, já que o próprio padrão tem o conteúdo necessário. Isto permite o uso de um algoritmo mais eficiente e capaz para qualquer tarefa que o uso de marcadores não seja problemático.

Dentro dos códigos 1D existem vários formatos como: Codabar, Code 25 – (Non) interleaved 2 of 5, EAN 2, EAN-8, EAN-13, etc. A Figura 2.2 representa um exemplo de um código de barras de uma dimensão no formato EAN-8.

Figura 2.2.: Exemplo de código de barras 1D.

Dentro dos códigos 2D existem vários formatos como: Aztec Code, Data Matrix, MaxiCode, ShotCode, QR Code, etc. O código QR tem várias versões 1-40, cada uma com maior capacidade; na versão 40 consegue armazenar até 23Kb de informação. A Figura 2.3 representa um exemplo de um código de barras de duas dimensões no formato QR. Este formato encontra-se atualmente em forte expansão e é usado em qualquer produto para qualquer tarefa desde controlo de qualidade a direcionar um utilizador para um site da empresa de forma fácil e eficaz. O utilizador tem apenas que apontar o "smartphone", para o código QR com uma aplicação de leitura de QR aberta e a aplicação redireciona automaticamente o utilizador para o site pretendido. Neste caso o código representa um URL para o site da FEUP (http://fe.up.pt).

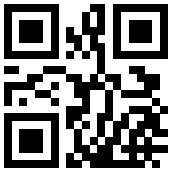

Figura 2.3.: Exemplo de código de barras 2D.

2.1.4. Marcadores Topológicos

Os marcadores topológicos são baseados na tonalidade das cores, tons claros e escuros. Não dependem assim da forma e possibilitam criar um marcador com significado para o utilizador, isto é, podem representar uma imagem, um sinal, etc. Como é necessária a criação destes marcadores, a aplicação tem de analisar e treinar. E assim, necessária uma fase de treino, para conseguir reconhecer estes marcadores. Por esse motivo isto têm um tempo e custo computacional elevado.

2.1.5. Seguimento sem marcadores

Tem a vantagem de não ser necessário qualquer tipo de marcador. Baseia-se apenas nos algoritmos CV para reconhecer o objeto alvo, permitindo que seja usado de forma mais genérica. A principal desvantagem, é que se torna mais complexo e consequentemente mais dispendioso computacionalmente. Desta forma as aplicações que usam esta técnica, são normalmente baseadas em computação em nuvem, o que permite aumentar o desempenho dos resultados e da aplicação.

Esta técnica consiste em três passos:

1. Deteção de pontos de interesse - Também chamado “keypoint detection”, consiste e detetar caraterísticas na imagem como por exemplo cantos. Estes pontos pode ser descritos como um vetor denso de dimensão fixa.

2. Extração de descritores - Os pontos de interesse são agrupados numa matriz, em que cada linha representa um ponto de interesse.

3. Comparação dos descritores - Conhecido como “descriptor matching”, tenta aplicar vários algoritmos como rotação e escala a um conjunto de descritores de forma a que consiga o numero máximo de “hits” com outro conjunto de descritores. Basicamente com os descritores de uma imagem, é possível comparar com os descritores doutra imagem e verificar quantos “hist” se obteve, quanto maior o numero de “hits”, maior a probabilidade de uma imagem estar contida na outra.

Existem vários algoritmos que podem ser usados para as diferentes fases, os algoritmos selecionados estão descritos na Secção 2.6.1. O primeiro passo é analisar cada pixel da imagem de forma a verificar se é um ponto de interesse (keypoint), por exemplo um canto ou borda; Depois é criada uma representação de cada ponto de interesse; Por fim realiza-se uma comparação dos descritores da imagem analisada com os descritores na base de dados (treinada por um conjunto de imagens de treino), numa tentativa de determinar se o objeto corresponde a alguma existente na base de dados. O sistema tenta usar diferentes perspetivas e ângulos para determinar as semelhanças das imagens. Diferentes resultados são obtidos com diferentes algoritmos quer na deteção quer na extração dos descritores.

2.2. Comparação dos vários métodos de rastreio

Os códigos de barras são sistemas maduros, normalmente usados em produtos ou em processos de automação nas fábricas. No contexto da realidade aumentada contêm a informação do objeto, assim não necessita de treinar a aplicação para o uso nestes sistemas. Normalmente não têm significado visual para o utilizador, isto é, o utilizador não consegue reconhecer facilmente o padrão. No entanto o uso de códigos de barra 2D como "QR code", que têm um sistema de correção de erros permite que estes sejam modificados de forma a torna-los mais apelativos e com significado, à custa do sistema de correção de erros.

Sistemas baseados em marcadores modelo, podem ser bastante eficazes desde que o numero de modelos seja reduzido, a razão é que quanto maior número de marcadores, maior a complexidade. Este tipo de marcadores consegue representar algum significado para o utilizador mas encontra-se muito limitado na forma e simplicidade.

Os marcadores topológicos conseguem ser identificados pelo utilizador. São apenas usadas as tonalidades do padrão para representar o marcador, no entanto, isto implica uma câmara estável e necessita de uma fase de treino para que a aplicação consiga reconhecer os padrões. A complexidade aumenta de acordo com o número de padrões necessários, e é mais custoso que os marcadores baseados em modelos.

O rastreio sem qualquer tipo de marcadores é o mais complexo, mas permite que o ambiente não seja "poluído" com estes marcadores. Este sistema é o ideal para a aplicação proposta, isto porque podem existir milhares de objetos na base de dados e estes não têm necessariamente código de barras ou marcadores.

Um resumo comparativo dos diferentes métodos é indicado na Tabela 2.1.

Tabela 2.1.: Comparação dos métodos de rastreio

	Modelo/ Template	**Código de barras 1D**	**Código de barras 2D**	**Topológicos**	**Sem Marcadores**
Complexidade	Média	Baixa	Baixa	Média/Alta	Alta
Auto identificativo	Não	Sim	Sim	Não	Não
Suscetível a sujidade	Médio	Médio	Baixo	Alto	Médio
Intrusivo	Sim	Sim	Sim	Sim	Não

2.3. Algoritmos de deteção de pontos de interesse e extração de descritores

Vários algoritmos de deteção de pontos de interesse e extração de descritores existem, a biblioteca OpenCV providencia muitos destes algoritmos entre os quais:

- GFTT [ST94] *Good Features To Track*, tenta detetar cantos e linhas fortes, ao escolher características fortes nas imagens, estas conseguem ser identificadas mais facilmente, uma vez que são menos afetadas pela rotação ou a escala da imagem.
- HARRIS [HS88] GFTT com detector Harris activado, "Harris" é um algoritmo que permite a deteção de cantos e tolera rotações mas não escala da imagem.
- SIFT [Low99] *Scale Invariant Feature Transform*, como do algoritmo de "Harris", a deteção de cantos é tolerável a rotações, mas SIFT traz a vantagem de se também ser tolerável à escala. Não pode ser usado para fins comerciais sem autorização do detentor da patente.
- SURF [BTG06] *Speeded Up Robust Features*, foi primeiramente apresentado por Herbert Bay et al. em 2006, pode ser usado em reconhecimento de objetos, bem como reconstrução 3D. É parcialmente inspirado no SIFT. É várias vezes mais rápido que o SIFT e é baseado em somas de respostas "2D Haar wavelet" e faz um uso eficiente de "integral images". Não pode ser usado para fins comerciais sem autorização do detentor da patente.
- BRISK [LCS11] *Binary Robust Invariant Scalable Keypoints*, Tenta apresentar uma alternativa ao SIFT e SURF, mantendo a robustez e alta velocidade. A chave para tal velocidade está na aplicação de um detetor baseado em FAST escalável em combinação com um descritor "binary string" a partir de comparações obtidas por amostras a cada ponto de interesse vizinho.
- FAST [RD05] *Features from Accelerated Segment Test*, extremamente rápido é um algoritmo de deteção de cantos, ideal para processamento em tempo real. No entanto deteta demasiadas características, ou seja por não ser seletivo (características fortes), é muito provável que a característica selecionada não seja ótimas e se encontrem características adjacentes umas das outras.
- BRIEF [CLSF10] *Binary Robust Independent Elementary Features*, não é usado como método de pesquisa de pontos de interesse, mas sim um método que permite reduzir as dimensões dos pontos de interesse, convertendo-as em "binary strings", sem necessidade de obter primeiro os descritores. Por exemplo SIFT usa um vetor com uma dimensão de 128, todas estas dimensões podem não ser necessárias para

o "matching" dos descritores, desta forma estes pontos podem ser convertidos para "binary strings", que são muito eficientes de efetuar o "matching" com a distancia de "Hamming", que basicamente usa instruções XOR e "bit count" dos processador, estas instruções são extremamente rápidas em processadores equipados com instruções SSE. Desta forma BRIEF é um algoritmo que consome pouca memória e é mais eficiente de que usar descritores com elevadas dimensões.

- ORB [RRKB11] *Oriented FAST and Rotated BRIEF*, é considerado uma alternativa eficiente ao SIFT e SURF, considerando que não requer patente. Consiste numa fusão do algoritmo FAST com descritores BRIEF, com algumas modificações que permite obter a orientação que o FAST não obtém.

- STAR [AKB08] Derivado no CenSurE (Center Surrounded Extrema), é uma implementação no OpenCV trazida por K. Konolige, mas usa polígonos como quadrados, pentágono e hexágonos, como alternativas computacionalmente menos custosas a círculos, sobrepõem dois quadrados para realizar estas aproximações.

- FREAK [AOV12] *Fast Retina Keypoint*, tenta ser mais eficiente em sistemas com pouca memoria, mas apresenta resultados competitivos com os seus concorrentes (incluso comerciais SURF), baseia-se no sistema visual humano nomeadamente a retina, para atingir os seus objetivos de rápido processamento e uso de pouca memória.

Dos vários algoritmos de extração de descritores, optou-se pelo algoritmo FREAK. Este algoritmo apresenta um bom rácio de performance e tempo necessário [Tei13]. Para deteção dos pontos de interesse, foi adotado o algoritmo ORB (baseado no FAST e no BRIEF), já que o uso de SURF encontra-se limitado para uso não comercial, e é uma boa alternativa [RRKB11]. Uma vez que se pretende com este projeto uma ferramenta 100% livre, SURF ou SIFT não são opções.

2.4. Computação em nuvem

Computação em nuvem (em inglês, *cloud computing*) refere-se à partilha de recursos computacionais, memória, capacidade de processamento, armazenamento, etc, numa rede (neste caso Internet), exemplo disso é a Figura 2.4. No contexto desta aplicação, o servidor central pode ser interpretado como uma única máquina que tem capacidades computacionais muito mais elevados que os "smartphone", dos quais, recebe pedidos de processamento, isto é, vários utilizadores podem ligar-se ao servidor central e realizar pedidos de informação, após os quais o servidor responde. O servidor internamente pode ou não ser composto por vários sistemas que partilham os seus recursos, mas para a aplicação apenas uma máquina é visível e responde aos seus pedidos.

Para além da grande capacidade de processamento, esta técnica oferece muitas vantagens como:

- É independente do sistema operativo do servidor
- Atualizações feitas aos algoritmos usados no servidor ficam imediatamente aplicados aos resultados fornecidos à aplicação
- A informação pode ser acedida em qualquer lugar, desde que exista ligação à rede.

Contudo existem desvantagens no uso desta tecnologia:

- Necessita de uma manutenção mais cuidada do servidor (ou servidores) para garantir a disponibilidade do mesmo.
- A aplicação fica totalmente dependente do servidor, o que implica que se o servidor não se encontra ativo devido a um problema técnico, todos os utilizadores são afetados (técnicas como "load-balancing" podem ser usadas para combater este problema, mas requerem servidores extras).

