UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE ENGENHARIA GEOGRÁFICA, GEOFÍSICA E DA ENERGIA

Aplicação Móvel para Exploração de Monumentos Históricos

João David dos Santos Barroso Rodrigues

Mestrado em Engenharia Geoespacial

Trabalho de Projeto orientado por:
Prof. Doutor João Catalão Fernandes e Prof. Doutora Ana Paula Afonso

2019

Agradecimentos

Para a concretização deste projeto, e deste ciclo de estudos, recebi sempre algum tipo de contributo, implicitamente ou explicitamente.

Deste modo, tenho de começar por fazer um agradecimento especial ao que foi para mim a maior fonte de apoio que recebi ao longo destes anos e durante o desenvolvimento deste projeto em particular: os meus pais. Portanto, deixo aqui registado o meu grande obrigado, a vocês, por todo o investimento e dedicação que sempre tiveram por mim. Obrigado.

Depois gostaria de agradecer aos meus orientadores, professor Doutor João Catalão Fernandes e professora Doutora Ana Paula Afonso, por me terem guiado ao longo deste projeto, ajudando-me a estabelecer o percurso a seguir para a concretização deste projeto.

Gostaria também de agradecer a todos os professores de Engenharia Geoespacial, pelos conhecimentos transmitidos, e o apoio oferecido, sempre que este fosse necessário.

E finalmente, agradeço a todos os meus amigos e colegas pela companhia e convívio, pelos bons momentos e apoio dado ao longo deste percurso.

Resumo

O presente projeto consiste na criação de uma aplicação móvel com a capacidade de reconhecer objetos com características suficientes e distintas para o reconhecimento, em particular, entidades notáveis das cidades.

Esta aplicação, fornece ao utilizador a capacidade de aceder a informação descritiva associada a determinado objeto de interesse. O utilizador coloca o dispositivo móvel em linha com o objeto de interesse, prime o botão de captura de imagem, e momentos depois, surge a descrição do mesmo. O funcionamento de toda a aplicação está assente em algoritmos de reconhecimento de objetos que a partir de dados contidos na base de dados, a qual contém informação relativa aos objetos, tem a capacidade de relacionar o objeto de interesse com o objeto contido na base de dados e enviar a sua descrição ao utilizador.

Foi também desenvolvida uma aplicação Web com funcionalidades que permitem ao utilizador visualizar todos os objetos de interesse representados num mapa por marcadores e adicionar ou editar informação relativa a cada um destes.

A aplicação foi testada em vários monumentos na cidade de Lisboa, incluindo estátuas no Estádio Universitário de Lisboa, estátua do professor doutor José Pinto Peixoto, no campus da FCUL, busto de João Gonçalves Zarco da Câmara, fachada do Arco da Rua Augusta, entre outros.

Palavras-chave: Aplicação Móvel; Aplicação *Web*; SIFT; Funções Hash; PostgreSQL.

Abstract

The present project consists in the creation of a mobile application with the capacity to recognize objects with sufficient and distinctive characteristics for recognition, in particular, remarkable entities of the cities.

This application, provides the user with the ability to access the descriptive information associated to an object. The user align the mobile device with the object of interest, presses the image capture button, and moments later, the description of the object is presented. The functionality of the entire application is based on object recognition algorithms, that from data contained in the database, which contains information of the objects, has the ability to relate the object of interest to the object contained in the database and send the description of this object to the user.

It was also developed a Web application with features that allow the user to view all objects represented in a map by markers and add or edit object information.

The application was tested in several monuments in the city of Lisbon, including statues at the University Stadium of Lisbon, statue of the professor Dr. José Pinto Peixoto, on the campus of FCUL, bust of João Gonçalves Zarco da Câmara, facade of the Arch of Rua Augusta, among others.

Key-words: Mobile Application; *Web* Application; SIFT; Hash Functions; PostgreSQL.

Índice

Lista de Figuras

Lista de tabelas

Lista de acrónimos e siglas

API : Application Programming Interface

CSS : Cascading Style Sheets

FCUL : Faculdade de Ciências da Universidade de Lisboa

HTML : Hyper Text Markup Language

HTTP : Hypertext Transfer Protocol

JAX-RS : JAVA API for RESTful Web Services

JSON : JavaScript Object Notation

JSP : Java Server pages

MIME : Multipurpose Internet Mail Extensions

REST : Representational State Transfer

SDK : Software Development Kit

SIFT : Scale-Invariant Feature Transform

SOAP : Simple Object Access Protocol

URI : Uniform Resource Identifier

WSDL : Web Services Description Language

XML : Extensible Markup Language

Capítulo 1 – Introdução

Neste capítulo é apresentada a motivação, os objetivos e a metodologia usada no desenvolvimento do projeto. São também apresentados trabalhos existentes relacionados com este projeto.

1.1 Motivação

Com o significativo aumento do número de turistas a visitar Lisboa e outras cidades históricas de Portugal, surgiu a ideia de desenvolver um sistema colaborativo de disponibilização e partilha de informação sobre monumentos em cidades baseado numa aplicação móvel e numa aplicação Web.

O que também motivou a realização deste projeto foi o facto de no primeiro semestre do mestrado ter sido exposto ao processamento digital de imagem, na disciplina do mesmo nome. Nesta unidade curricular desenvolvi interesse sobre a capacidade de manipular imagens digitais e extrair informação útil acerca das mesmas, aplicando vários tipos de algoritmos. Deste modo, propus-me a realizar este projeto com o intuito de aprofundar a temática.

Desta forma, surgiu a ideia de desenvolver uma aplicação móvel com a premissa de identificar um objeto numa imagem adquirida por um dispositivo móvel recorrendo a algoritmos de reconhecimento de objetos.

Outros aspetos importantes que me levaram a envolver no tópico de visão de computador, foi a minha vontade em trabalhar com tecnologias para as quais não estava familiarizado, como os serviços *Web*, bases de dados geográficas e o sistema *Android*.

1.2 Objetivos

O principal objetivo do projeto consiste no desenvolvimento de uma aplicação móvel com capacidade para identificar determinado objeto e apresentar uma descrição do mesmo. Neste projeto, os objetos a identificar são estátuas e fachadas de edifícios. Esta aplicação é caracterizada por três funcionalidades principais:

1. Capturar uma fotografia do objeto;
2. Reconhecimento do objeto;
3. Exibir a descrição do objeto.

A aplicação terá de registar a imagem capturada e as coordenadas do utilizador, e enviar estas para um servidor Web que responde com a descrição do objeto.

O segundo objetivo consiste em desenvolver uma aplicação web que providencie ao utilizador a capacidade de visualizar a localização dos objetos e de acrescentar/alterar informação relativa aos mesmos.

1.3 Trabalho Relacionado

Existem várias ferramentas que permitem o reconhecimento de objetos disponíveis para o público. As propriedades destas consistem, em parte, na utilização de marcadores, os quais podem ser imagens 2D - como a capa duma revista, ou um objeto - como uma mesa de café. Estes marcadores são usados para associar aos mesmos: animações, imagens, objetos 3D, sons, notificações, entre outros. Estas ferramentas oferecem também a capacidade de reconhecimento de superfícies planas, como: mesas, chão e paredes. Isto permite que o dispositivo móvel tenha compreensão do meio circundante. As ferramentas mais populares são: Wikitude [21], ARKit [2] e ARCore [7].

Wikitude é um kit de desenvolvimento de software (SDK) de realidade aumentada. Este kit possibilita o reconhecimento de marcadores 3D. Estes marcadores são previamente mapeados e enviados para o Wikitude Studio, onde o utilizador cria a sua experiência de realidade aumentada. Possibilita também o reconhecimento de imagens e o reconhecimento do ambiente. Este último, permite ver a extensão do conteúdo digital do objeto reconhecido, quando este está fora do campo de visão. Isto é possível pela capacidade desta tecnologia em perceber a localização e orientação do dispositivo, com base na cena.

O ARKit, é um SDK de realidade aumentada desenvolvido pela Apple que permite o reconhecimento de imagens e objetos 3D. Sobre estes, é também possível associar animações, vídeos, imagens, objetos 3D, etc. Contudo, a principal característica deste SDK é o reconhecimento do ambiente e o rastreamento de objetos digitais. Esta característica permite ao utilizador, num ambiente *indoor*, adicionar um objeto digital no ambiente, e mover-se em torno deste, observando todas as características do mesmo, sem que o objeto altere a sua localização e orientação em relação à cena.

O ARCore é a plataforma da Google para criar experiências de realidade aumentada. Este, como os anteriores, permite ao dispositivo móvel a compreensão do ambiente circundante e interagir com informação digital contida na cena. O ARCore usa 3 funcionalidades principais para integrar conteúdo virtual com o mundo real:

- Rastreamento de movimento: este permite ao dispositivo móvel compreender e rastrear a sua posição relativa ao mundo.

- Compreensão do ambiente: permite ao dispositivo detetar o tamanho e a localização de todos os tipos de superfícies.

- Estimação da luminosidade: esta funcionalidade permite ao dispositivo estimar as condições atuais da iluminação do ambiente.

No contexto deste projeto, apenas tem interesse a capacidade de reconhecer objetos, com o intuito de lhes associar a informação descritiva relacionada. Para isto foram aplicados algoritmos específicos para o efeito, sem fazer uso de nenhuma das ferramentas aqui discutidas.

1.4 Metodologia de Desenvolvimento

A aplicação desenvolvida tem uma estrutura modular organizada de acordo com o equipamento e com as funções primitivas a implementar:

- O primeiro módulo, implementado no dispositivo móvel, irá realizar a captura da imagem do objeto de interesse, registar a localização do utilizador e enviar, ambos a imagem e as coordenadas para o servidor.

- O segundo módulo, implementado no servidor, consiste no desenvolvimento de operações para o reconhecimento do objeto, após a receção da localização do dispositivo e da imagem capturada. Primeiramente, será feita uma seleção de todos os objetos candidatos contidos na base de dados. Isto é, apenas serão considerados os objetos que se encontram num raio inferior a 50 metros da localização do utilizador. Depois, através de algoritmos de reconhecimento de objetos, será determinado o objeto que mais se assemelha ao objeto de interesse e no final, é enviada para a aplicação móvel informação descritiva, correspondente ao objeto candidato.

- O terceiro módulo, implementado no dispositivo móvel, consiste neste disponibilizar a informação descritiva relativa ao monumento. Caso o objeto de interesse não conste na base de dados, o mesmo será automaticamente adicionado à base de dados, e o utilizador receberá uma notificação que o informa que o objeto de interesse não existe.

1.5 Organização do Documento

Este documento está divido em 4 capítulos:

Capítulo 1: neste capítulo é feita uma discussão das razões que levaram à criação deste projeto, quais os principais objetivos e as etapas para a sua concretização e são também apresentados projetos relacionados com este.

Capítulo 2: neste capítulo são apresentados os conceitos teóricos relativos aos algoritmos utilizados neste projeto.

Capítulo 3: neste capítulo é discutido em detalhe todo o processo de construção do sistema. São apresentadas as ferramentas e tecnologias utilizadas, os requisitos do sistema, o modelo de

dados, a arquitetura do sistema, a criação do serviço web, a criação da aplicação móvel e a criação da aplicação Web.

Capítulo 4: no último capítulo são discutidas as conclusões relativas ao projeto, problemas e dificuldades que surgiram no desenvolvimento do mesmo e trabalho futuro.

Capítulo 2 – Enquadramento Teórico

Neste capítulo são apresentados os conceitos teóricos subjacentes aos algoritmos utilizados no projeto e tecnologias utilizadas. Na secção 2.1 é abordado o que são funções *hash* percetuais, de que forma são utilizadas no projeto, e o algoritmo usado para extrair o valor *hash* da imagem. Na secção 2.2 é discutido a forma como são extraídas características de um objeto numa imagem segundo o algoritmo SIFT, na secção 2.3 são discutidos os tipos de serviços web e na secção 2.4 é apresentada a definição da API JAX-RS.