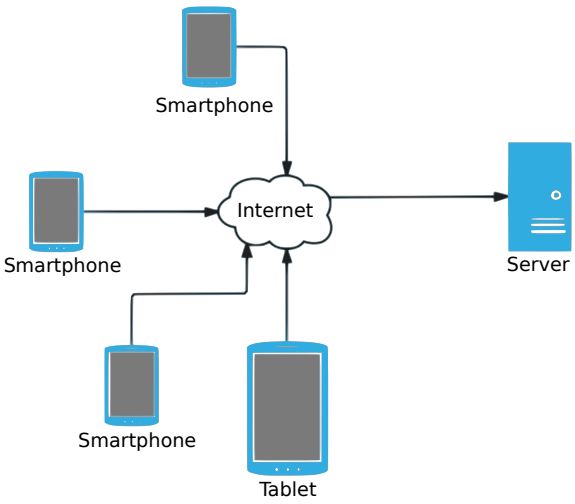

Figura 2.4.: Exemplo de computação em nuvem.

2.5. Web services

2.5.1. SOAP

SOAP (Simple Object Access Protocol), é um protocolo que permite a troca de informação estruturada em XML. Geralmente baseiam-se em HTTP para transferência de dados, no entanto pode ser implementado em praticamente qualquer protocolo, como por exemplo FTP.

É um protocolo complexo, uma vez que nas suas mensagens encontram-se estabelecidos os dados, as regras de como processar os dados e uma definição dos mesmos. Consome mais recursos que alternativas como o REST, mas é um protocolo maduro e seguro [zMNS05].

Possui vários mecanismos como:

- Expansão
- Troca de mensagens
- Tratamento de erros
- Representação de dados

2.5.2. REST

REST (Representational State Transfer), é uma técnica, não um protocolo como o SOAP, que permite a criação de serviços orientados para a WEB, usa o protocolo HTML para transferência de mensagens e a informação pode ser estruturada em vários formatos, como por exemplo JSON, XML ou YAML.

Esta técnica usa métodos existentes no protocolo HTML desde os seus primórdios como PUT, GET, POST, DELETE e PATCH, embora normalmente no protocolo HTTP são usados apenas POST e GET na maioria das vezes. Desta forma o REST propõe o uso destes métodos para, sem necessidade de "reinventar a roda", resolver o problema dos recursos distribuídos.

Segundo o artigo [zMNS05], em dispositivos móveis, este método é preferível ao SOAP pois permite não só economizar tráfego da rede, como consome menos recursos e é mais eficiente. Por essas razões vai ser adotado este método no desenvolvimento da API.

2.6. Ferramentas

O sistema vai ser desenvolvido com ferramentas *open-source*, e com várias linguagens de programação, no entanto vai ser considerada a performance do sistema. Pelo que C/C++ vai ser usado para partes críticas do sistema.

2.6.1. OpenCV

O OpenCV (Open Source Computer Vision Library) é uma biblioteca *open-source* que permite um acesso fácil a algoritmos de CV. Encontra-se disponível para várias plataformas, incluindo Android. Tem já incorporados os algoritmos necessários para o desenvolvimento desta aplicação: rastreio sem marcadores.

A versão do OpenCV usada foi a versão 2.4.5, que tem melhorias de performance relativo à versão 2.4.3 anteriormente adotada [Tei13].

O *OpenCV Manager*, é uma aplicação Android que é necessária para que seja possível o uso da biblioteca num dispositivo Android. Funciona como um "wrap" da biblioteca (nativa), e permite que esta seja partilhada por outras aplicações que usem OpenCV, possibilita dessa forma o acesso a partir de da linguagem de programação Java. Mesmo que só exista uma aplicação, a atualização da biblioteca não afeta negativamente as aplicações que a usarem. Pelo contrário, melhorias de performance são automaticamente aplicadas sem necessidade de atualizar as aplicações. Outra vantagem do uso do *OpenCV Manager* é que este é independente da plataforma, o que permite a qualquer dispositivo que tenha Android executar a aplicação sem que esta tenha de ser recompilada para o hardware específico.

2.6.2. Java

O Java é uma linguagem de programação, padrão no sistema Android, que permite compilar o código uma vez e executar em todas as plataformas suportadas. Isto é possível devido ao facto de o código ser compilado para *Bytecode*, um código de baixo nível mas não código máquina nativo. O *Bytecode* é então interpretado pela *Java Virtual Machine (JVM)*. O *Bytecode* é um código muito semelhante ao código nativo, o que significa que a performance é consideravelmente maior a linguagens puramente interpretadas. No entanto ainda existe perda de performance mas permite executar em todas as plataformas, já que existem inúmeros dispositivos e inúmeros plataformas não seria possível criar uma aplicação que executasse em todas as plataformas de forma eficaz.

2.6.3. C/C++

C++ é uma linguagem de uso geral que permite o uso de diferentes paradigmas, com um grande desempenho e é muito popular entre as linguagens comerciais. É uma linguagem de médio/alto nível e combina assim características de linguagens de alto e baixo níveis.

A linguagem para partes críticas é desenvolvida em C/C++, isto permite um aumento de performance relativo ao uso de Java, ou linguagens interpretadas como Ruby, Python ou PHP, no entanto impossibilita a escrita para todas as plataformas suportadas por Android, por isso apenas a parte do servidor que suporte CV, é implementada em C/C++.

2.6.4. OpenMP

OpenMP (Open Multi-Processing), é uma API que estende linguagens existentes como C/C++ e FORTRAN com diretivas:

```
#pragma omp [directive]
```

O objetivo é possibilitar uma forma fácil e rápida de usar o modelo de programação paralela com memória partilhada entre vários processadores[Sat02].

Esta API é uma norma da indústria atualmente muito usado em aplicações de alta performance e com elevadas exigências computacionais.

No projeto, é usado o OpenMP para escalar o servidor para sistemas com múltiplos núcleos de processamento, o que vai permitir aproveitar todo o poder de processamento da máquina, aumentando assim a performance do sistema, embora o OpenCV já suporta o uso de várias "threads" o que pode gerar concorrência entre elas. Ativar o OpenMP é uma opção configurável na compilação quer do OpenCV quer da API do servidor, desta forma o utilizador pode testar qual das opções fornece maior performance.

2.6.5. Ruby on Rails

O "Ruby on Rails" é uma tecnologia que combina a *framework* Rails, com a linguagem Ruby. Esta *framework* baseada em *MVC (Model-View-Controller)* é livre e tem como principal objetivo aumentar a facilidade e a produtividade no desenvolvimento de sites orientados a base de dados e serviços (REST).

"Ruby on Rails é um avanço na redução das barreiras na inicialização à programação. Aplicações web poderosas que anteriormente poderiam ter levado semanas ou meses para desenvolver podem ser produzida numa questão de dias." - Tim O'Reilly, Fundador do O'Reilly Media.

Esta tecnologia vai ser usada para o servidor, o Rails integra-se bem com o C++, o que facilita o uso desta tecnologia integrada com os módulos CV. A parte desenvolvida em Rails vai concentrar-se na API REST do sistema.

2.6.6. SWIG - Simplified Wrapper and Interface Generator

SWIG (*Simplified Wrapper and Interface Generator*), permite a uma variedade linguagens de programação de alto nível, conectar com programas escritos em C/C++. Suporta linguagens de "script" como Perl, PHP, Python e Ruby, assim como linguagens compiladas (para código nativo ou para uma máquina virtual), como C#, D, Java, Lua, Octave e R. SWIG é mais usado para gerar o código necessário, para que a linguagem alvo consiga realizar chamadas a código C/C++. No entanto pode gerar "parse tree" no formato XML e expressões Lisp. SWIG é um software livre e o código gerado é compatível com projetos comerciais e não comerciais.

Este software é usado para permitir o Ruby interagir com a parte crítica do sistema.

2.6.7. Apache Cordova

Apache Cordova é uma plataforma para desenvolver aplicações nativas com o uso de HTML, CSS e JavaScript. Constitui um conjunto de APIs que permite um programador aceder a funções nativas como a câmara ou o acelerómetro a a partir de JavaScript.

É usado na aplicação de reconhecimento de cartazes.

2.6.8. Redis

Redis é um software "open source", de armazenamento avançado do tipo "key-value". É frequentemente referido como um servidor de estrutura de dados, já que as chaves ("keys") podem conter strings, hashes, listas, conjuntos e conjuntos ordenados.

Escrito em ANSI C, funciona na maioria dos sistemas POSIX, como Linux, BSD e OS X sem dependências externas. Não existe suporte para Windows, no entanto existe um versão experimental do Redis.

É usado como dependência para a ferramenta Sidekiq 2.6.9.

2.6.9. Sidekiq

Sidekiq é um software de código fonte aberto, e é um simples mas eficiente processador de tarefas em "background" para Ruby.

Usa threads para processar várias tarefas em simultâneo no mesmo processo. Não requer Rails mas encontra-se fortemente integrado no Rails 3/4 para facilitar o seu uso.

É compatível com outro software usado normalmente para as mesmas funções, por exemplo o "Resque" usa o mesmo formato de mensagens, sendo assim possível integrar num "Resque farm", já existente.

O Sidekiq é usado para processar as tarefas CV em "background" e seguir a evolução das mesmas. Desta forma o cliente pode verificar o estado do seu pedido de forma assíncrona.

2.6.10. unixODBC

ODBC é uma especificação aberta, que proporciona a programadores uma API previsível para aceder a fontes de dados como por exemplo uma base de dados SQL, que possuem drivers ODBC. Esta aplicação "unixODBC", pretende ser um standard definitivo para plataformas não Windows.

Tem a vantagem do código fonte tornar-se mais portável, assim como não limitar a aplicação a uma única base de dados. É usada pela API em C/C++ para aceder, adicionar e remover os descritores de treino, assim como adicionar os resultados de um pedido à base de dados.

2.6.11. Resumo do uso das ferramentas/tecnologias

As várias ferramentas/tecnologias referidas anteriormente, são usadas para diferentes fins no sistema. Para ajudar a entender onde são usadas, a Tabela 2.2 compila estas informações.

Tabela 2.2.: Resumo do uso das ferramentas/tecnologias

		Servidor	
	Aplicação cliente	**API CV**	**API Rest**
OpenCV	X	X	
Java	X		
C/C++		X	
OpenMP		X	
Ruby on Rails			X
SWIG			X
Apache Cordova	X		
Redis			X
Sidekiq			X
unixODBC		X	

2.7. Trabalhos Relacionados

2.7.1. Google Goggles

O "Google Goggles" permite pesquisar por fotografias tiradas pela câmara do "smartphone". O utilizador tira uma fotografia do objeto pretendido, por exemplo um quadro, um código de barras, um produto ou monumento e se o Goggles encontrar a imagem na base de dados é apresentada informação útil. Funciona também como tradutor já que consegue "ler" várias línguas e traduz de um idioma para outro.

A Figura 2.5 apresenta a aplicação a ser executada em vários sistemas móveis.

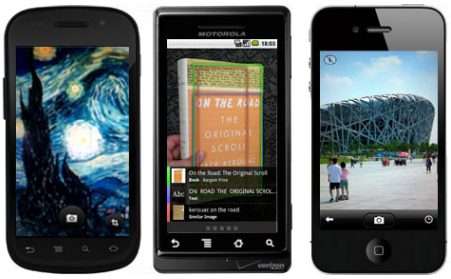

Figura 2.5.: Aplicação Google Goggles. Fonte: Google

2.7.2. Augment - 3D Augmented Reality

O Augment é um exemplo de aplicação prática da sobreposição de objetos à imagem da câmara a partir de marcadores. Com este sistema é possível visualizar modelos 3D como se existissem na realidade em tamanho real. Permite assim que um designer mostre como o produto ficaria na realidade antes de este entrar em produção, ou uma companhia de vendas de mobiliário mostre como o sofá fica bem na sala do cliente antes deste comprar o produto. O Augment funciona em "smartphone" e tablet baseados em iOS e Android.

O Augment é um sistema baseado em marcadores, não é esse o objetivo do sistema a desenvolver.

A Figura 2.6 apresenta a aplicação a ser executada num tablet Android.

Figura 2.6.: Augment - 3D Augmented Reality. Fonte: augmentedev.com

2.7.3. Aplicação de reconhecimento de Livros em Coluna

A aplicação de Chen et al. [CSK+10], permite reconhecer livros expostos em coluna, o artigo citado aborda semelhanças com o objetivo proposto, como:

- A utilização de uma grande base de dados
- Aplicação realizada em Android
- Uso de métodos para reduzir o tempo de pesquisa, ex: utilização de "cloud-computing"
- Apresenta a informação dos objetos reconhecidos

A principal diferença, é que a aplicação a ser desenvolvida vai conter também uma API, que permite o reconhecimento de objetos gerais, não apenas de cartazes de cinema, desta forma o sistema vai ser mais escalável e de fácil implementação para utilização noutras aplicações. A aplicação usa SURF, dessa forma não é possível usar comercialmente se autorização dos detentores da patente. Outros métodos de pesquisa e comunicação com o servidor foram também testados.

A Figura 2.7 apresenta a aplicação a ser executada em um "smartphone" Android.

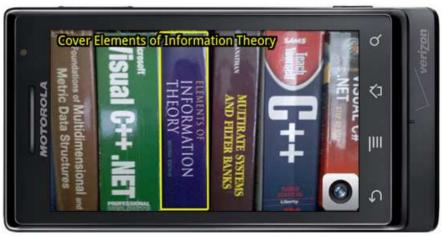

Figura 2.7.: Exemplo de reconhecimento de Livros em Coluna, Fonte: respetivo artigo [CSK+10].

2.7.4. Aplicação de reconhecimento de Ambientes exteriores

A aplicação de Takacs et al. [TXG+08], reconhece ambientes exteriores de forma eficaz devido a este usar o GPS embebido no dispositivo móvel. Isto permite que todo o processamento seja efetuado no dispositivo móvel, eliminando assim a necessidade de um servidor dedicado a processamento de uma base de dados massiva.

Para o reconhecimento é usada uma versão modificada do algoritmo SURF de modo a melhorar a performance, o que torna a algoritmo mais adequado aos dispositivos móveis.

A desvantagem desta aplicação é que não é gratuita, o que torna o uso do mesmo inadequado ao sistema desenvolvido.

A Figura 2.8 apresenta a aplicação a ser executada em um "smartphone" Android.

Figura 2.8.: Exemplo de reconhecimento de ambientes exteriores, Fonte: respetivo artigo [TXG+08].

2.7.5. Layar

A aplicação Layar, tem uma forma bastante original de "pensar" em revistas, código QR, e localizações geográficas.

A aplicação centra-se em conteúdo dinâmico, por exemplo revista suportadas conseguem ter vídeos nas suas páginas o que atrai mais o público. É possível também usar código QR para tocar música, carregar modelos 3D ou ver vídeos.

O objetivo das aplicações são diferentes, mas o reconhecimento de páginas de revistas, não se baseia em marcadores como códigos QR. Isto implica que as páginas da revista (em vídeo obtido pala câmara) são processadas em tempo real.

Esta aplicação é gratuita de usar mas é fechada em código, e só as revistas suportadas têm a possibilidade de usufruir do serviço.

A Figura 2.9 apresenta a aplicação a ser executada num "smartphone"; a aplicação encontra-se disponível em vários sistemas.

Figura 2.9.: Aplicação Layar. Fonte: layar.com

2.7.6. Word Lens Translator

"Word Lens Translator", é uma aplicação (da Quest Visual) muito útil que traduz instantaneamente o texto capturado pela câmara de um dispositivo de um idioma para outro.

Embora a tradução de textos, não seja o objetivo da aplicação a ser desenvolvida, o facto desta aplicação usar OCR, para o reconhecimento de caracteres na imagem, como um extraordinário exemplo, serve como referência à performance da aplicação. Desta forma o uso de OCR poderá ser considerado como trabalho futuro, como um elemento extra. No entanto o texto em cartazes podem ser muito estilizados o que torna inviável o uso de OCR como parâmetro principal de reconhecimento.

A Figura 2.10 apresenta a aplicação a ser executada num "smartphone"; a aplicação encontra-se disponível em vários sistemas.

Figura 2.10.: Aplicação Word Lens Translator. Fonte: questvisual.com

2.7.7. Vivino Wine Scanner

A aplicação "Vivino Wine", tem como objetivo ajudar a recordar os vinhos favoritos, aprender mais sobre os vinhos, e a escolher o vinho conforme as preferências. Ao tirar uma fotografia a um vinho, esta é automaticamente reconhecida a partir de uma base de dados com mais de dois milhões de vinhos. No entanto, no caso do vinho não ser encontrado, uma equipa tenta reconhecer o vinho por nós.

Esta aplicação é interessante pois apresenta pontos comuns com a aplicação a ser desenvolvida, mas mais uma vez, trata-se de uma aplicação comercial, isto é, sem código fonte aberto, não sendo desta forma possível expandir para outro tipo aplicações.

A Figura 2.11, apresenta algumas das funcionalidades da aplicação.

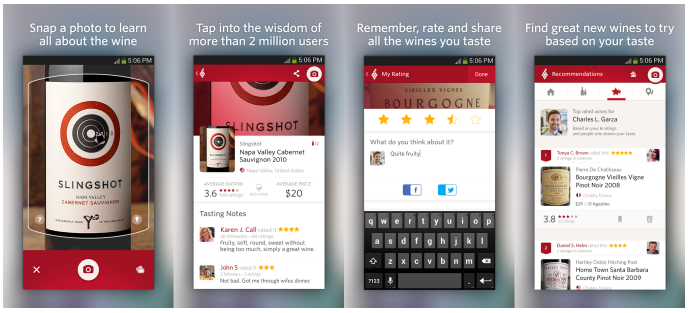

Figura 2.11.: Aplicação Vivino Wine. Fonte: Google Play (serviço de aplicações para Android)

2.7.8. Resumo de aplicações semelhantes

Todas as aplicações anteriormente referidas encontram-se de alguma forma relacionadas com o tema, no entanto, ou não podem ser usadas comercialmente e/ou o código encontra-se fechado. Algumas não reconhecem objetos, mas estão presentes pois apresentam características semelhantes. Nenhuma fornece uma API, que facilite a criação de novas aplicações para reconhecimento de outro tipo de objetos. É resumido na Tabela 2.3 as possibilidades de interesse das aplicações.

Tabela 2.3.: Comparação de aplicações semelhantes

	Permite o uso comercial	**Reconhece objetos**	**Fornece uma API facilmente expansível**
Sistema desenvolvido	Sim	Sim	Sim
Google Goggles	Não	Sim	Não
Augment 3D Augmented Reality	Não	Não	Não
Aplicação de reconhecimento de Livros em Coluna	Não	Sim	Não
Aplicação de reconhecimento de Ambientes exteriores	Não	Sim	Não
Layar	Não	Sim	Não
Word Lens Translator	Não	Não	Não
Vivino Wine Scanner	Não	Sim	Não

2.8. Resumo

O reconhecimento de objetos divide-se em dois campos fundamentais, baseados em marcadores ou sem marcadores. O uso de marcadores simplifica muito a tarefa de reconhecimento, mas requer que o ambiente seja modificado (adicionando os marcadores), o que pode não ser possível. No caso de deteção de cartazes de cinema não é fazível, já que os cartazes são controlados pelas companhias de cinema e não é habitual conter qualquer elemento que possa servir como marcador.

A fase principal de desenvolvimento passou pela criação da API que permite o reconhecimento de objetos. Existem dois métodos principais para a criação de uma API web service: SOAP e REST. O SOAP é mais maduro, tem mecanismos de tratamento de erros, mas é mais pesado que o REST. O REST é mais leve, eficiente e fácil de implementar, e é portanto mais adequado ao uso em dispositivos móveis.

De acordo com os requisitos de performance o sistema usa diversas linguagens que facilitem o desenvolvimento do sistema. Para a implementação do web service, foi adotado o framework Ruby on Rais devido à sua fácil integração com a linguagem C/C++ para programação das componentes críticas do sistema CV (OpenCV) e ser fortemente direcionada para web services e Java e tecnologias Web (com auxílio da ferramenta "Apache Cordova") foram usadas para o desenvolvimento da aplicação Android.

Como o sistema CV é muito dispendioso no CPU, a otimização do uso do mesmo deve ser um objetivo à partida, desta forma a divisão de tarefas por diferentes núcleos de processamento aumenta certamente a rentabilidade do sistema por esse motivo o OpenMP é uma opção no ato de compilação do sistema.

3. Implementação

Neste capítulo encontra-se destacada a arquitetura fundamental do sistema. Desta forma são demonstradas e explicadas as decisões feitas na escolha desta arquitetura e como estas estão agregadas de forma a resolver o problema em questão.

3.1. Arquitetura do Sistema

O sistema é composto por um conjunto de serviços interconectados. Os dispositivos móveis são responsáveis por enviar uma imagem ou os descritores respetivos para o servidor. Este processa-os estes dados com auxílio da API, e devolve um ID que representa um objeto na base de dados do serviço "Sapo Cinema". Por fim, o servidor envia para os dispositivos correspondentes a informação resultante do processamento (objetos do serviço "Sapo Cinema"), este processo é ilustrado na Figura 3.1.

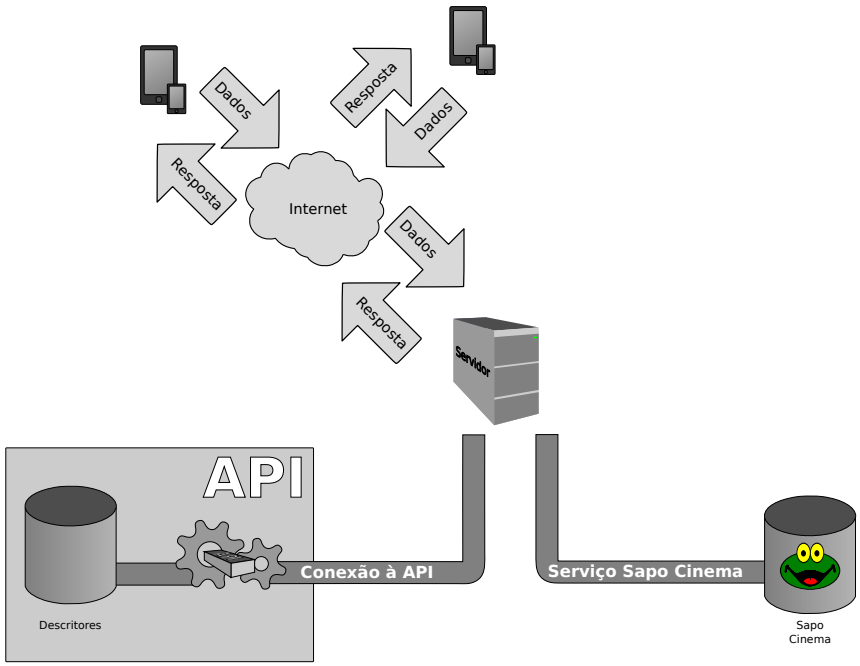

Figura 3.1.: Arquitetura da Solução Proposta

O acesso ao serviço "Sapo Cinema" não foi facultado. Desta forma para popular a base de dados foi usado "scraping", para isso foi criado uma tarefa "Rake". O programa "Rake" faz parte do "framework" "Ruby on Rails". Com esta tarefa a base de dados pode ser facilmente atualizada com o seguinte comando:

```
rake sapocinema:populate
```

Uma tarefa "Rake" é um programa escrito em Ruby, foi usado uma biblioteca Ruby que facilita a tarefa de "scraping" neste caso o "Mechanize".