2.1 Funções Hash

Funções *hash*, são funções que extraem uma *string* de bits de comprimento fixo a partir de um input (ficheiro de computador ou imagem) de qualquer comprimento [5]. São usadas, largamente, para indexar conteúdos multimédia em bases de dados, gerar assinaturas digitais, autenticar *passwords*, entre outros.

As funções *hash* podem ser categorizadas em **funções *hash* sem-chave** e **funções *hash* com-chave**. Uma função *hash* sem-chave *H* gera um valor *hash h* a partir de um input arbitrário *x* tal que: $h = H(x)$. Uma função *hash* com-chave gera um valor *hash h* a partir de um input arbitrário *x* e de uma chave secreta *k* tal que: $h = H(x,k)$ [22].

Neste projeto são utilizadas funções *hash* sem-chave, pois a encriptação do input não oferece nenhuma vantagem prática no contexto deste projeto.

De acordo com [22] as funções sem-chave têm de respeitar no mínimo 2 propriedades:

1. **Compressão:** *H* mapeia um input *x* de comprimento bit finito e arbitrário, para um output $H(x)$ de comprimento bit finito *n*.
2. **Facilidade de computação:** dado *H* e um input *x*, $H(x)$ é fácil de computar.

Neste projeto o objetivo é usar funções *hash* que criem um identificador (valor *hash*) associado a uma dada imagem. Para isto são usadas **funções *hash* percetuais** que geram um identificador relativo ao conteúdo da imagem. Ao contrário de outras funções *hash* (que, para um determinado input *x* geram valores *hash* completamente distintos pela simples alteração de 1 bit em *x*), funções *hash* percetuais são concebidas para que imagens com conteúdo semelhante

produzam um valor *hash* semelhante. Este valor *hash* vai ser usado para, numa primeira abordagem, organizar os objetos candidatos na base de dados por grau de semelhança.

Existem várias formas de calcular funções *hash* percetuais, tal como é demonstrado em [22]. Para este projeto foi utilizado o algoritmo baseado no valor médio de blocos, pela sua simplicidade e rápida computação. Este algoritmo está dividido em 4 passos:

1. A imagem input *I* é convertida para uma escala de cinza e é reduzida para dimensões predefinidas;

2. Sendo *N* o número de bits do valor *hash* final, a imagem *I* é dividida em *N* blocos (que não se intersetam) $I_1, I_2, \dots, I_N$;

3. Para cada bloco $\{I_1, I_2, \dots, I_N\}$ é calculada a média dos valores de intensidade dos píxeis $\{M_1, M_2, \dots, M_N\}$. Depois, é determinado o valor da mediana M_d da sequência dos valores da média de cada bloco M_i;

4. A sequência M_i é normalizada em formato binário, onde o valor *hash h* é dado por:

$$h(i) = \begin{cases} 0, M_i < M_d \\ 1, M_i \geq M_d \end{cases}$$

Cada imagem vai ter associada um valor *hash* único na base de dados. O conceito é: quando uma nova imagem for enviada para o servidor, a primeira operação a ocorrer é o cálculo do valor *hash* desta; este valor é depois comparado com os valores *hash* na base de dados, por forma a encontrar a imagem mais semelhante. A comparação entre duas imagens é feita pelo cálculo da distância de *Hamming [22]*

A distância de *Hamming*, é definida por:

$$\Delta(x, y) \coloneqq \sum_{x_i \neq y_i} 1 \, , i = 1, \dots, n$$

onde *x* e *y* correspondem a *strings* com o mesmo comprimento, pertencentes a um alfabeto *A* de tamanho finito [22]. Isto significa que sempre que um caractere ou bit em *x* for diferente do caractere ou bit em *y*, na mesma posição, a distância de *Hamming* é incrementada por 1.

A tabela 2.1 exemplifica a determinação da distância de *Hamming*. Como é possível ver, valores *hash* idênticos vão naturalmente resultar numa distância igual a 0, e valores *hash* diferentes, resultam em distâncias superiores a 0. Portanto, quanto mais semelhantes forem as imagens, menor será a distância de *Hamming*.

Tabela 2.1 - Exemplo da determinação da distância de Hamming

Valor hash *x*	Valor hash *y*	Distância de *Hamming*
010011	010011	0
100101	10011**1**	1
010110	**10**01**01**	4

As funções *hash* percetuais são eficazes a comparar o quanto duas imagens são semelhantes, mas não têm qualquer utilidade no que diz respeito à identificação de objetos idênticos em imagens diferentes, para isto é necessário recorrer a algoritmos especificamente desenvolvidos para o efeito.

2.2 Scale Invariant Feature Transform (SIFT)

O algoritmo SIFT [11], desenvolvido por David G. Lowe, é um método de extração de características distintas dum objeto contido numa imagem, as quais são usadas para identificar o mesmo objeto noutras imagens. Estas características são invariantes à escala da imagem e rotação, e apresentam uma correspondência robusta para distorções afins, mudanças do ponto de vista 3D, adição de ruído e alterações de luminosidade.

Este é um dos mais populares algoritmos de reconhecimento de objetos. Até ao surgimento de algoritmos que envolvem inteligência artificial, os algoritmos propostos após o SIFT, tinham por base os conceitos desenvolvidos por David G. Lowe, no que diz respeito a este algoritmo.

O cálculo do SIFT é realizado em 4 fases:

1. **Deteção de extremos no espaço-escala:** nesta fase, potenciais pontos de interesse são identificados pela aplicação duma função diferença-de-Gaussianas.

2. **Localização de pontos chave:** a cada localização de um ponto candidato, é ajustado um modelo detalhado para determinar a localização e escala.

3. **Atribuição de orientação:** uma ou mais orientações são atribuídas a cada localização de um ponto chave baseado em direções de gradiente da imagem local. Todas as operações futuras são feitas em dados de imagem que foram transformados

relativamente à orientação, escala e localização atribuída a cada característica, assegurando assim, invariância a estas transformações.

4. **Determinação dos descritores dos pontos chave:** é determinado um vetor descritor a partir dos píxeis em redor do ponto chave.

2.2.1 Deteção de extremos no espaço-escala

Numa primeira abordagem, o propósito é determinar localizações e escalas dos pontos chave que possam ser usados para identificar um determinado objeto em diferentes pontos de vista. Desta forma, começa-se por gerar o espaço-escala. O espaço-escala corresponde ao conjunto de imagens resultantes da redução da escala da imagem original – várias vezes – separadas por um fator *k* (Figura 2.1). Este, é definido como uma função $L(x,y,\sigma)$, a qual é gerada pela convolução do filtro Gaussiano $G(x,y,\sigma)$, a uma escala σ, com uma imagem $I(x,y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

onde * é a operação de convolução em *x* e *y*, e

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}.$$

Para detetar a localização dos pontos chave de forma eficiente é feita a diferença entre duas escalas separadas pelo fator *k*:

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y)$$

$$L(x, y, k\sigma) - L(x, y, \sigma).$$

Em adição, a função diferença-de-gaussianos, *D*, proporciona uma aproximação equivalente ao filtro Laplaciano Gaussiano normalizado em relação à escala, $\sigma^2\nabla^2 G$ (Este filtro é usado, essencialmente, para encontrar arestas numa imagem).

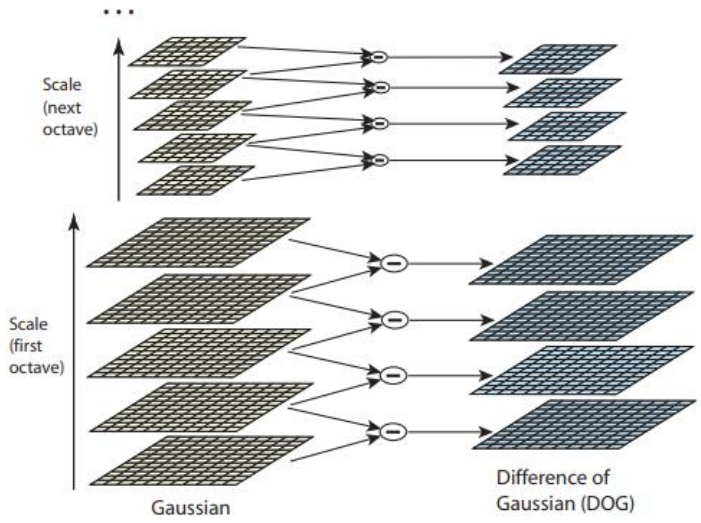

Figura 2.1 – Espaço-escala / diferença de gaussianas [11].

A relação entre a função *D* e $\sigma^2\nabla^2 G$ pode ser entendida pela equação do calor, tal que:

$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G.$$

A partir desta equação, verifica-se que $\nabla^2 G$ pode ser determinado pela derivada de *G* em relação a σ, $\partial\, G/\partial\, \sigma$, usando a diferença entre escalas vizinhas em $k\sigma$ e σ:

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

que resulta em:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G$$

Isto demonstra que quando a função *D* tem escalas a variar por um fator constante, já incorpora o σ^2 necessário para tornar $\nabla^2 G$ invariante à escala. Contundo, esta operação, adiciona um fator (*k*-1) na equação. Mas, como este fator é constante sobre todas as escalas, não vai influenciar a localização de extremos.

Segundo o autor, uma forma eficiente de gerar $D(\mathbf{x}, y, \sigma)$ consiste em:

1. Incrementalmente convolver a imagem original com filtros Gaussianos que variam segundo $k\sigma$. O recomendado é incrementar 5 vezes, ou seja, até 5 imagens. Estas 5 imagens perfazem um *octave*.

2. Em cada *octave*, as imagens adjacentes são subtraídas para obter a função diferença-de-Gaussianos, *D*.

3. A imagem Gaussiana que tem o dobro do valor inicial de σ é reamostrada para metade do tamanho e são repetidos os passos 1 e 2.

O autor sugere criar um espaço-escala com 4 *octaves*, cada um composto por 5 imagens Gaussianas, e um fator *k* igual a $\sqrt{2}$ com σ igual a 1.6.

2.2.2 Localização de pontos chave

Para localizar os máximos e os mínimos de $D(\mathbf{x},y,\sigma)$, cada ponto é comparado com os seus 8 vizinhos na imagem atual, e os 9 vizinhos da imagem em cima e em baixo (Figura 2.2). Se o ponto em questão for maior ou menor que todos os seus vizinhos, então é considerado um extremo e é selecionado como um ponto candidato.

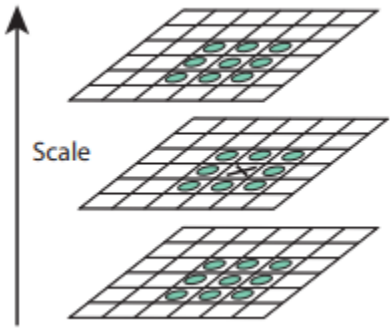

Figura 2.2 - Deteção dos máximos e mínimos [11].

Para obter localizações mais exatas dos pontos candidatos, os extremos são interpolados a partir dos pontos em torno do ponto candidato. Deste modo é determinada a série de Taylor de $D(x,y,\sigma)$, com origem neste ponto:

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x$$

onde *D* e as suas derivadas são calculadas no ponto candidato e $P = (x, y, \sigma)^T$ é o desvio deste ponto. A localização do extremo, $\hat{P}$, é determinada a partir do cálculo da derivada desta função com respeito a *x* e igualando-a a 0, o que resulta em:

$$\hat{P} = \frac{-\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x}.$$

Se o desvio $\hat{\hat{P}}$ é maior que 0.5 em qualquer dimensão, então o extremo em questão está mais próximo de um outro ponto candidato. Neste caso, muda-se o mesmo, e faz-se uma nova interpolação sobre este novo ponto. O desvio final $\hat{P}$ é adicionado à localização do seu ponto candidato por forma a obter a estimativa da localização do extremo.