3.1.1. Componentes da API

A API, desenvolvida usando várias bibliotecas e linguagens de programação, divide-se sobretudo em duas partes: a parte crítica (CV), desenvolvida em C/C++ e a interface pública desenvolvida em "Ruby on Rails", referida na Secção 3.1.2.

Esta API intitulada "libgor", "lib" de "library" e "GOR" (*Generic Object Recognition*), consiste sobretudo numa interface desenvolvida em C/C++ e um "wrap" com o auxilio da ferramenta SWIG 2.6.6, que permite a uma variedade linguagens de programação de alto nível conectar com programas escritos em C/C++. Isto é, é possível aceder às funções criadas no libgor através do Ruby de forma nativa pois foi criado um "wrap" que permite o acesso conforme ilustrado na Tabela 3.1.

Tabela 3.1.: Exemplo de funções em libgor (C++) e suas equivalentes em libgor_wrap (Ruby)

C++	**Ruby**
GorMat image; Settings settings; openImage(imagefile, image, settings);	image = new Libgor_wrap.GorMat() settings = new Libgor_wrap.Settings() Libgor_wrap.openImage(imagefile, image, settings)

Na Figura 3.2 é representada a forma como a API é dividida internamente.

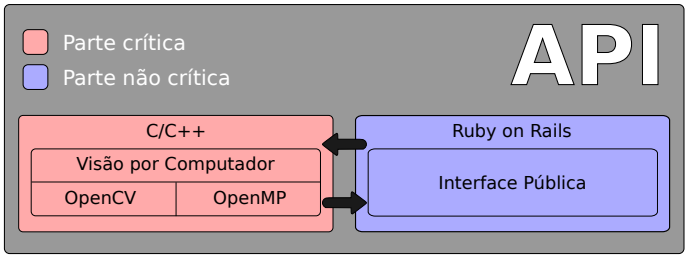

Figura 3.2.: Esquema interno da API

3.1.2. Mensagens da API (Interface Pública)

A API foi desenvolvida de forma a suportar objetos genéricos. Como tal, a base de dados dos descritores que suporta a API é mínima, isto é vai ser limitada apenas ao preenchimento obrigatório de três campos(id, descritor_path, data), dos quais o ID é auto gerado. Conforme a aplicação para o qual a API é usado pode-se acrescentar mais campos que podem auxiliar na pesquisa (exemplo: date e gps). O campo "descriptor_path" representa o descritor de um objeto, ou seja um elemento da base de dados de treino. O campo "data", representa os dados devolvidos pelo API. Como são suportados objetos genéricos, "data", pode representar um ID de um objeto noutra base de dados, um URL, ou mesmo um objeto em JSON. Assim as aplicações podem usar outros campos, para uma representação que faça sentido no contexto de pesquisa. Por exemplo, o campo nos cartazes de cinema poderia corresponder a "release date" do filme, assim como "gps", representar coordenadas GPS onde foi tirado a fotografia do objeto. Isto pode auxiliar na pesquisa numa aplicação de reconhecimento de monumentos ou verificar os filmes em exibição numa determinada sala de cinema.

Existe ainda uma tabela adicional que é onde os resultados aos pedidos são adicionados "requests". Nesta tabela existe apenas os campos "id", "token", "descriptor_id" e "hits", sendo que o "id" é auto gerado pela base de dados e o "descriptor_id" representa o objeto no qual obteve "hits" (chave estrangeira à tabela "descriptors"). O numero de "hits" é guardado na campo com o mesmo nome. O "token" é uma string gerada com o algoritmo de "hashing" SHA-256 sobre os dados recebidos, desta forma consegue-se garantir que o "token" é único e pode identificar um pedido.

Vários outros campos podem ser adicionados e filtrados na pesquisa. Os campos "date" e "gps", são apenas exemplos ilustrativos (dependendo da aplicação a usar a API podem ser criados estes outros campos). Um exemplo de um esquema da base de dados para a API pode ser visualizado na Figura 3.3.

Como os descritores são objetos grandes, estes são guardados em ficheiros em vez de usar uma base de dados relacional. Desta forma o campo "descriptor_path", representa o caminho para o ficheiro onde se encontra o descritor. Estes descritores podem ser guardados em XML, YAML e PNG, pode ser ainda aplicado uma variante dos formatos mais verbosos (XML e YAML) com o uso do algoritmo de deflate/inflate (usado ZLIB) para comprimir estes ficheiros. A biblioteca libgor já se encontra preparada para ler estes formatos ("file.xml.zlib"/"file.yml.zlib").

A interação com a API é feita através de um conjunto de mensagens que tentam ser simples, fáceis e flexíveis de usar, como é mostrado na Figura 3.4.

Esta API é usada pelo cliente para realizar os pedidos ao servidor, e desta forma o servidor realiza as comparações dos descritores, o treino do sistema e a devolução dos resultados em JSON.

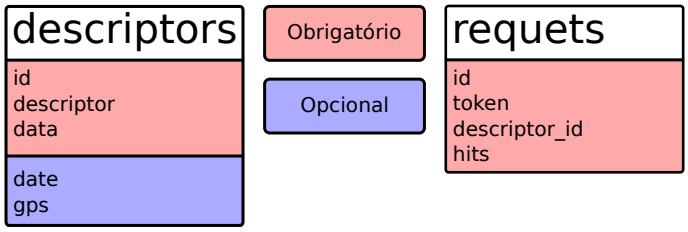

Figura 3.3.: Base de Dados da API

Um uso comum é comparar uma imagem com a base de dados de treino. Para isso executa-se o pedido `images(image, type)` , que suporta dois tipos de abordagens, o parâmetro "image", corresponde à imagem ou descritor e o parâmetro "type", indica o tipo "i" (imagem), "d" (descritor) ou "dz" (descritor comprimido com "deflate"). A API devolve assim um "token", uma lista com identificadores dos processos("job_id") e um estado("status") que indica se foi possível iniciar a pesquisa ("success" ou "fail"). O "token" identifica o pedido e o "job_id" identifica os processos que trabalham no pedido.

Desta forma resultados devem usar o token e o job_id para serem obtidos, `requests(token,job_id)` . Este comando evolve o estado do pedido ("unknown", "working", "queued", "failed", "complete") e neste caso uma lista de filmes que corresponde ao parâmetro "data" na base de dados. O estado "unknown" é devolvido apenas se o campo "job_id" é inválido ou inexistente, no entanto os resultados são devolvidos. Apenas o estado da pesquisa é desconhecido e desta forma não se consegue saber se a pesquisa terminou, não começou ou falhou.

A lista "job_id" é usada para identificar se os processos terminaram, já que a pesquisa é realizada de forma assíncrona. O "status" apenas identifica de forma genérica, se todos os processos terminam("complete"), se existe pelo menos um "working" ou "failed" ou se todos estão "queued". Este campo "job_id" é uma lista pois a tarefa de pesquisa pode ser dividida por vários processos, inclusive em centros de processamento distintos, isto é, a tarefa pode ser dividida por vários sistemas que trabalham em paralelo de forma a dividir a base de dados e obter os resultados o mais rápido possível. No entanto, esta lista é no contexto de JSON uma string. Os seus elementos apenas são divididos por um virgula, o que facilita o envio deste campo para `requests(token,job_id)` pelo método "GET".

Para obter um descritor usado no treino (incluindo os dados que representa) pode ser executado usando o método "POST" ou "GET" `descriptors(id)` , que permite obter apenas um elemento pelo ID. A resposta pode ser observada no Código JSON 2, no anexo A. É possível ainda combinar com operadores, conforme mostrado no Código JSON 1, no anexo A.

Para enviar uma imagem ou descritor para o servidor é usado o método "POST" com os

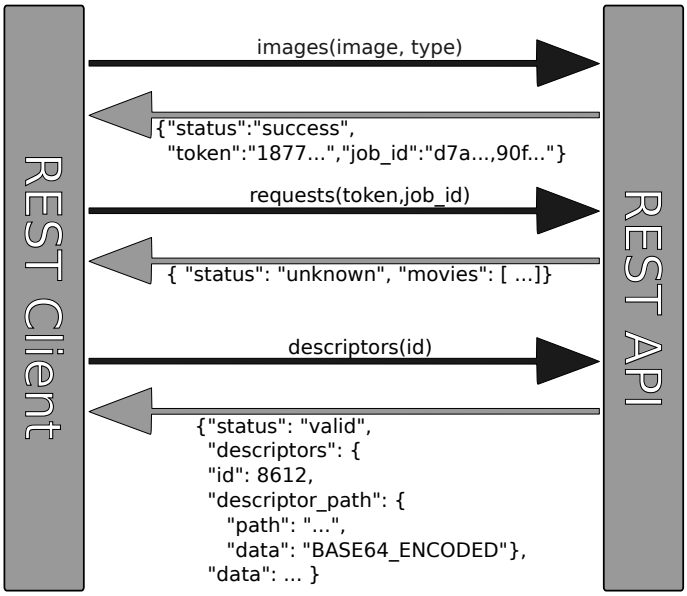

Figura 3.4.: Algumas Mensagens da API

campos “image” e “type”, que conforme referido anteriormente nesta secção 3.1.2, estes podem ser “i”, “d” ou “dz” conforme os dados enviados. A resposta pode ser observada no Código JSON 5, no anexo A. Não pode ser usado o método “GET”, pois embora não exista uma especificação com o limite exato, a maior parte dos servidores Web tem um limite de 8 KB para o pedido.

Outros exemplos do uso da API podem ser encontrados no anexo A. Um resumo dos métodos pode ser observado na Tabela 3.2.

Tabela 3.2.: Resumo de métodos do API Rest

URL	Parâmetros	Método Suportado	Descrição
/api/images	image, type	POST	Permite efetuar um pedido de pesquisa image: representa uma imagem ou um descritor ou um descritor. type: indica o tipo de objeto que é enviado (i - imagem ou d - descritor)
/api/descriptors	be_id, le_id, limit, offset	POST e GET	Permite obter descritores, filtrando-os com parâmetros. be_id: com ID >= ? le_id: com ID <= ? limit: Limitar a ? resultados offset: pagina ?
/api/descriptors/:id		POST e GET	Permite obter o descritor com o ID = :id
/api/requests	token, job_id	POST e GET	Receber resultados e estado da pesquisa token: identifica o pedido job_id: identifica os processos responsáveis pelo pedido
/api/train	image, data	POST	Permite adicionar um elemento à base de dados. image: imagem do objeto a ser treinado data: algo que identifica o objeto (exemplo: JSON)

3.1.3. Servidor

O servidor é responsável por conectar a aplicação Android com o serviço "Sapo Cinema", e usa a API para realizar as suas decisões.

Foi desenvolvido em "Ruby on Rails" e é responsável por tratar a informação do serviço "Sapo Cinema" e os dados recebidos do cliente. Representa um cliente do ponto de vista da API e um servidor do ponto de vista do utilizador, conforme indicado anteriormente na Figura 3.1.

Possui a capacidade de escalar para uma base de dados de grandes dimensões. Para esta tarefa é usado um sistema de processamento de trabalhos em "background" para Ruby, o "Sidekiq" 2.6.9. Este sistema usa uma base de dados extra de alta performance, não relacional, o "Redis" 2.6.8. No "Sidekiq", o primeiro passo consiste em serializar a tarefa a ser executada para o servidor "Redis". Depois, um processo "Sidekiq" verifica se tem tarefas no servidor "Redis" e executa as mesmas marcando-as com "working", "queued", "failed" e "complete", conforme o estado da execução.

Para escalar o servidor basta que seja fornecida a parte que recebe os pedidos e um servidor "Redis". Outros servidores ou processos podem depois ser inicializados. Desde que estes tenham acesso à base de dados, aos descritores e ao servidor "Redis", o sistema realiza uma auto gestão e permite responder a um maior número de pedidos simultâneos

e a divisão da base de dados dos descritores por vários núcleos de processamento. Pode ainda ser usado um "load-balancing" para permutar entre vários servidores (parte que recebe os pedidos) e aumentar ainda mais o número de pedidos simultâneos.