Para eliminar extremos com baixo contraste, calcula-se a função *D* no extremo, $D(\hat{P})$, ou seja:

$$D(\hat{P}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \hat{P}.$$

Todos os extremos com valor de $D(\hat{P})$ menor do que 0.03 são rejeitados.

Para melhorar os resultados, não basta descartar os pontos com baixo contraste. Os pontos de maior interesse são cantos. Isto implica que, pontos que se encontrem sobre arestas não têm qualquer utilidade, logo, são também descartados. Para isto, são calculadas as curvaturas principais sobre o ponto chave. Segundo [10], existem 3 situações possíveis:

1. Se ambas as curvaturas são pequenas, a imagem em torno do ponto chave é plana (ambos os gradientes, na direção de *x* e *y* são pequenos).

2. Se uma curvatura for grande e a outra pequena, estamos na presença de uma aresta (Um gradiente vai ser pequeno ao longo da aresta e o outro, perpendicular à aresta, será grande).

3. Se ambas as curvaturas forem grandes, temos um canto (ambos os gradientes serão elevados).

Estas curvaturas, podem ser calculadas a partir duma matriz Hessiana, H, 2x2:

$$H = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}$$

Os autovalores de H são proporcionais às curvaturas principais de *D*. Nesta situação, não é necessário calcular explicitamente os autovalores, pois apenas interessa o rácio entre eles. Admitindo que α é o autovalor com a maior magnitude e β o autovalor com a menor magnitude e sabendo que:

Seja A uma matriz quadrada com autovalores $\lambda_i = 1,2, \dots, n$, tem-se que:

$$Diag(A) = \sum_{i=1}^{n} \lambda_i$$

e

$$Det(A) = \prod_{i=1}^{n} \lambda_i,$$

temos que:

$$Diag(H) = \frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2} = \alpha + \beta,$$

$$Det(H) = \frac{\partial^2 D}{\partial x^2}\frac{\partial^2 D}{\partial y^2} - \left(\frac{\partial^2 D}{\partial x \partial y}\right)^2 = \alpha\beta.$$

Admitindo *r* como sendo o rácio entre o autovalor com maior magnitude e β o autovalor com menor magnitude, tal que $\alpha = r\beta$. Então:

$$\frac{Diag(H)^2}{Det(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

a qual depende apenas do rácio entre os autovalores. A quantidade $(r+1)^2/r$ é mínima quando os autovalores são iguais e aumenta com *r*. Assim, para verificar que o rácio de curvaturas principais está abaixo de um certo limiar, *r*, é apenas necessário verificar

$$\frac{Diag(H)^2}{Det(H)} < \frac{(r+1)^2}{r}.$$

2.2.3 Atribuição de orientação

Ao associar uma orientação a cada ponto chave com base nas propriedades da imagem local, o descritor do ponto chave pode ser representado relativamente a esta orientação e assim alcançar invariância em relação à rotação de imagem.

A escala do ponto chave é usada para selecionar a imagem Gaussiana, *L*, com a escala mais próxima, para que todos os cálculos sejam efetuados sobre invariância de escala. Para cada amostra

da imagem, $L(x,y)$, à escala do ponto chave, a magnitude do gradiente, $m(x,y)$, e orientação, $\theta(x,y)$, é pré-calculado usando diferenças de píxeis:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

É criado um histograma a partir das orientações dos gradientes dos píxeis sobre uma região em redor do ponto chave. De modo a que os píxeis mais próximos do ponto chave tenham mais relevância é usada uma janela Gaussiana circular, com um valor de σ igual a 1.5 vezes a escala deste ponto chave. O histograma tem 36 classes que cobrem 360 graus de orientações. Cada orientação determinada é adicionada à sua classe correspondente, de acordo com os graus da orientação. A magnitude calculada, é incrementada à quantidade da classe. A classe com a quantidade mais elevada, corresponde a uma orientação dominante, logo é atribuído ao ponto chave esta orientação. Se a quantidade de outra classe for superior a 80% da classe com a quantidade mais elevada, é gerado um novo ponto chave com as mesmas coordenadas e escala que o ponto chave em análise, mas, com a orientação desta classe. Finalmente, o pico da classe "dominante" é interpolado, fixando uma parábola à volta do ponto máximo. Isto é feito para obter melhores resultados.

2.2.4 Determinação dos descritores dos pontos chave

Após esta série de operações, o resultado final é um conjunto de pontos chave invariantes à escala, localização e orientação. Estes pontos, são descritos por 4 parâmetros: as coordenadas (x,y) dos pontos, a escala σ e a orientação θ. O próximo passo é calcular um vetor descritor da região da imagem em redor dos pontos. Para isto, o procedimento passa por primeiro dispor uma janela 16x16 píxeis sobre a imagem Gaussiana de escala igual à do ponto chave, no centro do mesmo. Por forma a ter invariância em relação à orientação, as coordenadas do descritor e as orientações dos gradientes são rodadas relativamente à orientação do ponto chave. Novamente, é usada uma janela gaussiana com σ igual a 0.5 vezes a largura da janela do descritor, para dar mais peso aos gradientes mais próximos do ponto chave. A janela do descritor é divida em janelas 4 x 4, e, assumindo que os gradientes de cada píxel já foram calculados, como na secção anterior, são criados 16 histogramas (um histograma para cada janela 4x4, Figura 2.3, figura à esquerda), onde cada um destes está dividido em 8 classes de orientações (0º-45º,46º-90º, …, 316-360º), em vez de 36 (Figura 2.3, imagem à direita). Por fim, é criado um vetor, cujos os elementos correspondem a todas estas direções. Ou seja, no final, tem-se um vetor descritor com 4x4x8=128 elementos.

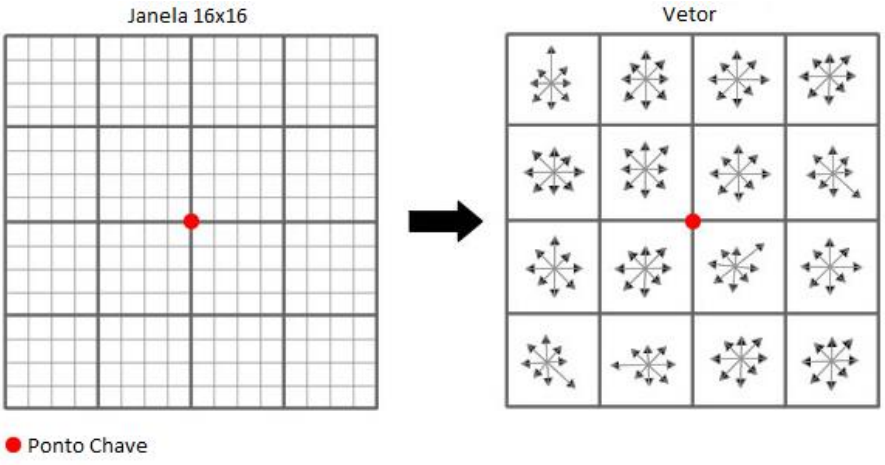

Figura 2.3 - Determinação do vetor descritor [1].

Para reduzir os efeitos da mudança de iluminação, o vetor descritor é normalizado para um comprimento de uma unidade. Outros efeitos problemáticos estão relacionados com alterações de iluminação não-lineares, que causam uma grande mudança em magnitudes relativas para alguns gradientes. Nesta medida, a influência de grandes magnitudes de gradientes, é reduzida impondo um limiar ao vetor descritor unitário, garantido que nenhum valor é superior a 0.2. Feito isto, o vetor é novamente normalizado.

2.3 Serviços Web

Um serviço Web é um software que permite a comunicação, segundo o protocolo *HyperText Transfer Protocol* (HTTP), entre diferentes aplicações, independentemente da arquitetura e tecnologia utilizadas por estas. Este, tem uma interface descrita segundo o formato *Web Services Decription Language* (*WSDL*) que descreve como outros sistemas podem interagir com o serviço Web.

Existem dois tipos de tecnologias de criação de serviços Web:

1. SOAP (*Simple Object Access Protocol*) - SOAP é um protocolo que fornece um envelope para enviar mensagens ao serviço web e à aplicação do cliente. Estas mensagens são documentos XML, e são transmitidas por HTTP. A grande vantagem do SOAP é este definir a sua própria segurança e tem como desvantagem o facto de apenas usar o formato XML [17].

2. REST (*Representational State Transfer*) – REST é um estilo de arquitetura baseada em padrões web, que usa o protocolo HTTP para comunicação de dados. Opera com base em recursos onde cada componente é um recurso, e um recurso é acedido por uma

interface comum através de métodos padrão HTTP (POST, GET, PUT, DELETE). Na arquitetura REST, um servidor REST simplesmente fornece acesso a recursos e o cliente REST acede e apresenta os recursos. Cada recurso é identificado por URIs (*Uniform Resource Identifier*). A grande vantagem do REST, é este aceitar vários formatos, como: JSON e XML [20].

2.4 API JAX-RS

JAX-RS é uma API baseada na linguagem Java e uma especificação para fornecer suporte a serviços web REST [19] que facilita a criação dos mesmos. Esta API usa anotações introduzidas no Java SE 5, com o intuito de simplificar o desenvolvimento de serviços web baseados em Java. Na tabela em baixo, estão definidas as anotações usadas neste projeto juntamente com a descrição de cada uma delas [19]. A API JAX-RS foi implementada pela *framework* jersey [24].

Tabela 2.2 - Recursos REST.

Anotação	Descrição
@Path	Caminho relativo do recurso
@GET	Método HTTP GET. Usado para buscar um recurso
@PUT	Método HTTP PUT. Usado para atualizar um recurso
@POST	Método HTTP POST. Usado para criar um recurso
@Produces	Define o tipo de MIME gerado pelo serviço web
@Consumes	Define o tipo de MIME consumido pelo serviço web
@PathParam	Associa o parâmetro passado ao método a um valor no caminho

Capítulo 3 - Trabalho Realizado

Neste capítulo são mencionadas as ferramentas e tecnologias utilizadas, a arquitetura do sistema, o processo de criação e funcionamento do serviço Web e o desenvolvimento da aplicação móvel e da aplicação Web. Na primeira secção são descritas as ferramentas e tecnologias utilizadas no projeto. Na secção 3.2 são descritos os requisitos do sistema, na secção 3.3 são apresentados os componentes que compõem o sistema e as relações entre eles, e o modo de operacionalidade deste. Na secção 3.4 é apresentada a base de dados. Na secção 3.6 é descrito o desenvolvimento do serviço Web, os recursos criados e modo de funcionamento destes. Na secção 3.7 e 3.8 são apresentadas a aplicação móvel e a aplicação Web, respetivamente, e exemplos de utilização. Por fim, na última secção, é feita uma discussão dos resultados dos testes feitos a todo o sistema.

3.1 Ferramentas e Tecnologias Utilizadas

Todo o trabalho foi desenvolvido num computador portátil com sistema operativo Windows 10 Home x64, um processador Intel Core i5 @ 2.60 GHz e 6 GB de RAM. O servidor corre o sistema operativo Windows 7 Professional x86, tem um processador intel Core i3 @ 3.30 GHz e 4 GB de RAM. A aplicação móvel foi testada num *smartphone* com o sistema operativo *Android*, versão: 7.1.1 Nougat, com 2 GB de RAM, com um processador Qualcomm Snapdrago 425 Quad Core (MSM8917) @1,4 GHz e uma câmara traseira de 8 MP.

Foi utilizado para o projeto a base de dados PostgreSQL [15], com a extensão PostGIS [14]. A razão de se optar por esta base de dados, deve-se às funcionalidades de que dispõe, que permitem fazer interrogações a objetos geográficos. Para gerir a base de dados foi utilizada a plataforma pgAdmin 4 [25].

A opção de desenvolvimento do projeto em Java implicou que o serviço Web fosse também desenvolvido na mesma linguagem. Desta forma, o servidor web de eleição foi o Apache Tomcat 9.0, um software de código livre, desenvolvido pela *Apache Software Foundation*, que consiste num *container* de *servelets e javaSerever pages* (JSP), que implementa as tecnologias *java Expression Language* e *java WebSocket* [18]. Essencialmente, o Apache Tomcat é usado para executar aplicações Web Java.

Para facilitar o desenvolvimento do serviço web, no que diz respeito à escrita do código, foi utilizado o IDE Eclipse Oxygen 3 (pacote Java EE) [4]. Para a criação do serviço web foi utilizada a API JAX-RS. Esta API foi projetada para facilitar o desenvolvimento de aplicações que usam a arquitetura REST [13].

Em relação à aplicação móvel, foi utilizado o IDE do *Android*: Android Studio. Este IDE foi criado especificamente para desenvolver aplicações *Android* e tem todo um conjunto de

ferramentas que facilitam e tornam mais eficiente todo o processo de criação da aplicação, desde escrever o código, até testar o funcionamento da aplicação.

A aplicação web foi criada com recurso ao IDE Visual Studio Code [12] e foram usadas as linguagens HTML/CSS e JavaScript. Para a geração dos marcadores no mapa com base na localização dos objetos foi também utilizada a API *Maps javaScript* [9].

3.2 Requisitos do Sistema

O sistema desenvolvido deverá cumprir um conjunto de requisitos e possuir certas funcionalidades.

O primeiro requisito permite que o utilizador obtenha em tempo real informação relativa a um qualquer objeto, pelo uso de um dispositivo móvel.

O segundo requisito consiste em oferecer ao utilizador uma plataforma onde este possa visualizar os objetos existentes sobre um mapa.

O terceiro, e último requisito, passa por criar um conjunto de funcionalidades que permitam ao utilizador acrescentar/editar informação relativa a um determinado objeto.

3.3 Arquitetura do Sistema

Na Figura 3.1 está representada a arquitetura do sistema desenvolvido constituída por 4 componentes principais: aplicação móvel, serviço Web, base de dados e aplicação Web.

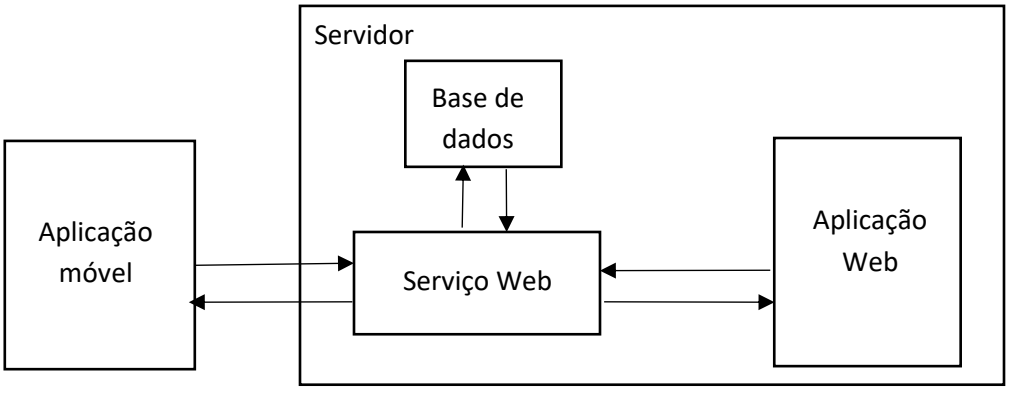

Figura 3.1 - Arquitetura do sistema

A aplicação tem início com o utilizador a adquirir a imagem de um monumento do qual pretende obter informação, a partir do seu dispositivo móvel. Para o efeito, a aplicação móvel tem uma interface que permite capturar uma imagem do monumento. Assim que o utilizador prime o botão de "captura", a imagem é registada no dispositivo móvel e são determinadas as coordenadas geográficas do dispositivo. Esta informação é depois enviada para o servidor, onde é processada pelo serviço web. O serviço Web efetua uma interrogação espacial à base de dados com o intuito de receber todos os objetos que estejam num raio inferior a 50 m das coordenadas do dispositivo móvel. Estes serão os objetos tidos como possíveis candidatos do monumento em questão. De seguida é determinado o valor *hash* da imagem adquirida (objeto de interesse), e este é comparado com os valores *hash* dos objetos candidatos. O resultado desta operação é uma lista de objetos candidatos organizados por grau de semelhança. Por fim, para cada objeto candidato são determinados os descritores SIFT e estes são comparados com os descritores do objeto de interesse. No final, o objeto candidato selecionado será o que apresentar mais descritores em comum com os descritores do objeto de interesse. Depois é enviada a descrição do objeto para aplicação no dispositivo móvel e apresentada ao utilizador numa interface própria. Caso não seja obtida nenhuma correspondência, está-se perante uma situação em que o objeto de interesse ainda não existe na base de dados. Deste modo, o objeto é adicionado à base de dados e o utilizador em vez de receber uma descrição, recebe uma notificação a indicar que não existe informação relativa ao monumento.

A aplicação Web, ao iniciar, vai comunicar com o serviço Web para obter os dados relativos aos objetos contidos na base de dados e apresentar estes num mapa, representados por um marcador. A aplicação vai conter funcionalidade que permite acrescentar e editar a informação relativa a determinado objeto. Sempre que é feita uma alteração são usados os recursos do serviço Web para atualizar a informação na base de dados.

Cada componente do sistema é explicada em mais detalhe nas próximas secções.

3.4 Base de Dados

A extensão do sistema de gestão de base de dados relacional PostgreSQL, PostGIS, adiciona funções espaciais como a distância, área, união, intersecção e tipos de dados geométricos [3], o que permite executar operações espaciais sobre informação geográfica. Neste projeto, será usada a função distância para determinar quais os objetos que se encontram a uma distância inferior ou igual a 50 m da localização do dispositivo móvel.

A base de dados é constituída apenas por uma tabela que contém informação dos objetos. Para cada objeto é guardada informação relativa aos valores *hash*, as coordenadas – latitude e

longitude – do dispositivo móvel, o nome do objeto, a sua descrição e a imagem capturada pelo dispositivo móvel codificada em formato Base64.

No módulo de administração do PosGIS, pgAdmin, é criada uma base de dados espacial e a tabela **mnt** com os campos: **id**, **nome**, **hash**, **desc**, **img**, **lat** e **lng.** Estes correspondem ao identificador único, nome, valor *hash* da imagem, descrição do objeto, valor Base64 da imagem capturada e a latitude e longitude do dispositivo móvel

Em baixo é apresentado o código SQL da criação da tabela **mnt**:

```
CREATE TABLE public.mnt
(
    id integer NOT NULL DEFAULT nextval('mnt_id_seq'::regclass),
    nome text COLLATE pg_catalog."default",
    hash text COLLATE pg_catalog."default",
    img text COLLATE pg_catalog."default",
    desc text COLLATE pg_catalog."default",
    lat double precision,
    lng double precision,
    features text COLLATE pg_catalog."default",
    CONSTRAINT mnt_pkey PRIMARY KEY (id)
);
```

A Tabela 3.1 mostra alguns objetos inseridos na tabela **mnt**.

Tabela 3.1: Exemplos de objetos existentes na tabela mnt

ID	NOME	HASH	DESC	IMG	LAT	LNG
1	Museu de Lisboa	110001000111001110 111001010011000010 011001111000100111 1111000100	O núcleo-sede do Museu de Lisboa […] Bicos e o Torreão Poente da Praça do Comércio	/9j/4AAQSk ZJRgABAQ AAAQ […] X/f4/193r/9k =	38.758539	-9.156334
2	José Pinto Peixoto	101110010100001000 100001100111111100 111010111001011100 0111100100	José Pinto Peixoto GCSE […] do ciclo global de água na atmosfera.	/9j/4DHUSIZ JRgABAQA ERT […] X/f54gs/9k=	38.757243	-9.155696
3	João I de Portugal	100101001101010111 010011100101000000 011111010100111100 0111010100	João I de Portugal (Lisboa, 11 de abril de 1357 – Lisboa, 14 de agosto de 1433), […] desenvolvimento do reino.	/8i/4DDDCL PPZJRgABA QARET […] X/4f5gs/9k=	38.758737	-9.155244

3.5 Serviço Web

O serviço Web é a componente principal do sistema. As funções deste incluem: enviar ao utilizador a informação requisitada, adicionar novo conteúdo à base de dados e fazer reconhecimento de objetos. Como já foi referido, o serviço Web foi desenvolvido em Java e segue a arquitetura REST. Para facilitar a criação do serviço foi utilizada a API JAX-RS. O serviço web consume e produz conteúdos no formato JSON.

Para este serviço foram criados 5 recursos REST definidos na Tabela 3.2. O primeiro recurso será utilizado para descarregar todos os objetos contidos na base de dados. Os dois recursos seguintes, serão utilizados para adquirir informação relativa a um único objeto, identificados por um identificador único ou nome. Por último, os métodos POST e PUT, serão responsáveis por adicionar e atualizar informação na base de dados.

Tabela 3.2 - Recursos REST

Recurso	Método	Parâmetro	Resultado
/monuments	GET		Devolve todos os objetos na base de dados.
/monuments/id/{id}	GET	**id** = identificador do objeto	Devolve um objeto com identificador igual a **id**
/monuments/name/{name}	GET	**name** = nome do objeto	Devolve um objeto com nome igual a **name**
/monuments	POST		Adiciona um novo objeto
/monuments/id/{id}	PUT	**id** = identificador do objeto	Atualiza o conteúdo do objeto de identificador igual a **id**

As próximas secções são dedicadas à explicação da construção e funcionamento de cada um dos recursos declarados na tabela anterior.

3.5.1 Obter todos os objetos

Para obter uma lista de todos os objetos é usado o recurso `/monuments`. Este, tem como objetivo extrair toda a informação contida na base de dados e disponibilizar ao cliente (aplicação móvel) em formato JSON. O recurso é criado da seguinte forma:

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public Monument[] getMonuments() {
     db=new DataBase();
```

```
      monumentsList=db.getMonuments();
     return monumentsList.toArray(new Monument[monumentsList.size()]);
 }
```

A declaração da anotação `@GET` implica que o método `getMonuments()` é responsável por enviar ao cliente um determinado recurso. No método acima, a primeira linha corresponde à declaração de uma nova instância da classe `DataBase()` em `db`. Na linha seguinte, é criada a lista `monumentList` do tipo `Monument[]`, inicializada por `db.getMonuments()`. O método `getMonuments()` da classe `DataBase()`, realiza a conexão à base de dados e executa a *query*: *'"SELECT * FROM mnt" '*, que devolve o conteúdo de todas as linhas e colunas da tabela **mnt**. O resultado desta *query* é iterado por forma a criar uma lista de objetos do tipo `Monument`. Na última declaração do método `getMonuments()`, a lista `monumentList` é enviada ao cliente, mas antes, é convertida num *Array* para ser convertida em JSON pela biblioteca *Jackson*. Abaixo está apresentado o resultado do recurso:

"http://194.117.43.210/reconhecimentodemonumentos/data/monuments"


```
[
{"desc":"José Pinto Peixoto […] água na atmosfera.",

"hash":"1011100101000010001000001100111111001110101110010111000111100100",
 "id":151,
 "img":"/9j/4AAQSkZJRgABAQAAAQAB[…]A5qnt+97DX//Z",
 "lat":38.757243,
 "lng":-9.155696,
 "name":"José Pinto Peixoto"},
{"desc":"No caso de detetar um incêndio[…]abandone a sala;",

"hash":"1100110010111001111101000111001101001000000010111000011110100011",
 "id":152,
 "img":"/9j/4AAQSkZJRg[…] Pl/T/br/9k=",
 "lat":38.7075163,
 "lng":-9.1363793,
 "name":"fcul c8"},
[…]
{"desc":"null",

"hash":"1111111111110110110101101110000011100000110000000000110000011100",
 "id":340,
 "img":"/9j/4AAQSQBPff8A[…]+2AIT/In7H/PH//Z",
 "lat":37.9468144,
 "lng":-8.8591906,
 "name":"null"}
]
```

3.5.2 Obter um objeto pelo identificador

Este recurso foi construído de modo a ser obtida informação de um único objeto identificado pelo respetivo `id`, através do recurso `/monuments/id/{id}`. Este recurso é idêntico ao anterior, com a exceção de que está a ser requisitado um elemento específico. Em baixo está representado o código relativo a este recurso.