Um exemplo de uma configuração pode ser visualizada na Figura 3.5.

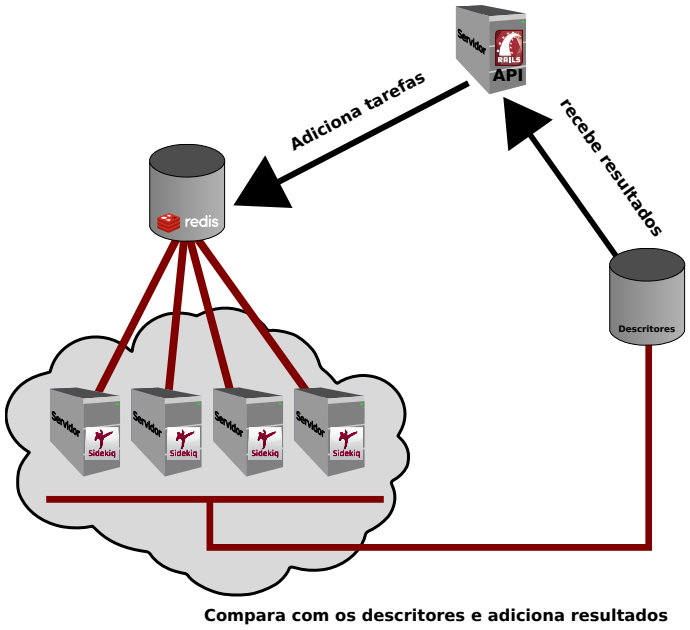

Figura 3.5.: Esquema de uma possível configuração da rede interna

3.1.4. Aplicação Cliente

A aplicação cliente foi desenvolvida para um dispositivo móvel (Android), com tecnologias Web (HTML5, JavaScript e CSS) para toda a interface e comunicação com o servidor. Foram ainda usados componentes em Java, neste caso para a extração dos descritores, e usar um "wrap" da biblioteca OpenCV disponível no "Google Play", o "OpenCV Manager".

De forma a testar as duas abordagens a aplicação pode ser configurada com a abordagem a ser usada.

Para a aplicação optou-se também pelo uso da ferramenta "Apache Cordova" 2.6.7, que permite desenvolver as aplicações com os standards Web, e estender de forma a que estas tenham acesso a partes nativas do aparelho, como tirar fotografias ou ainda abrir outras aplicações ou funcionalidades. Neste caso, isto foi feito através de um "plugin" para realizar tarefas extras como CV.

A aplicação requer que o dispositivo móvel tenha uma câmara e conexão com a Internet. Caso não exista conexão com a WWW a aplicação não consegue realizar a tarefa de

transmitir a imagem/descritores para o servidor, e o utilizador é informado através de uma mensagem de erro de conexão com a Internet.

Após a fase de verificação, o utilizador pode fotografar o cartaz, nesta fase a imagem é processada (redimensionada, convertida para escala cinza), dependendo da abordagem usada pode ser ainda retirado os descritores.

No caso de os descritores serem extraídos da imagem, estes são enviados para o servidor, caso este passo não exista, apenas a imagem é enviada.

O servidor então realiza o "match" com os descritores presentes na base de dados, e devolve a informação à aplicação.

A aplicação apresenta a informação de forma formatada ao utilizador, ou apresenta a mensagem de que não foi possível encontrar resultados, se for o caso.

A aplicação pode ser visualizada nas seguintes Figuras.

A Figura 3.6 representa a abertura da aplicação. No primeiro terço da figura é representado o "icon" no ambiente de trabalho de um dispositivo Android. No segundo terço, é representado a aplicação aberta quando esta não tem pedidos antigos, isto é, o botão do meio permite consultar o estado da ultima pesquisa, mas encontra-se desativado devido a ainda não ter sido efetuado pedidos ao servidor. O terceiro terço, representa a primeira opção do menu "Tirar Fotografia", aqui, é onde o utilizador escolhe tirar a fotografia e enviar para o servidor, desta forma, o servidor trata de encontrar respostas e enviar para a aplicação. Para que as respostas sejam passíveis consulta, o servidor envia um "token" e "job_id" conforme referido anteriormente na Secção 3.1.2, desta forma a aplicação pode consultar sempre que necessário o estado da pesquisa.

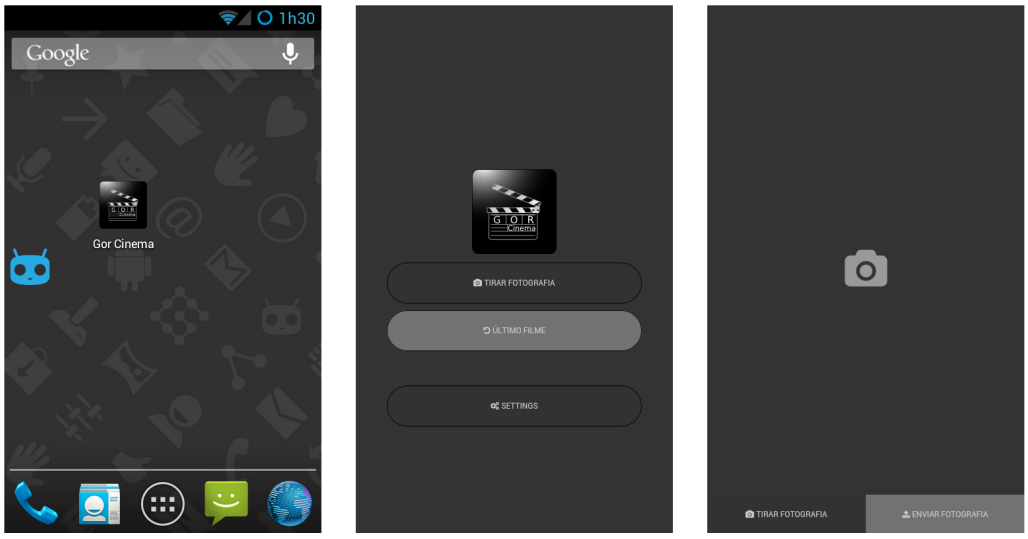

Figura 3.6.: Aplicação cliente quando é aberta

A Figura 3.7 representa a aplicação no ato de enviar a imagem para o servidor. No primeiro terço da figura, é representada a aplicação nativa do Android para tirar fotografias, esta aplicação foi aberta quando o utilizador pediu para tirar uma fotografia de acordo com o terceiro terço da Figura 3.6. No segundo terço, é representado a imagem tirada já redimensionada para o envio para o servidor. Aqui é possível constatar que o botão "Enviar Fotografia" foi ativado, pois já existe fotografia para ser enviada. O terceiro terço, representa quando a imagem já foi enviada, a aplicação consulta regularmente (de um em um segundo), o estado da pesquisa e conforme os resultados chegam, estes são adicionados. É de notar que no canto superior esquerdo existe um circulo animado, este circulo é normalmente usado em aplicações para informar que o processamento ainda não terminou.

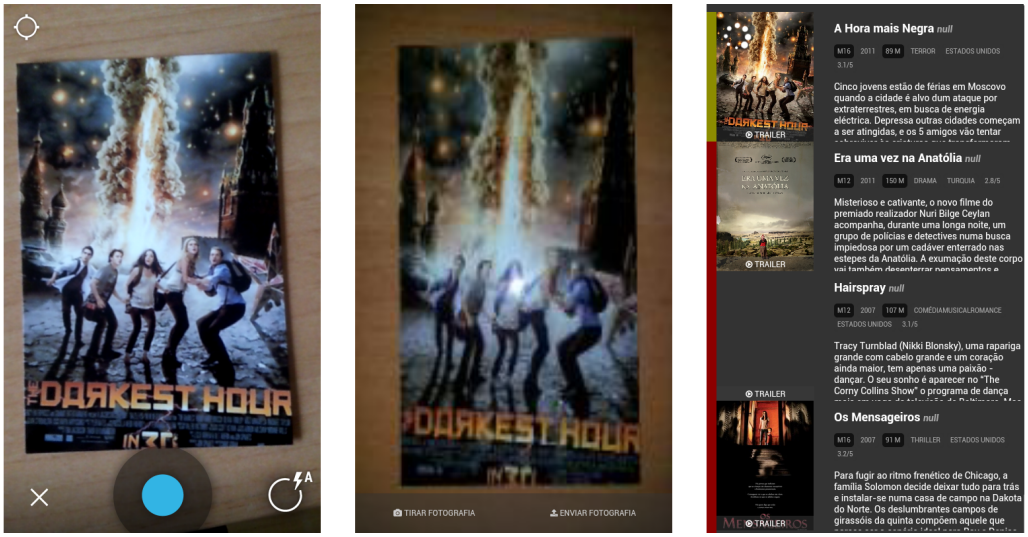

Figura 3.7.: Aplicação cliente quando é fotografado o objeto e enviado para o servidor

A Figura 3.8 representa a aplicação quando o processamento da imagem terminou, ou seja quando a pesquisa da imagem na base de dados do servidor termina, quando é detetado que a pesquisa termina a consulta regular cessa, isto porque já não é necessário uma vez que não vão ser adicionados resultados. No primeiro terço da imagem pode-se verificar que a pesquisa terminou já que, o circulo animado no canto superior esquerdo desapareceu. As cores laterais representam o quanto o sistema está confiante que é o resultado esperado, existem três cores possíveis: verde, indica que está bastante confiante, o amarelo não tem certeza e o vermelho, dificilmente é o resultado esperado. A lista encontra-se ordenada e os resultados de cima são os mais prováveis de ser os esperados. No segundo terço, é representado mais informação sobre o filme ao clicar sobre este elemento na lista. No terceiro terço da imagem, é representado o "trailer" do filme, este pode ser obtido ao clicar num cartaz que tenha a palavra "Trailer" por baixo.

A Figura 3.9 representa as configurações da aplicação. Estas configurações podem

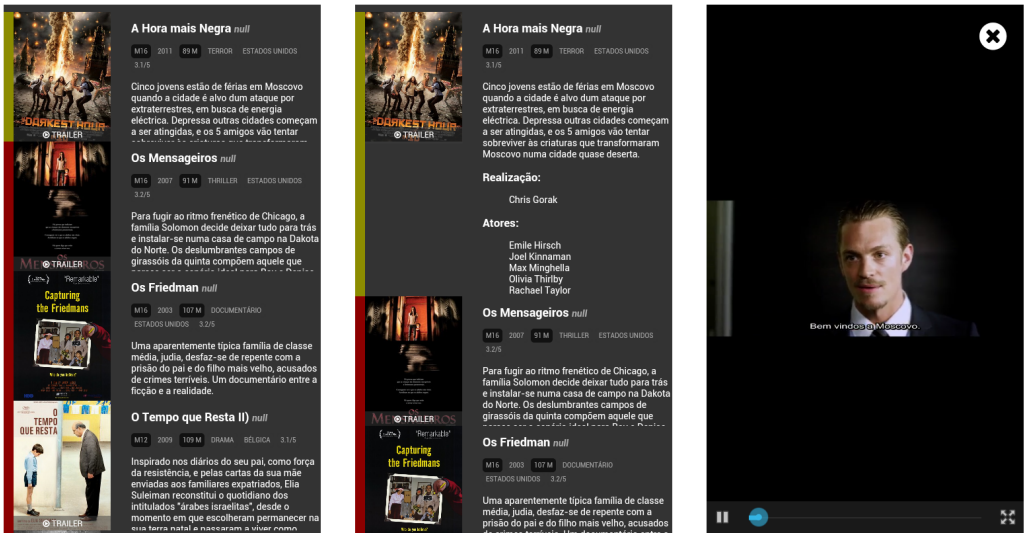

Figura 3.8.: Aplicação cliente quando a pesquisa termina e suas funcionalidades

ser acedidas ao clicar no botão "Settings" no primeiro terço da imagem, aqui também já se pode visualizar que a opção "Ultimo Filme", já se encontra ativado, pois já feito consultas com a aplicação. O segundo terço, representa as configurações da aplicação, que se resume em enviar descritores em vez de imagem, e o formato em que o descritor é enviado. A diferença entre o segundo e terceiro terço da imagem é o facto da opção estar ou não ativada. Estas opções influenciam a forma como a aplicação comunica com o servidor de acordo com a Secção 3.1.2.