```
@GET
@Path("id/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Monument getMonumentByID(@PathParam("id") int id) {
      db=new DataBase();
      return db.getMonumentById(id);
}
```

Novamente, trata-se de um método `GET` que acede à base de dados, procura um elemento específico, e envia a informação relativa a esse elemento ao cliente. Neste caso é declarada a anotação `@Path("id/{id}")`. Esta anotação define o caminho ao recurso. No *input* de `getMonumentByID`, a declaração da anotação `@PathParam("id")`, faz associar o valor do parâmetro do URI, `{id}`, à variável `int id`, a qual será usada para especificar o objeto na base de dados. O método, como no caso antecedente, envia também uma resposta no formato JSON.

Primeiro, a variável `db` é instanciada como um novo objeto `DataBase()`, em `db=new DataBase()`, e segundo, é enviado ao cliente - em formato JSON - o objeto `db.getMonumentById(id)`. O método `getMonumentById(id)` após conectar à base de dados, executa a *query*: *'"SELECT * FROM mnt WHERE id ="+id'*. A resposta desta *query* é usada para criar um novo objeto do tipo `Monument`, o qual é depois enviado ao cliente. Em baixo, tem-se o exemplo da resposta de uma requisição a um elemento da base de dados com `id=187`:

"http://194.117.43.210/reconhecimentodemonumentos/data/monuments/id/187"


```
{"desc":"José I (Lisboa, 6 de junho de 1714 […] do século XVIII.",

"hash":"1111101011111000111110000111100100010001001111010001100000011000",
 "id":187,
 "img":"/9j/4AAQS […] 5n+Xt/X1eP/2Q==",
 "lat":38.707506,
 "lng":-9.136485,
 "name":"José I de Portugal"}
```

3.5.3 Obter um objeto pelo nome

Este recurso é idêntico ao anterior, mas o valor do parâmetro do URI é do tipo *String* e é usado para fornecer informação de um elemento na base de dados, identificado pelo nome. O resultado em baixo, é o correspondente à requisição do elemento de nome: "João Gonçalves Zarco da Câmara".

"http://194.117.43.210/reconhecimentodemonumentos/data/monuments/name/Jo%C3%A3o%20Gon%C3%A7alves%20Zarco%20da%20C%C3%A2mara"


```
{"desc":"D. João Maria Evangelista Gonçalves Zarco da Câmara […] Alto
de São João.",

"hash":"1100101111000010010000100000111111001111110100101110001011101000",
 "id":181,
 "img":"/9j/4AAQSkZJRg […] P/AJfH/9k=",
 "lat":38.714784,
 "lng":-9.1407272,
 "name":"João Gonçalves Zarco da Câmara"}
```


3.5.4 Adicionar um objeto

Para adicionar nova informação à base de dados é utilizado o método POST no recurso `/monuments`. O método POST é apenas utilizado na aplicação móvel e envia a imagem capturada e as coordenadas geográficas do dispositivo móvel, em formato JSON, ao serviço Web, onde são efetuadas as operações para determinar os objetos candidatos, como é mostrado na Figura 3.2.

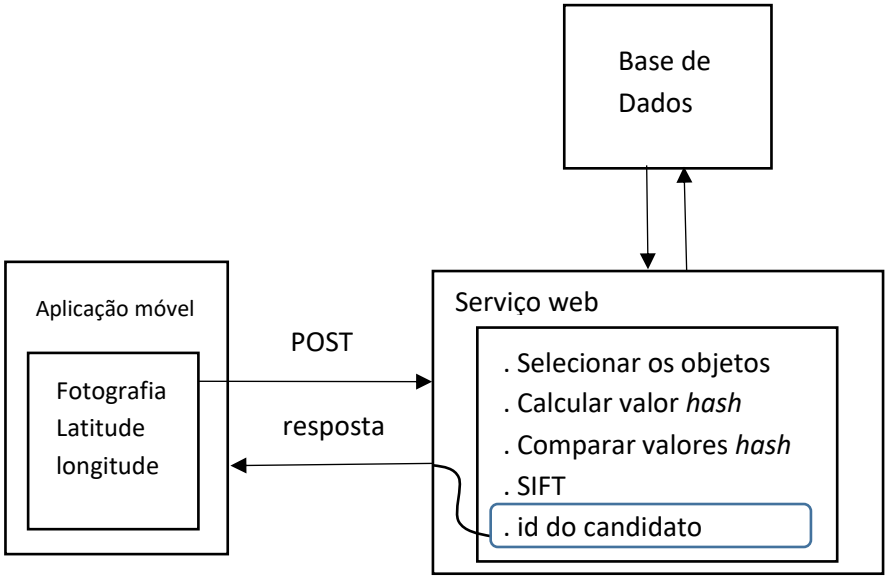

Figura 3.2: Fluxo de operações do método POST

O método POST é definido pelo seguinte bloco de código:

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Status insert(Monument mnt) {
	db=new DataBase();
	String monumentID=db.addMonument(mnt);

	return new Status("SUCCESS", monumentID);
}
```

Em primeiro lugar é efetuada a declaração da anotação @POST, seguida pela declaração do formato da mensagem que o método aceita receber – JSON, em @Consumes(MediaType.APPLICATION_JSON) e o formato da mensagem da resposta – também JSON, em @Produces(MediaType.APPLICATION_JSON).

O método `insert(Monument mnt)` realiza a criação de um novo objeto `DataBase()`, com `db=new DataBase()`, seguida pela declaração da variável `monumentID`, a qual vai ser igual à *String* resultante de `db.addMonument(mnt)`. No final é retornado objeto `new Status("SUCCESS", monumentID)`, que vai corresponder a uma

mensagem em formato JSON com dois campos. O primeiro indica que o pedido foi bem sucedido e o segundo corresponde ao identificador do objeto candidato.

Selecionar os objetos

O primeiro passo corresponde à seleção de todos os monumentos que estejam a uma distância inferior a 50 m através da execução da seguinte *query:*

```
'"SELECT * FROM mnt
WHERE   (ST_DistanceSphere(st_makepoint(lng,   lat),st_makepoint("+lng+",
"+lat+"))<50 )"'
```

As variáveis *lng* e *lat*, correspondem à longitude e latitude do dispositivo móvel atual. O resultado desta *query* é iterado por forma a criar uma lista de monumentos. Esta lista é alocada à variável *monumentList*, de tipo *List<Monument>*.

Calcular o valor *hash*

Após a lista de monumentos estar definida, o próximo passo consiste no cálculo do valor *hash* da imagem atual. Para tal, começa-se por converter a imagem numa escala de cinza. De seguida a imagem resultante é redimensionada para um tamanho fixo de 64x64 píxeis e procede-se à divisão da imagem em blocos. Dado que se optou por um valor *hash* final de 64 bits de comprimento, a imagem foi dividida em 64 blocos. Em cada um destes blocos é determinada a média dos valores de intensidade dos píxeis. O código desenvolvido para o cálculo destes valores é apresentado em baixo.

```
public  List<Double> blocks(BufferedImage img, int size){
      //Área de cada bloco
      int areaBlock=(int)(Math.sqrt(size))*(int)(Math.sqrt(size));
      //largura de cada bloco
      int blockW=(int) Math.sqrt(SIZE);
      int blockH=blockW;

      List<Double> blockMean = new ArrayList<>();

      double sum=0;
      //blocks
      for(int i=0;i<blockW;i++){
            for(int c=0;c<blockH;c++){
                  //píxeis em cada block
                  for(int x=0;x<blockW;x++){
                        for(int y=0;y<blockH;y++){
                           sum+=img.getRGB(x+i*blockW, y+c*blockH) & 0xFF;
                        }
```

```
                }
                blockMean.add(sum/areaBlock);
                sum=0;
            }
        }
        return blockMean;
}
```

O método `blocks(BufferedImage img, int size)`, determina primeiro a área dos blocos na variável `areaBlock`. De seguida, é determinada a largura e altura de cada bloco em `blockW` e `blockH`. Como a imagem *input* é quadrada, os blocos são quadrados e logo, `blockW = blockH`. A imagem é dividida em $\sqrt{size}$. $\sqrt{size}$ blocos, e cada bloco, do mesmo modo, tem $\sqrt{size}$. $\sqrt{size}$ píxeis. Na linha seguinte, é criada uma lista – `blockMean` – que vai alocar as médias de cada bloco. Para cada bloco é obtida a soma dos valores dos píxeis, e esta é depois dividida pela área do bloco, sendo no final adicionada à lista `blockMean`. Após o cálculo das médias dos blocos, prossegue-se para a determinação da mediana. O código é apresentado em baixo.

```
private  double median(List<Double> blockMean){
        List<Double> aux=new ArrayList<>(blockMean);
        Collections.sort(aux);
        int len=aux.size();
        int type=0;

        if(len%2==0) type=1;

        if(type==1){
                double n=aux.get(len/2-1);
                double n2=aux.get(len/2);
                return (n+n2)/2;
        }else{
                return aux.get((len+1)/2);
        }
}
```

Dado que para determinar a mediana, os valores das médias de cada bloco têm de estar organizados de forma crescente, na primeira linha é criada uma lista auxiliar, que corresponde a uma "cópia" da lista original, `blockMean`, para evitar que esta seja alterada. A lista `aux` é então organizada com o método `sort` da *framework* `Collections` e é determinada a mediana. O `type=1` corresponde a uma sequência de valores de comprimento par e `type=2` de comprimento ímpar.

Finalmente, é determinado o valor *hash*. O processo consiste em criar uma *String* de zeros e uns. Os valores das médias dos blocos são comparados à mediana; se forem inferiores a esta, é

incrementado o caracter ‘0’ à String `hash` (no código em baixo) e se forem superiores ou iguais, é incrementado o caracter ‘1’.

```
private String hash(double[] means, double median){
      String hash="";
      int ones=0,zeros=0;
      for(int i=0;i<means.length;i++){
            if(means[i]<median)
                  hash+=0;
            else
                  hash+=1;
      }
      return hash;
}
```

Comparar os valores *hash*

Após o cálculo do valor *hash* da imagem *input,* é possível comparar este valor com os valores *hash* dos objetos na lista *mnts.* Isto é feito calculando a distância de *Hamming* para cada par: valor *hash* da imagem original – valor *hash* do objeto na lista. Cada uma destas distâncias é definida no atributo *dist* da classe *Monument* para cada objeto, segundo o método `sethashDistance(int dist)`. Quando este processo estiver concluído, a lista *mnts* é organizada segundo o atributo *dist*. O resultado final, será os objetos com a menor distância a aparecerem no início da lista e os com as maiores distâncias, no final.