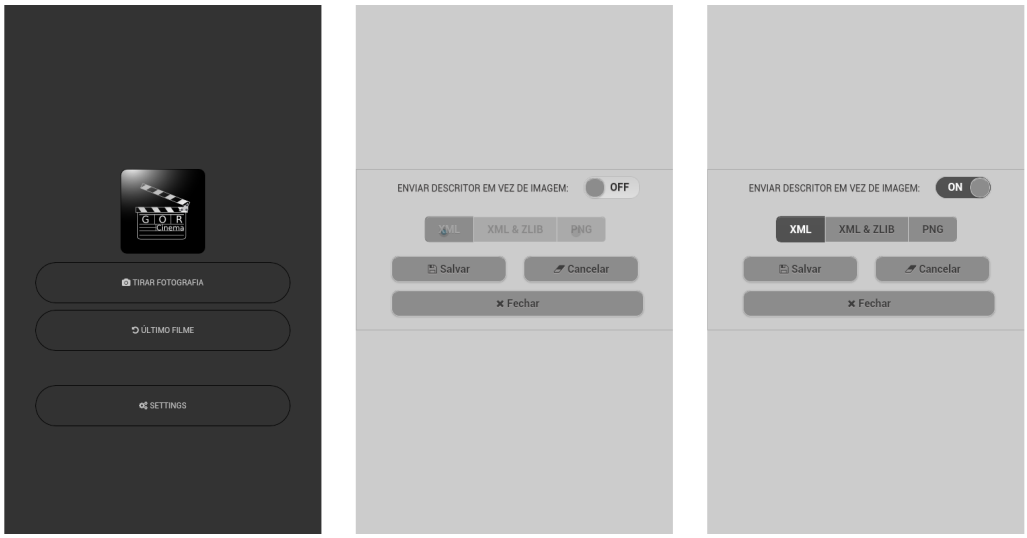

Figura 3.9.: Aplicação cliente nas suas configurações

3.2. Resumo

O sistema encontra-se dividido em quatro partes, a aplicação Android o servidor (API Rest), a API (C++/Wrap Ruby) e o serviço "Sapo Cinema".

Duas abordagens são consideradas neste sistema, como referido na Secção 3.1.4: numa a imagem redimensionada é convertida para escala cinza e por fim é enviada, noutra são enviados os descritores da imagem. Em ambas as abordagens o servidor faz "match" do descritores, mas na primeira hipótese o servidor tem de extrair os descritores.

A biblioteca OpenCV é usada na API, para a visão por computador, mas pode também ser usada na aplicação se a hipótese da aplicação extrair os descritores for selecionada na aplicação.

A API Rest, funciona como um serviço em que a aplicação cliente envia e recebe a informação.

A parte crítica encontra-se desenvolvido em C/C++ e tem um "wrap" (criado com o auxilio da ferramenta SWIG) para Ruby, que permite aceder a estas funções.

A implementação que usa OpenMP pode ser ativada no ato de compilação da API, o utilizador deve ter em conta se compensa o uso da versão OpenMP, considerando que o OpenCV já pode usar "multi-thread".

4. Testes e Resultados

Este capítulo é dedicado a apresentar todos os testes e resultados executados durante a tese. Como a aplicação e a API é vocacionada para o uso em tempo real, vários fatores têm de ser avaliados: o tempo necessário para pesquisar em toda a base de dados, a taxa de sucesso relativo a questões realizadas e o tipo de abordagens.

4.1. Especificações do hardware e Condições dos Testes

O "smartphone" usado é um Samsung Galaxy S2 (I9100), que contém 1GB de RAM e um CPU Dual-core 1.2 GHz Cortex-A9, com o sistema operativo Android 4.2.2 (não oficial CyanogenMod 10.1.3). O computador que serve como plataforma de desenvolvimento e testes, tem um CPU Intel Intel(R) Core(TM)2 Duo T9400 2.53GHz, e 4GB de RAM DDR2, ainda os discos são de 5400 rpm.

O "dataset" foi obtido a partir do serviço "Sapo Cinema" e representa as imagens de treino, que são imagens com os quais os pedidos vão ser comparados, consiste num total de 3570 descritores extraídos e guardados. Outras imagens de diferentes cartazes foram tiradas com a câmera do "smartphone" e redimensionadas para o tamanho final de 192x144 pixeis.

As imagens tiradas com o "smartphone" são impressas em papel normal e fotográfico, com e sem flash, diferentes ângulos e distancias, de modo a simular condições reais. Exemplo podem ser observados na Figura 4.1.

Figura 4.1.: Imagens extraídas do dataset de fotografias tiradas com telemóvel

4.2. Transferência da informação via Rede

Um dos problemas nestes sistemas é a necessidade de ligação com o servidor por forma a que seja possível comparar com um extensa base de dados. Isto porque é necessário transferir a imagem ou os descritores, o que pode ser um problema se estes dados forem grandes não só pelo tempo necessário como também pelos custos de utilização da aplicação imposta pelas operadoras (serviço de Internet).

Por forma a resolver este problema é necessário diminuir o tamanho dos dados enviados, já que uma imagem tirada por um "smartphone" pode facilmente ultrapassar 1.5MB, se já se encontrar comprimido num formato como JPG (normalmente o caso). No entanto o OpenCV tem um limite para a extração de descritores, ou seja, a imagem tem de ultrapassar ou ser igual em largura ou altura a 300 pixeis.

Por exemplo a imagem original tirada pelo "smartphone", tem 3MB com as dimensões 2448x3264, redimensionada para 225x300 ocupa drasticamente menos: 72,1KB.

O algoritmo usado para redimensionar tem custos muito reduzidos (algoritmo nativo do "Cordova" no ato de tirar a fotografia), por isso optou-se por redimensionar a imagem para 144x192 pixeis, fica claro que a imagem fica mais pequena e fácil de transmitir, no entanto esta tem de der redimensionada (algoritmo nativo do OpenCV) no servidor para que este a possa usar (limite mínimo de 300 pixeis em altura ou largura). Durante estas experiências observou-se que uma imagem por ser redimensionada pode melhorar o resultado, no entanto quanto maior a imagem maior o tempo de pesquisa, pelo que o tamanho não deve ser nem muito grande nem muito pequeno. Adicionalmente a normalização da imagem, pode melhorar os resultados. Estas operações tem bom resultado em imagens que vão ser comparadas mas não nas imagens de treino.

O facto de se ter usado normalização nas imagens de treino originou falsos positivos. Por outro lado o redimensionamento (escalar) as imagens de treino não traz benefícios já que estas encontram-se com um máximo de 500x700 pixeis (imagens no SapoCinema). Como o treino ocorre apenas uma vez o reduzir as dimensões das imagens de treino poderia reduzir os descritores extraídos com efeitos adversos, enquanto que aumentar as dimensões aumenta o tempo necessário sem garantias de melhorias. Por isso aplicar qualquer fator de escala numa imagem de treino é um procedimento que deve ser efetuado com cautela e ponderação (aplicar a escala reduzida a uma imagem, pelo facto de reduzir o número de descritores pode colocar o sistema mais rápido, mas implica perda precisão).

No servidor aplicar um fator de escala a quatro vezes superior e efetuar a normalização não ultrapassou o 1ms, e o facto de aumentar quatro vezes fez com que a imagem se encontre mais próximo do tamanho original das imagens de treino (de 144x192 para 576x768 contra 500x700), o que melhora substancialmente os resultados.

A colocação de uma tabela para as 3570 imagens tiradas com o "smartphone" seria impossível pelo que foi feito um excerto de imagens com o número de "hits" que os pedidos obtiveram com as diferentes configurações: Tabela 4.1. O uso da letra "N" representa que foi usado normalização.

Tabela 4.1.: Exemplo de resultados com diferentes configurações ao abrir a imagem no servidor

Imagem de teste	225x300	288x384	288x384 + N	576x768	576x768 + N
	3	7	16	29	63
	6	12	10	29	38
	5	14	23	46	57

O envio de descritores, em vez de imagens reduzidas, traz poucos benefícios já que embora neste "smartphone" o tempo necessário para extrair os descritores é negligenciável, o tamanho dos descritores quase sempre ultrapassam o tamanho das imagens. Uma vez que a extração dos descritores no servidor ainda mais rápido, é recomendado que apenas se envie a imagem.

4.3. Formatos de armazenamento dos descritores

Existem três formatos principais usados durante a tese: serialização nativa do OpenCV XML e YAML e PNG, formato de imagem na sua configuração "lossless", ou seja um formato que comprime a imagem mas sem perdas de informação. Existe outro formato criado nesta tese que consiste em comprimir o XML e YAML com o algoritmo "deflate" (com auxílio da biblioteca zlib).

Apenas foram testados o formato XML e PNG, no que se referem a tempos de pesquisa e tamanhos finais, no entanto todos os formatos encontram-se implementados.

O formato PNG apresentou melhores resultados, pois este é mais pequeno que a versão serializada em XML (menos acesso ao disco), é um formato de imagem, uma imagem é uma matriz, e um descritor é representado como uma matriz, desta forma a imagem é o descritor e não exige conversão entre tipos de dados, estes resultados podem ser observados na Tabela 4.2.

Na Tabela 4.2 está presente uma comparação entre duas abordagens de pesquisa, no qual na abordagem com "k-means" existe dois exemplos, assim é mostrado que o número

de "clusters" tem importância para o tempo de pesquisa, mas a precisão é afetada como referido na Secção 4.4.

Os resultados nestas tabelas encontram-se em segundos, AVG corresponde ao tempo médio, 1 Divisão corresponde a execução de uma instância de processamento, ou seja com a base de dados completa. Quanto a 2 Divisões significa que a base de dados foi dividida em dois, e T1 significa na primeira instância "thread" e T2 na segunda.

Tabela 4.2.: Comparação da divisão da base de dados e formatos de armazenamento dos descritores.

Abordagem Simples				
Formato do Descritor	**1 Divisão**	**2 Divisões T1**	**2 Divisões T2**	**2 Divisões Efetivo**
PNG	141.296	134.164	138.254	138.254
	141.395	138.101	134.709	138.101
	141.090	134.547	138.288	138.288
	141.595	133.690	138.096	138.096
AVG	**141.344**	**135.126**	**137.337**	**138.185**
XML	158.920	144.370	139.863	144.370
	156.454	139.140	143.977	143.977
	154.796	148.033	144.298	148.033
	155.513	144.019	139.328	144.019
AVG	**156.421**	**143.891**	**141.867**	**145.100**
Abordagem com "k-means" 100 "clusters"				
Formato do Descritor	**1 Divisão**	**2 Divisões T1**	**2 Divisões T2**	**2 Divisões Efetivo**
PNG	55.282	72.314	72.213	72.314
	52.571	67.952	69.924	69.924
	55.218	77.704	76.121	77.704
	54.641	71.016	71.378	71.378
AVG	**54.428**	**72.247**	**72.409**	**72.830**
Abordagem com "k-means" 64 "clusters"				
Formato do Descritor	**1 Divisão**	**2 Divisões T1**	**2 Divisões T2**	**2 Divisões Efetivo**
PNG	43.808	66.907	67.511	67.511
	42.331	69.305	68.668	69.305
	44.493	65.722	66.225	66.225
	42.950	64.105	65.503	65.503
AVG	**43.396**	**66.510**	**66.977**	**67.136**

O formato PNG apresenta uma base de dados muito mais pequena, que a sua versão serializada em XML, cerca de 27.16% do tamanho da versão XML, conforme a Tabela 4.3.

Tabela 4.3.: Tamanho da base de dados nos formatos XML, XML+ZLIB e PNG para 3570 cartazes.

XML	XML+ZLIB	PNG
595.3 MB	205.4 MB	161.7 MB

4.4. Execução dos pedidos

O tempo necessário para percorrer a base de dados depende do formato em que os descritores se encontram armazenados e da abordagem usada. Existem duas abordagens usadas: uma com "k-means" outra sem.

A abordagem sem "k-means" consiste em percorrer a base de dados em conjuntos de cem elementos de cada vez. Em cada conjunto é realizada uma comparação com os descritores extraídos da imagem tirada pelo "smartphone". A imagem que tiver maior número de "hits" desse conjunto é selecionada e adicionada à base de dados (é armazenado na tabela "requests", Figura 3.3) com o identificador do pedido ("token").

A abordagem com "k-means" consiste em carregar toda a base de dados para um objeto "FlannBasedMatcher" do OpenCV com o "KMeansIndexParams", que permite usar o k-means hierárquico na pesquisa, a pesquisa devolve um vetor com os indexes dos descritores e respetivos "hits", no qual são adicionados à base de dados com o identificador do pedido ("token"). Nesta abordagem é também realizado um passo sobre os descritores, reduzido o número máximo de pontos de interesse, isto é, é realizado "k-means" sobre o próprio descritor. Este passo é realizado apenas uma vez sobre a base de dados, com o auxílio de uma ferramenta criada durante esta tese, que permite converter os descritores de um formato para outro (XML, PNG e YAML) com opção de realizar o "k-means" sobre o próprio descritor.

Uma vez executado "k-means" sobre o próprio descritor este torna-se incompatível com a abordagem simples (sem "k-means"), por isso os descritores originais devem ser guardados, para ser possível trocar de abordagens ou o número de "clusters". Para este efeito foi criado uma ferramenta auxiliar que permite converter os descritores entre formatos e aplicar "k-means" com um número de "clusters", se assim for desejado, para isso recomenda-se que no ato de popular a base de dados seja usado os descritores da abordagem simples, desta forma é possível trocar de abordagem mais tarde.

Embora a abordagem com "k-means" seja mais rápida, na prática os resultados não são apresentados os resultados de forma assíncrona (os resultados são apresentados apenas no final) o que pode originar um problema para o utilizador se a base de dados for muito grande. Este problema resulta do facto de ser necessário carregar todos os descritores em memória em simultâneo e realizar as comparações em memória. Uma forma de

ultrapassar será com a divisão da base de dados em partes mais pequenas por vários núcleos de processamento.

Um problema que afeta ambas as abordagens é facto do sistema requer muitos pedidos concorrentes "IO" ao servidor, o que poderá ser ultrapassado com o uso de discos mais rápidos como discos SSD.

A divisão da base de dados pelos processos consiste em atribuir um conjunto de dados a um determinado processo, isto é, a base de dados é dividida em X partes, assim um processo apenas tem conhecimento de N descritores (N corresponde ao número de descritores resultante da divisão da base de dados em X partes). Isto ocorre quer na abordagem sem "k-means" como com "k-means".

Na abordagem com "k-means" é necessário ter em conta o número de "clusters"; Este número vai indicar quantos pontos são usados no máximo para comparar as imagens, quanto maior o número maior a precisão, mas torna-se mais dispendioso temporalmente. Inversamente, quanto menor número de "clusters" menor é tempo de execução, mas implica menor precisão, resultando em falsos positivos. Assim deve ser encontrado um equilíbrio para se poder tirar partido deste método.

Foi realizada uma colheita de dados para comparação de performance das abordagens, com diferentes configurações conforme a Tabela 4.2. Nesta tabela pode-se observar que na abordagem simples a simples troca do formato XML para PNG melhorou a performance em cerca de 9.64% para execução por uma instância e 4.77% para uma execução com duas instâncias. Pode ser ainda observado que o uso de "k-means" com 100 "clusters" obteve uma melhoria de performance de cerca de 61.9% para uma instância e 47.30% para duas. Uma possível explicação para estes números será que o OpenCV já suporta multi-core e estão a ser consumidos ainda mais recursos ao dividir a base de dados (sobrecarga de "threads" no sistema). Desta forma é possível que se os testes fossem realizados noutra máquina com mais cores, ou com instâncias em máquinas separadas, os resultados fossem melhores, como esperado.

Para realizar o teste do serviço é usado a ferramenta "curl", alguns exemplos podem ser observados no anexo A. Apenas é contabilizado o tempo em que a parte CV é executada, estes dados são obtidos diretamente pelo "sidekiq" que fornece a informação de quanto tempo durou a tarefa desde seu inicio, ou seja tempo despendido pelos protocolos de Rede não são contabilizados.

4.5. Resumo

Na realização dos teste é usado um hardware com algum tempo, e a performance pode ser consideravelmente melhor com hardware de média gama recente.

O uso da abordagem que consiste em enviar a imagem reduzida para o servidor (Secção 3.1.2) é recomendada. Existem dois formatos de armazenamento dos descritores testados, serializados para XML ou PNG, este último formato fornece maior performance já que é necessário menos acesso aos discos do computador por este ser mais pequeno e um descritor é uma matriz e o formato PNG é um formato de imagem.

Existem duas forma de realizar a comparação com a base de dados, comparação simples ou com "k-means", o "k-means" apresentou melhores resultados como referido na Tabela 4.2.

Os resultados são obtidos pela "Rest API" com e podem ser observados exemplos no anexo A.

5. Conclusões e trabalho futuro

Neste capitulo é fornecida uma reflexão sobre os resultados e objetivos alcançados. Tenta também sugerir estudos futuros, para melhorias de sistemas semelhantes, assim como melhorias ao sistema que puderam vir a ser implementadas.

5.1. Conclusões

Com o objetivo de adquirir um estudo de métodos de visão por computador e tecnologias vocacionadas para "webservice" capaz de: fornecer uma API, ser apta a reconhecer objetos genéricos em tempo real e ser usável por dispositivos móveis, a tese propõe um sistema que pretende ser simples e fácil de usar e estender com novos módulos. São usados métodos de rastreio sem marcadores, e como tal é necessário poder computacional, para auxiliar nessa tarefa. É aproveitada ao máximo a capacidade dos processadores com múltiplos núcleos de processamento, usando-se métodos de "clustering" como "k-means" para obter rapidamente a informação pretendida, assim como a distribuição da base de dados por vários núcleos de processamento. O sistema é dividido em vários subsistemas, API, Servidor e por fim a aplicação cliente, este sistema encontra-se apto para qualquer situação de reconhecimento, desde que as imagens tenham sido fornecidas para a base de dados.

No desenvolvimento da API (Capítulo 3) é usado a tecnologia baseada em REST, sendo para tal o framework Ruby on Rails usado. No entanto a parte crítica do sistema (CV) usa C/C++, com a possibilidade de realizar a parte do processamento no lado do dispositivo móvel. Neste caso é usado o "OpenCV Manager", uma aplicação que serve como um "wrap" em Java para a biblioteca OpenCV. A API usa o FREAK para extração dos descritores e o ORB para deteção dos pontos de interesse (Capítulo 2).

O Servidor foi desenvolvido em "Ruby on Rails" (Capítulo 3), e conecta a API e o serviço Sapo Cinema, respondendo aos pedidos da aplicação Cliente.

A aplicação Cliente foi desenvolvido para Android (Capítulo 3), e pode usar duas abordagens, numa a imagem é enviada para o servidor, e o servidor extrai os descritores, noutra apenas os descritores são enviados para o servidor, poupando assim esta tarefa ao servidor.

Foram efetuados testes sobre diversos aspetos do sistema (Capítulo 4), desde o formato de armazenamento dos descritores XML e PNG, o uso de "k-means" e sem "k-means" na pesquisa, assim como a redução das imagens e métodos que permitam melhorar a precisão do sistema de pesquisa.

Os resultados mostram que o uso de "k-means" é mais rápido, mas há uma degradação de performance. Os resultados são apresentados apenas no final da pesquisa e não no decorrer da pesquisa. É recomendado que a imagem tirada pelo dispositivo "smartphone", seja redimensionada para um tamanho menor antes do envio (reduz custos de envio), posteriormente redimensionada no servidor. Adicionalmente, a normalização da imagem, aumenta a precisão durante a pesquisa.

A performance encontra-se associado ao tempo de execução e à precisão. Como a aplicação tem de ser executada em tempo real (tempo útil), considera-se que a aplicação tem a melhor performance quando é executada em tempo útil com a melhor precisão possível nesse tempo, ou seja, com o maior fator de identificação correta da imagem. Desta forma a utilização do "k-means", tem de ser estudado, para identificar o equilíbrio necessário.

Devido à natureza dos algoritmos usados, este sistema pode ser usado para fins comerciais sem necessidade de pagar patentes.

5.2. Trabalho futuro

Os testes foram efetuados em apenas um computador, mas podem ser realizados também numa estrutura semelhante à da Figura 3.5. O sistema "k-means" à semelhança do simples poderá ser implementado de forma a que ao percorrer a base de dados este efetue comparações em pequenos conjuntos de dados (por exemplo de 100 em 100). Isto deverá aumentar o sentimento de assincronicidade, isto é, na proposta atual a proposta simples começa a devolver resultados mais rapidamente apesar de demorar mais, conforme a Secção 4.3 indica.

Os resultados podem ser melhorados no simples (e em "k-means" se a base de dados é dividida), se for adicionado um parâmetro de ordenação. Por exemplo, adiciona-se coordenadas GPS, depende da aplicação que vai ser desenvolvida para usar esta API. No caso de cartazes de cinema, o uso de coordenadas GPS pode referir-se a locais onde a fotografia foi tirada e pode-se assim, limitar a pesquisa aos filmes em exibição (ou com exibição numa data próxima) às salas de cinema perto daquele local. Estas funcionalidades não devem representar problemas na implementação, pois o sistema está desenvolvido de forma modular que torna simples a sua expansão.

Algumas sugestões propostas por [Tei13], ainda se aplicam como:

- Verificar se a imagem tirada se encontra focada ou não no dispositivo, antes desta ser enviada para o servidor, desta forma pode-se, pedir ao utilizador que volte a tirar a fotografia.
- O uso de OCR pode ser estudado, se é benéfico a sua implementação em conjugação com o sistema atual, por exemplo o texto detetado pode ser usado como filtro de pesquisa, como OCR pode não ser bem sucedido o filtro deve funcionar com palavras próximas e não exatas.
- Apesar de ter sido testado em redes 3G, apenas os testes em Wi-Fi são aprofundados na tese.

Atualmente o preenchimento da base de dados foi efetuada através de "scraping" do serviço "Sapo Cinema", pelo que a funcionalidade pode ser melhorada se for fornecido acesso direto ao serviço.

Referências

[AKB08] Motilal Agrawal, Kurt Konolige e MortenRufus Blas. Censure: Center surround extremas for realtime feature detection and matching. In David Forsyth, Philip Torr e Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, volume 5305 of *Lecture Notes in Computer Science*, pages 102–115. Springer Berlin Heidelberg, 2008. URL: http://dx.doi.org/10.1007/978-3-540-88693-8_8, doi:10.1007/978-3-540-88693-8_8.

[AOV12] A. Alahi, R. Ortiz e P. Vandergheynst. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510–517, 2012. doi:10.1109/CVPR.2012.6247715.