SIFT

Por último, a lista com os objetos candidatos, *mnts,* é analisada, por forma a determinar qual destes corresponde melhor com o objeto de interesse. A análise é iniciada com uma condição que impõe que a distância de *Hamming* tem de ser inferior ou igual a 35. Este valor foi determinado empiricamente, após vários testes de semelhança entre imagens pelas funções *hash,* onde valores acima de 35 garantiam que duas imagens não são semelhantes. Se esta condição não for aceite, o objeto *i* é rejeitado e passa-se para o próximo. Se for aceite, o conjunto de imagens do atributo *img* (caso haja mais do que uma) é separado em diferentes *Strings*, e associado à variável `String[]seperatedimages`.

O passo seguinte, consiste na interação do *Array* `seperatedimages`. Nesta iteração, é criado um ficheiro .jpg da imagem atual e um outro da imagem do objeto candidato. Ambos os ficheiros se encontram no mesmo diretório. Esta imagens são depois usadas para o processar o algoritmo SIFT. O código relativo a este algoritmo foi desenvolvido por Stephan Saalfeld [16] e

importado para este projeto. Para executar o algoritmo, antes da iteração de `seperatedimages` é criada uma nova instância da classe `SIFT_Align` com `align=new SIFT_Align()`. Depois dentro da iteração é invocado o método `align.run("true")`, que por sua vez vai buscar as imagens ao diretório onde estas se encontram, e processar o SIFT. Do resultado deste processamento interessa o resultado de duas variáveis: `resultantEpsilon e n_inliers`. A primeira variável corresponde ao desvio entre pontos chave correspondentes nas duas imagens e `n_inliers` corresponde ao número de pontos chave que tiveram correspondência. Ou seja, o que se pretende é um valor para `resultantEpsilon` baixo e um valor para `n_inliers` alto. Durante a iteração estes valores são incrementados nas variáveis: `float mediumEpsiolon` e `float mediumInliers`. São depois adicionados à variável `Map<Integer,String> siftResults`, que vai conter as médias das variáveis `mediumEpsiolon e mediumInliers` de todos os objetos. Estes valores são introduzidos da forma: `siftResults.put(identificador,mediumEpsiolon+"-"+mediumInliers)`. No final, é escolhido o melhor candidato e é enviado o *id* do mesmo ao cliente. Caso não seja encontrado nenhum candidato então, o objeto de interesse é adicionado à base de dados com o seguinte comando SQL:

```
"INSERT INTO mnt (\"nome\",\"hash\", \"img\", \"desc\", \"lat\", \"lng\")
"
VALUES
(\'"+inputMonument.getName()+"\',\'"+inputMonument.getHash()+"\',\'"+in
putMonument.getImg()+"',\'"+inputMonument.getDesc()+"\',\'"+inputMonume
nt.getLat()+"\',\'"+inputMonument.getLng()+"\')"
```

3.5.5 Atualizar um objeto

Para atualizar um objeto é usado o recurso `/monuments/id/{id}`. Este recurso desempenha a função de atualizar os dados relativos a um determinado objeto, identificado pelo seu *id*. O recurso é criado segundo o seguinte código:

```
@PUT
@PATH("id/{id})
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
Public Status update(@PathParam("id") int id, Monument mnt){
      db = new DataBase();
      db.updateMonument(mnt,id);
      return new Status("SUCCESS", "Updated "+ mnt + "id "+ id);
      }
}
```

O recurso consome e produz conteúdo no formato JSON. O método `update` recebe as variáveis `int id` e `Monument mnt`. O valor do parâmetro do URI, `{id}`, é associado à variável `int id`. O método começa com a criação de uma nova instância da classe `DataBase()`, em `db = new DataBase()`. `db` é depois usada para realizar o método `updateMonument(mnt, id)`. Este método vai executar uma atualização à base de dados:

```
"UPDATE mnt SET \"nome\" =\'" +mnt.getName() + "\', \"hash\" = \'" +
mnt.getHash() + "\', \"img\" = \'" + mnt.getImg() + "\', \"dic\" = \'" +
mnt.getDesc() + "\', \"lat\" = " + mnt.getLat() + ",\"lng\" = " +
mnt.getLng() + "\' WHERE \"id\" = "+ id + ""
```

No final, é enviada uma resposta com a informação de que a atualização foi bem sucedida, e uma mensagem com a declaração dos conteúdos dos campos que acabaram de ser atualizados.

3.6 Aplicação Móvel

A aplicação móvel permite ao utilizador obter informação dos monumentos de interesse. O esquema seguinte mostra a interação entre a aplicação móvel e as outras componentes do sistema:

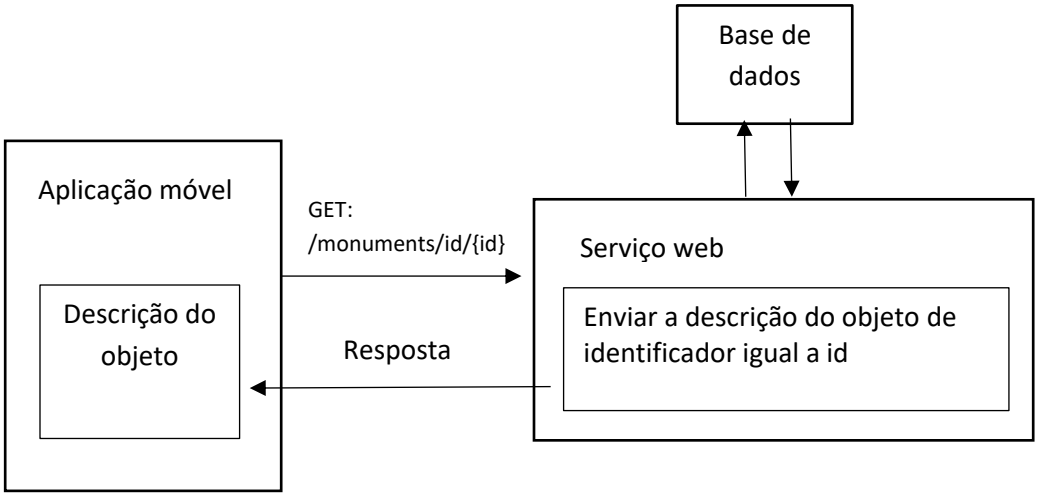

Figura 3.3: Esquema de funcionamento da aplicação móvel

Após a determinação do objeto candidato, o *id* deste é enviado para a aplicação móvel. Este *id* é depois usado para fazer um pedido ao serviço web (`GET: /monumentst/id/{id}`) do objeto em questão. O serviço Web faz uma interrogação à base de dados para extrair informação relativa a este objeto. Esta informação é por fim enviada à aplicação móvel, e é monstrada ao

utilizador. Nas secções seguintes será descrito a interface da aplicação móvel e as diversas funcionalidades.

3.6.1 Interface da aplicação móvel

O primeiro aspeto a considerar é como vai ser concebida a interface gráfica da aplicação. Foi decidido, desde o início, que esta, deveria ser o mais simples possível. A interface será dividida em duas componentes: uma componente corresponde à captura da imagem e a outra à apresentação da informação descritiva do objeto. O que foi proposto, como interface, está representado na Figura 3.4. Na imagem à esquerda, tem-se a componente da captura de imagem, que consiste apenas num ecrã em que é visualizado o que está a ser capturado pela câmara fotográfica e um botão - em baixo, cuja função é registar a imagem e as coordenadas do dispositivo móvel. A imagem à direita, representa a segunda componente, cujo objetivo é apresentar o nome do objeto, a imagem e descrição do mesmo.

Figura 3.4 - Protótipo da aplicação móvel.

3.6.2 Captura de imagem

Tal como foi mencionado, a aplicação móvel é iniciada com a câmara pronta a tirar uma fotografia. Isto implica que o *display* do dispositivo mostre ao utilizador o que está a ser captado pela câmara e que esta, esteja pronta para a captura. O desenvolvimento desta componente foi adaptado a partir do código em [23], que faz uso do pacote *android.hardware.camera2* [6].

A aplicação começa com a atividade *CameraActivity.java*, onde é definido o *layout activity_camera.xml* que corresponde à interface da primeira componente; depois, é instanciado o fragmento *Camera2BasicFragment.java*. Este fragmento é responsável por exercer as operações necessárias para a captura e registo da fotografia no dispositivo. Quando o botão for premido, estas operações são executadas e no final é iniciada a atividade *ResultActivity.java*, que dá início à segunda componente, apresentação da descrição do monumento.

3.6.3 Apresentação da descrição

A atividade começa com a definição do *layout result_activity_layout.xml*. Este é constituído por um *toolbar* - que apresenta o nome da aplicação, um *ScrollView,* o qual contém um *TextView* no topo (nome do objeto), uma *ImageViem* no meio (imagem do objeto) e um *TextView na* base (descrição do objeto).

A imagem que foi capturada - na componente anterior - é, primeiro, redimensionada para um tamanho fixo de 306 píxeis de largura e 408 píxeis de altura e depois, é codificada para uma *String* Base64 segundo o formato *Base64*. De seguida é determinada a última localização do utilizador, com a utilização das APIs de localização disponíveis nos serviços da Google Play [8]. Em baixo está representado o código para a determinação da localização.

```
FusedLocationProviderClient  mFusedLocationClient.getLastLocation()
.addOnSuccessListener(ResultActivity.this,
 new OnSuccessListener<Location>() {
           @Override
           public void onSuccess(Location location) {
               double lng=location.getLongitude();
               double lat=location.getLatitude();

               if(location != null){
                  POST task = new POST();
                  String json="{\"name\": \"Sem nome\", \"img\" : \"" +
base64img + "\", \"desc\" : \" Sem discrição \" , \"lng\" : "+ lng + ",
\"lat\" : " + lat + "}";
                  task.execute(json);
```

```
                    // definir altura do scrollview
                    Display display = getWindowManager()
.getDefaultDisplay();
                    Point size = new Point();
                    display.getSize(size);
                    int height = size.y;

                    scrollview.getLayoutParams().height=height-
myToolbar.getLayoutParams().height;

                }
                else{
                    result.setText("Erro !");
                }
            }
        });
```

No método `onSuccess(Location location)` é feito um pedido POST, onde são passados a imagem (base64Img), a latitude (lat) e longitude (lng). Isto é conseguido criando uma instância da classe `POST`, em `POST task = new POST().` A variável `task` executa o método `execute(json)` que toma de entrada a *String* - em formato JSON - com os dados. Na classe POST, no método `doInBackground(String… json)` é feito o pedido ao recurso:

http://194.117.43.210/reconhecimentodemonumentos/data/monuments.

Após o pedido, é recebida uma resposta com o *id* do objeto candidato. Se o *id* for igual a 1, tem-se a situação onde nenhum objeto semelhante ao objeto de interesse foi encontrado, deste modo a componente *TextView* (nome do objeto) é definida pela mensagem: "Monumento não existente!" e a componente *TextView* (descrição do objeto) é definida pela mensagem: "O monumento não consta da base de dados!". Caso o *id* seja diferente de 1, é feito um pedido GET ao recurso:

http://194.117.43.210/reconhecimentodemonumentos/data/monuments/id/{id}

O resultado deste recurso corresponde aos dados do objeto candidato. Estes são usados para definir as componentes *TextView* (nome do objeto), *ImageViem* (imagem do objeto) e *TextView* (descrição do objeto).

3.6.4 Exemplo de utilização

Nesta secção é apresentado um cenário real de utilização da aplicação. O objeto em questão corresponde ao busto de João Gonçalves Zarco da Câmara, que se encontra no Rossio, entre o Teatro Nacional D. Maria II e a estação do Rossio. A Figura 3.5 à esquerda, consiste no estado inicial da aplicação. A câmara está em linha com o objeto e esta está pronta para capturar a imagem. Após premir o botão, há uma pequena pausa, e depois surge o *container* com a descrição. Este surge com uma animação, "desliza" de baixo do dispositivo até ao topo. Neste caso, o resultado é uma notificação que informa o utilizador de que o objeto não existe.

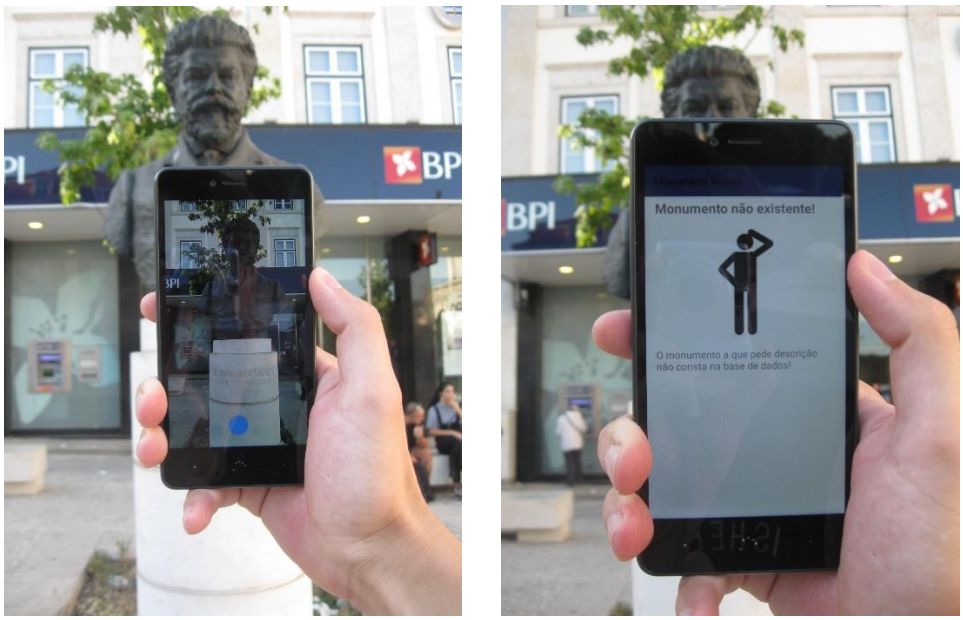

Figura 3.5 - Exemplo de utilização de objeto não reconhecido.

Como o objeto não existe, foi adicionado à base de dados. A partir da aplicação Web é possível preencher os campos relativos ao nome do objeto e a sua descrição. Assim, o próximo utilizador que utilizar a aplicação no busto de João Gonçalves Zarco da Câmara terá acesso a toda essa informação, como é mostrado na Figura 3.6 à direita.

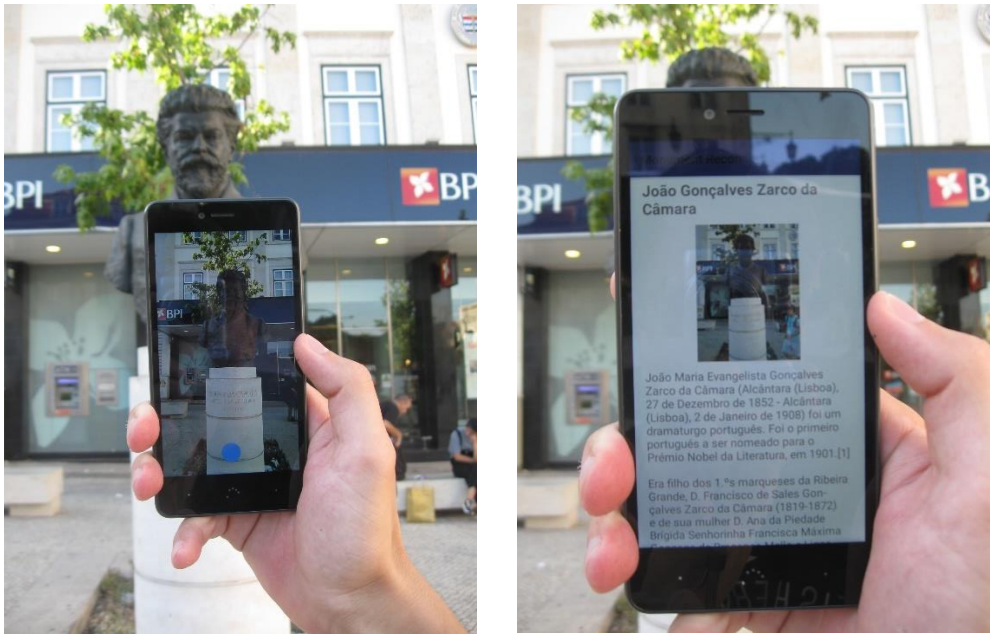

Figura 3.6 - Exemplo de utilização de objeto reconhecido

A aplicação pode ser obtida em: https://jrodriguesportfolio.com/app/

3.7 Aplicação Web

A aplicação surge com o intento de dar aos utilizadores uma ferramenta para atualizar o conteúdo sobre os objetos de interesse na base de dados duma forma interativa. Este conteúdo será aquele que é depois apresentado aos utilizadores na aplicação móvel.

3.7.1 Interface da aplicação Web

A interface desta componente é constituída por um mapa – o qual cobre a página na totalidade – com um menu no topo. Este mapa vai conter marcadores correspondentes aos objetos que foram registados pelos utilizadores. Os marcadores vão conter 3 funcionalidades:

- Ao passar com o rato sobre o marcador surgirá uma pequena janela com o nome e a imagem do objeto (Figura 3.7).
- Ao clicar no marcador surgirá uma janela maior com a imagem, nome e descrição do objeto. Esta janela dá ao utilizador a capacidade de acrescentar/editar o conteúdo relativo ao objeto (Figura 3.8).

- Ao arrastar o marcador, é possível dar a este uma nova localização caso as coordenadas do monumento estejam erradas.

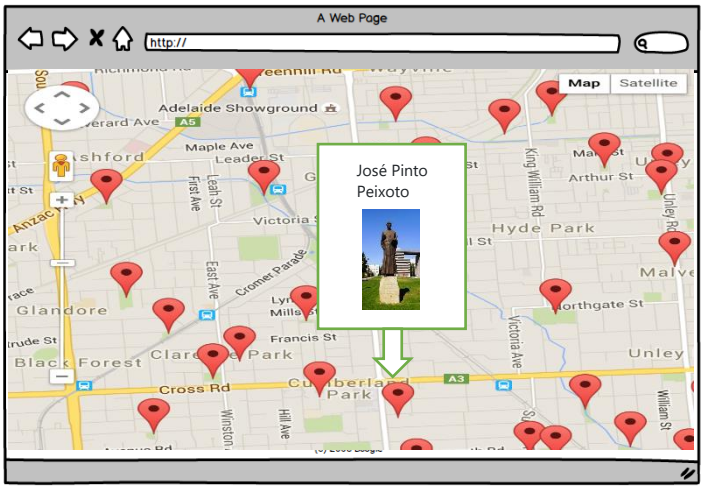

Figura 3.7 - Protótipo da aplicação Web: visualização do nome do objeto.

Figura 3.8 - Protótipo da aplicação Web: visualização e edição da descrição do objeto

3.7.2 Desenvolvimento

A aplicação é constituída por um navegador, um *container* – que corresponde ao mapa - e uma secção, composta por uma imagem, dois *headers*, e um parágrafo. Para o desenvolvimento desta aplicação foi utilizada a API *Maps JavaScript*.

Toda a funcionalidade começa com a função `initMap()`. Nesta função é primeiro instanciada a classe `google.maps.Map` associada à variável `var map`, segundo o código em baixo:

```
var map = new google.maps.Map(document.getElementById('map'), {
            center: new google.maps.LatLng(38.7266700, -9.1633300),
            zoom: 12,
            styles: [ […] ],
            });
```

A classe toma como argumentos o elemento `'map'`, que corresponde ao *container* que vai alojar o mapa, e um objeto JSON, onde é definida a localização inicial do mapa em `center` (neste caso, corresponde ao centro de Lisboa), a escala do mapa em `zoom` e o estilo deste em `styles`. O próximo passo consiste na criação dos marcadores. Para isto, é necessário, primeiro, fazer um pedido GET ao recurso:

http://194.117.43.210/reconhecimentodemonumentos/data/monuments

para obter os objetos da base de dados. O pedido é feito do seguinte modo:

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
            myObj = JSON.parse(this.responseText);
            for (var key in myObj) {
                  if (myObj.hasOwnProperty(key)) {
                        var titulo = myObj[key]["name"];
                        var img = myObj[key]["img"].split(";")[0];
                        var point = new
google.maps.LatLng(myObj[key]["lat"],myObj[key]["lng"]);
```

Na primeira linha é declarado o objeto `var xmlhttp`. Este objeto é responsável por fazer pedidos ao servidor Web. Na linha seguinte é especificada uma função a ser executada sempre que o estatuto de `XMLHttpRequest` for alterado. De seguida, tem-se a condição para verificar se a resposta está pronta, caso sim, a mensagem da resposta é convertida num objeto JSON, `myObj`. Feito isto,

procede-se à criação dos marcadores. É criado um ciclo que percorre todos os objetos contidos em `myObj`. Neste ciclo, é primeiro verificado se a propriedade '`key`' existe, depois são definidas as variáveis `titulo`, `img` e `point`. A variável `point` corresponde a um ponto de coordenadas `lat` e `lng`.

No código em baixo são criados os elementos HTML que vão constituir a janela que surgirá quando o utilizador passar com o rato sobre o marcador.

```
var infowincontent = document.createElement('div');
var strong = document.createElement('strong');
strong.textContent = titulo;
infowincontent.appendChild(strong);
infowincontent.appendChild(document.createElement('br'));

var image = document.createElement('img');
image.src = ""+img;
infowincontent.appendChild(image);
```

Por fim, é criado o marcador com o seguinte código:

```
var icon = 'icon.png';
var marker = new google.maps.Marker({
             position: point,
             map: map,
             title: titulo,
             id:key,
             icon:icon
            });
```

É criada a variável `var marker`, que corresponde a uma instância da classe `new google.maps.Marker`, que toma um objeto JSON, onde é definido a posição, o mapa, o título, o *id* e o *icon* do marcador. A cada marcador é-lhe atribuído um conjunto de *listeners* responsáveis por executar um conjunto de operações quando determinada ação ocorrer sobre este. Estes são atribuídos pela função `addClicker(marker, infowincontent)`, em baixo.