[BTG06] Herbert Bay, Tinne Tuytelaars e Luc Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof e Axel Pinz, editors, *Computer Vision – ECCV 2006*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer Berlin Heidelberg, 2006. URL: http://dx.doi.org/10.1007/11744023_32, doi:10.1007/11744023_32.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha e Pascal Fua. Brief: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos e Nikos Paragios, editors, *Computer Vision – ECCV 2010*, volume 6314 of *Lecture Notes in Computer Science*, pages 778–792. Springer Berlin Heidelberg, 2010. URL: http://dx.doi.org/10.1007/978-3-642-15561-1_56, doi:10.1007/978-3-642-15561-1_56.

[CSK+10] David Chen, Tsai Sam, Kyu-Han Kim, Cheng-Hsin Hsu, Jatinder Pal Singh e Bernd Girod. Low-cost asset tracking using location-aware camera phones. volume 7798, California, 2010. Stanford University. Keywords: Visual search, location sensing, inventory management. URL: http://www.stanford.edu/~dmchen/documents/SPIE2010_BookSpineRecognition.pdf, doi:10.1117/12.862426.

[HS88] Chris Harris e Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[HW79] J. A. Hartigan e M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979. URL: http://dx.doi.org/10.2307/2346830, doi:10.2307/2346830.

[LCS11] S. Leutenegger, M. Chli e R.Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555, 2011. doi:10.1109/ICCV.2011.6126542.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999. doi:10.1109/ICCV.1999.790410.

[RD05] Edward Rosten e Tom Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515 Vol. 2. 2005. doi:10.1109/ICCV.2005.104.

[RRKB11] E. Rublee, V. Rabaud, K. Konolige e G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571, 2011. doi:10.1109/ICCV.2011.6126544.

[Sat02] Mitsuhisa Sato. Openmp: parallel programming api for shared memory multiprocessors and on-chip multiprocessors. In *System Synthesis, 2002. 15th International Symposium on*, pages 109–111, 2002.

[ST94] J. Shi e C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593–600, 1994. doi:10.1109/CVPR.1994.323794.

[Tei13] Hugo Teixeira. *Recognition of Visual Objects Using Mobile Devices*. MSc thesis, Faculdade de Engenharia da Universidade do Porto, 2013.

[TXG+08] Gabriel Takacs, Yingen Xiong, Radek Grzeszczuk, Vijay Chandrasekhar, Wei-Chao Chen, Kari Pulli, Natasha Gelfand, Thanos Bismpigiannis e Bernd Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. New York, 2008. Stanford University. URL: http://www.stanford.edu/~bgirod/pdfs/Takacs_MIR2008.pdf, doi:10.1145/1460096.1460165.

[zMNS05] Michael zur Muehlen, Jeffrey V. Nickerson e Keith D. Swenson. Developing web services choreography standards—the case of {REST} vs. {SOAP}. *Decision Support Systems*, 40(1):9 – 29, 2005. <ce:title>Web services and process management</ce:title>. URL: http://www.sciencedirect.com/science/article/pii/S0167923604000612, doi:http://dx.doi.org/10.1016/j.dss.2004.04.008.

A. Código JSON da API

Neste anexo encontra-se disponível o código JSON devolvido pela API a determinados pedidos. Os pedidos encontram-se como comentários ao código JSON, no entanto deve-se entender que estes comentários apenas se encontram aqui presentes e não são devolvidos pela API, assim como a formatação (embelezamento) não existe na resposta.

Por questões estéticas, a palavra "BASE64_ENCODED" foi usada para descrever um descritor, uma vez que um descritor pode conter facilmente mais 100kB de informação de uma matriz serializada. Os descritores são convertidos a partir de ficheiros binários para base 64, o que não é de fácil leitura (`aGlzIG9ubGluZSBzYW1wbGUgZGVtb25zdH...`). O campo descriptors[0].descriptor_path.path representa o caminho no "FileSystem" do servidor, e o valor `descriptors[0].descriptor_path.data` representa o descritor, que com o auxilio de `descriptors[0].descriptor_path.path` consegue-se saber qual o formato gravado, nestes casos os descritores encontram-se armazenados no formato PNG.

Nos filmes (campo descriptors[0].data), a "synopsis" foi reduzida também por poder conter várias linhas e não ser essencial para a compreensão de como os dados se encontram estruturados.

```
/**
* Resposta ao pedido: http://exemplo/api/descriptors?be_id=8700&le_id=9000&limit=2&
    offset=0
*
* Comandos de Teste:
* echo '{"be_id":8700,"le_id":9000,"limit":2,"offset":0}' > api.json
* curl -X POST --data-binary @api.json -H "Content-Type: application/json" http://
    exemplo/api/descriptors
*/

{
    "status": "valid",
    "descriptors": [
        {
            "id": 8700,
            "descriptor_path": {
                "path": "descriptor/
                    a1b239540ed921a1e7714ab8e886686246bdf897657e62f953ad6c4721c4aad1.png
                    ",
                "data": "BASE64_ENCODED"
            }, "data": {
                "titles": {"original": "Beasts of the Southern Wild",
                            "pt": "Bestas do Sul Selvagem"},
                "year": 2012,
                "poster": "http://sm2.imgs.sapo.pt/mb/C/K/J/X3dxWFpXe56vr6eM5OOmvit8_.
                    jpg",
                "trailer": "http://videos.sapo.pt/22Hz2sS8U5JnT8udIFJh/mov/1",
                "duration": "93 m",
                "genre": "Drama", "indicated_age": "M12",
                "country": "Estados Unidos", "user_review": 3,
                "synopsis": "<p>Numa comunidade esquecida mas desafiante de uma zona
                    pantanosa ...",
                "production": ["Benh Zeitlin"],
                "actors": ["Dwight Henry", "Levy Easterly", "Lowell Landes",
                            "Quvenzhané Wallis"]
            }, "release_date": null, "created_at": null,
            "updated_at": "2013-11-29T16:13:15.240Z"
        },{
            "id": 8701,
            "descriptor_path": {
                "path": "descriptor/34784
                    b5f7f0a5c92aed11de7227c67794a43ece7d2c95d2adc72de7abf814969.png",
                "data": "BASE64_ENCODED"
            }, "data": {
                "titles": {"original": "A Good Day to Die Hard",
                            "pt": "Die Hard - Nunca é bom dia para morrer"},
                "year": 2013,
                "poster": "http://sm2.imgs.sapo.pt/mb/0/d/e/l1T2C8jh4dV8j66Yvl1AAr-s_.
                    jpg",
                "trailer": "http://videos.sapo.pt/Wgrmo5NzgomSt4N60UME/mov/1",
                "duration": "98 m",
                "genre": "Ação/Aventura", "indicated_age": "M12",
                "country": "Estados Unidos", "user_review": 3,
                "synopsis": "<p>Quando Jack, o filho de John McClane, se mete ...",
                "production": ["John Moore"],
                "actors": ["Amaury Nolasco", "Bruce Willis", "Cole Hauser",
                            "Jai Courtney", "Mary Elizabeth Winstead",
                            "Patrick Stewart"]
            }, "release_date": null, "created_at": null,
            "updated_at": "2013-11-29T16:13:19.093Z"
        }
    ]
}
```

Código JSON 1: Obter num limite de dois os descritores na primeira página com ID >= 8700 e ID <= 9000

```
// Resposta ao pedido: http://exemplo/api/descriptors/8612

{
    "status": "valid",
    "descriptors": {
        "id": 8612,
        "descriptor_path": {
            "path": "descriptor/8
                a2d80f080821386ffed6a67eeccf21d792c3f0b9e939b7628159c9db93c99d6.png",
            "data": "BASE64_ENCODED"
        },
        "data": {
            "titles": {
                "original": "Grudge Match",
                "pt": "Grudge Match"
            },
            "year": 2013,
            "poster": "http://sm2.imgs.sapo.pt/mb/F/7/e/vcxPE-yZdPtMZr6jqNn35AHo_.jpg",
            "trailer": "http://videos.sapo.pt/wQ7YQ4nnZw64ZevwPhZH/mov/1",
            "duration": "113 m",
            "genre": "Comédia",
            "indicated_age": "M12",
            "country": "Estados Unidos",
            "user_review": 3,
            "synopsis": "",
            "production": [
                "Peter Segal"
            ],
            "actors": [
                "Alan Arkin",
                "Jon Bernthal",
                "Kevin Hart",
                "Kim Basinger",
                "Robert De Niro",
                "Sylvester Stallone"
            ]
        },
        "release_date": null,
        "created_at": null,
        "updated_at": "2013-11-29T16:08:40.894Z",
        "describe": null
    }
}
```

Código JSON 2: Obter o descritor com o ID = 8612

```
/**
* Resposta ao envio para: http://exemplo/api/images
*
* Comandos de Teste:
* curl -X POST -H "Accept: application/json" --form image=@image.jpg --form type="i"
    http://exemplo/api/images
*/

{
  "status":"success",
  "token":"1877d8b12054afddfeb5e808b97208ebd079b7cc065d5ae8f514981e2939d832",
  "job_id":"d7a992a37bd5fe6ef2e70b67,90faf1963ecb0392ec1f5425"
}
```

Código JSON 3: Resposta ao envio de uma imagem ou descritor

```
/**
* Resposta ao envio para: http://exemplo/api/train
*
* Comandos de Teste:
* curl -X POST -H "Accept: application/json" --form image=@image.jpg --form data="Uma
    String pode ser um elemento JSON" http://exemplo/api/train
*/

{
  "status": "success",
  "id": 9600
}
```

Código JSON 4: Resposta a “train”

Código JSON da API

```
/**
* Resposta ao envio para: http://exemplo/api/requests
*
* Comandos de Teste:
* echo '{"token":"1877d8b12054afddfeb5e808b97208ebd079b7cc065d5ae8f514981e2939d832","
    job_id":"d7a992a37bd5fe6ef2e70b67,90faf1963ecb0392ec1f5425"}' > api.json
* curl -X POST --data-binary @api.json -H "Content-Type: application/json" http://
    exemplo/api/requests
*/

{
    "status": "working",
    "movies": [
        {
            "movie": {
                "titles": {"original": "Beasts of the Southern Wild",
                           "pt": "Bestas do Sul Selvagem"},
                "year": 2012,
                "poster": "http://sm2.imgs.sapo.pt/mb/C/K/J/X3dxWFpXe56vr6eM5OOmvit8_.
                    jpg",
                "trailer": "http://videos.sapo.pt/22Hz2sS8U5JnT8udIFJh/mov/1",
                "duration": "93 m",
                "genre": "Drama", "indicated_age": "M12",
                "country": "Estados Unidos", "user_review": 3,
                "synopsis": "<p>Numa comunidade esquecida mas desafiante de uma zona
                    pantanosa ...",
                "production": ["Benh Zeitlin"],
                "actors": ["Dwight Henry", "Levy Easterly", "Lowell Landes",
                           "Quvenzhané Wallis"]
            },
            "hits": 4
        },
        {
            "movie": {
                "titles": {"original": "A Good Day to Die Hard",
                           "pt": "Die Hard - Nunca é bom dia para morrer"},
                "year": 2013,
                "poster": "http://sm2.imgs.sapo.pt/mb/0/d/e/l1T2C8jh4dV8j66Yvl1AAr-s_.
                    jpg",
                "trailer": "http://videos.sapo.pt/Wgrmo5NzgomSt4N60UME/mov/1",
                "duration": "98 m",
                "genre": "Ação/Aventura", "indicated_age": "M12",
                "country": "Estados Unidos", "user_review": 3,
                "synopsis": "<p>Quando Jack, o filho de John McClane, se mete ...",
                "production": ["John Moore"],
                "actors": ["Amaury Nolasco", "Bruce Willis", "Cole Hauser",
                           "Jai Courtney", "Mary Elizabeth Winstead",
                           "Patrick Stewart"]
            },
            "hits": 4
        }
    ]
}
```

Código JSON 5: Resposta a "requests"