```
addClicker(marker, infowincontent);
            }
      }
}};
```

Finalmente é executado o pedido com o código em baixo:

```
xmlhttp.open("GET", url, true);
```

```
xmlhttp.send();
```

A função `addClicker` é constituída por 4 *listeners.* Estes são:

- **mouseover** – faz surgir uma janela com o título e a imagem do objeto, ao passar com o rato sobre o marcador;
- **mouseout** – faz desaparecer a janela gerada por **mouseover,** ao passar com o rato fora do marcador;
- **click** – faz aparecer uma janela com o conteúdo do objeto e funções para acrescentar/editar informação, ao clicar no marcador.
- **draggable** – permite arrastar um dos marcadores para uma nova localização no mapa.

A janela que surge quando um marcador é clicado, exibe ao utilizador a informação do objeto em questão relativamente ao título, imagem e descrição. Os valores do título e descrição podem ser alterados através do botão **Edit**, que se encontra no fundo da janela. Este, ao ser premido, substitui o *heading* - relativo ao título - e o parágrafo - relativo à descrição, por dois elementos <textarea>. Estes elementos permitem ao utilizador acrescentar/editar informação. Por *deafault*, cada elemento <textarea> contém os valores do *heading* e do parágrafo. Quando as alterações estiverem concluídas, prime-se o botão **Save.** Este, "agarra" nos valores dos elementos <textarea> e faz um pedido PUT ao recurso:

```
http://194.117.43.210/reconhecimentodemonumentos/data/monuments/id/{id}
```

por forma a atualizar os campos da base de dados relativas ao objeto `monuments/id/{id}`. Se o pedido for bem-sucedido, os elementos <textarea> são eliminados, e substituídos por um novo *heading* e um novo parágrafo com os valores atualizados.

3.7.3 Exemplo de utilização

Ao iniciar a aplicação, é visualizado um mapa com os marcadores de todos os objetos contidos na base de dados, como é mostrado na Figura 3.9.

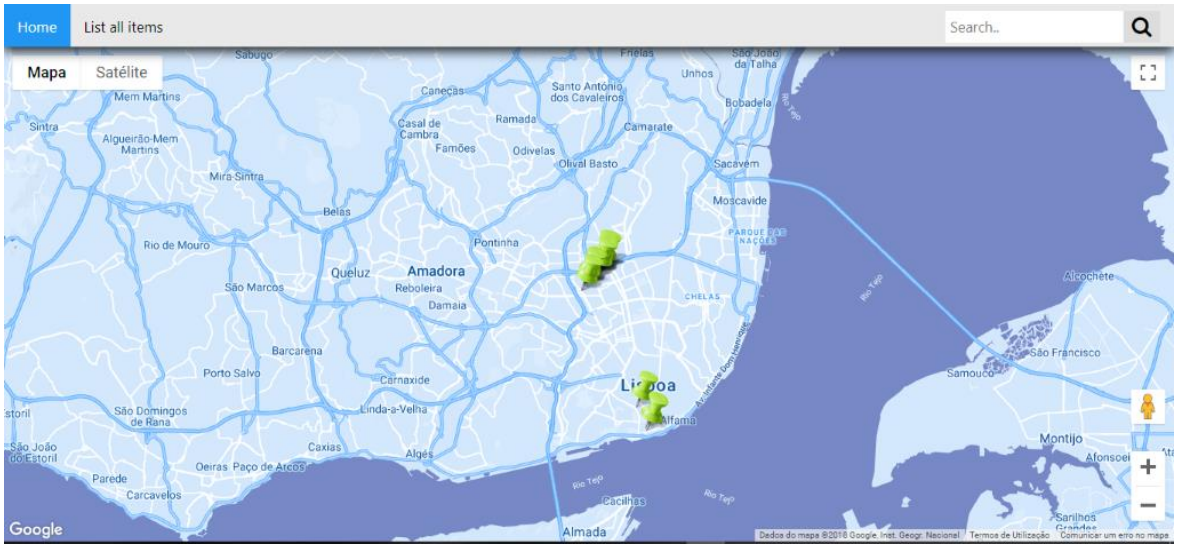

Figura 3.9 - Página inicial da aplicação Web.

Ao passar com o rato sobre o marcador, de por exemplo, do busto de João Câmara (registado pela aplicação móvel), surge uma pequena janela com o título e a imagem do objeto, tal como mostrado na Figura 3.10. Neste caso, ainda não existe informação relativa ao objeto, logo o título corresponde a "null".

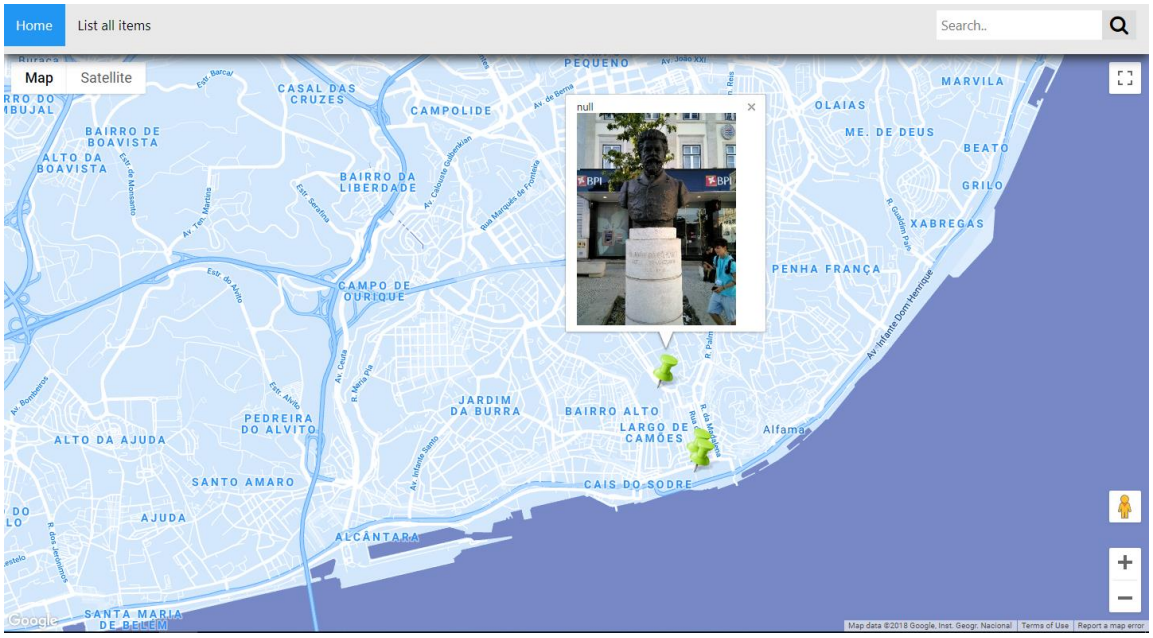

Figura 3.10 - Exemplo da informação associada a um marcador

Ao clicar no marcador, surge uma janela com a imagem, título e descrição do objeto, com os seus valores iguais a “null”, como é mostrado na Figura 3.11.

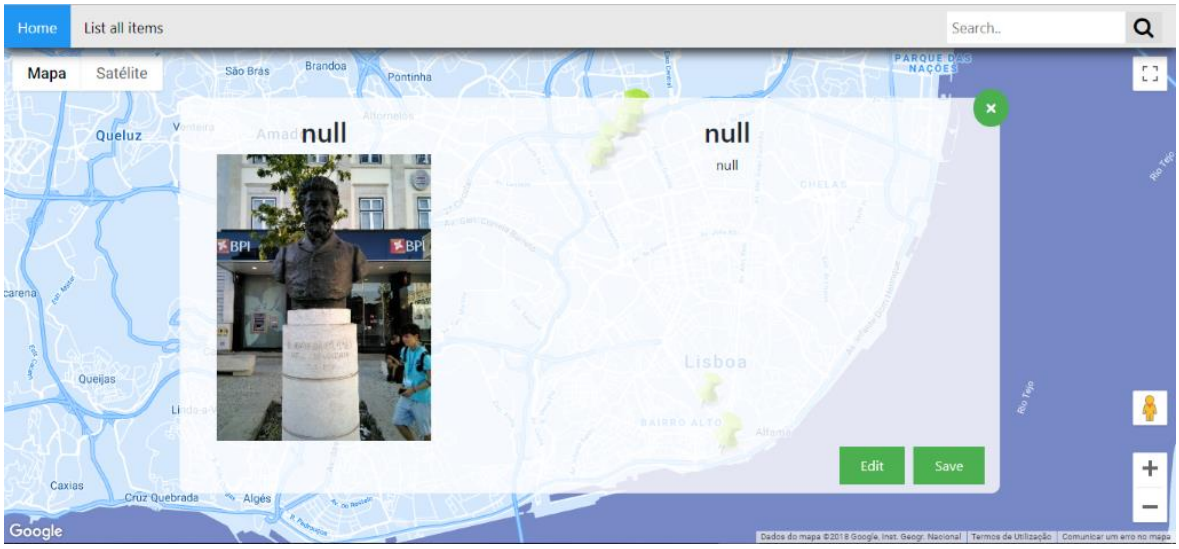

Figura 3.11 - Exemplo de visualização de monumento sem informação.

Para acrescentar informação, clica-se no botão **Edit**, onde surgem duas caixas de texto que permitem ao utilizador escrever, o nome do objeto e a descrição, respetivamente, como é exemplificado na Figura 3.12. Quando estiver tudo concluído, clica-se no botão **Save**, e a informação dos campos *titulo* e *descrição* são atualizados (Figura 3.13).

Figura 3.12 - Exemplo de edição de informação de um objeto

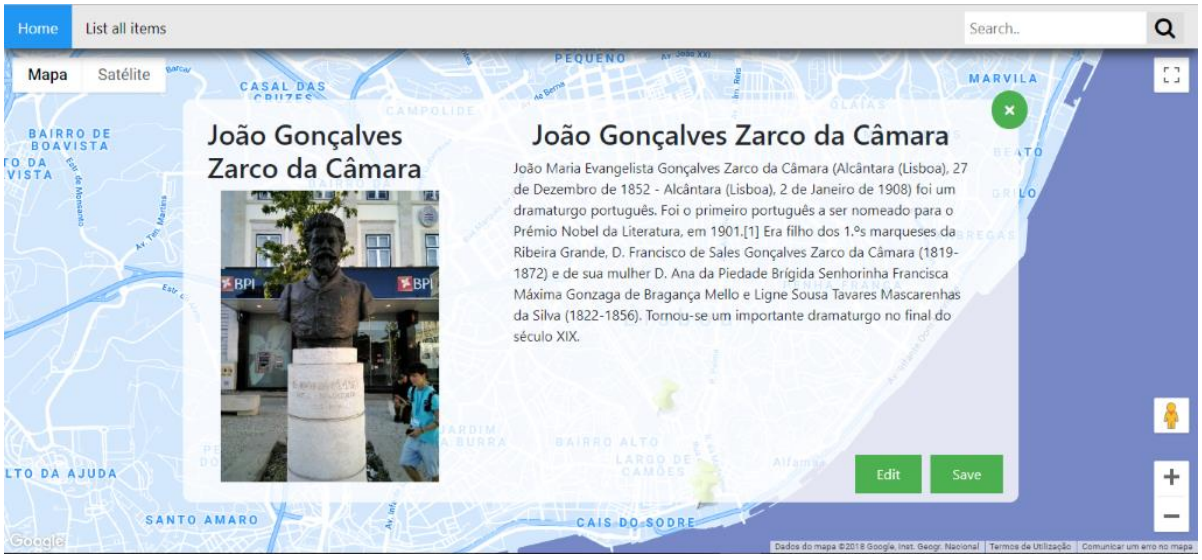

Figura 3.13 - Informação visualizada após edição da informação

A aplicação por ser acedida pelo link: http://194.117.43.210/.

3.8 Teste do Sistema

Após a finalização da aplicação móvel e da aplicação Web, foram feitos vários testes. Estes testes tinham como objetivo avaliar a comunicação entre as aplicações e o serviço Web, de modo a garantir que a informação é carregada da base de dados para o dispositivo móvel, ou browser, de forma consistente e garantir que os dados recolhidos pelo dispositivo móvel – coordenadas e imagem – são devidamente guardados na base de dados. Outro aspeto a avaliar é perceber o nível de exatidão do reconhecimento de objetos e o tempo de execução dos algoritmos de reconhecimento, desde a captura da imagem à apresentação dos resultados.

No caso da aplicação móvel os vários testes foram realizados em 4 fachadas e 9 estátuas; 13 monumentos no total. Em cada situação, para cada objeto, foram capturadas imagens em diferentes ângulos e escalas, e em períodos diferentes do dia, para testar o impacto da luminosidade no reconhecimento.

Verificou-se que, em primeiro lugar, não surgiram problemas a registar as coordenadas do dispositivo móvel, capturar a imagem e a enviar estes para o servidor. Da mesma forma, também não foram detetadas falhas no processamento dos algoritmos de reconhecimento, na deteção do objeto candidato, e no envio e exibição da informação deste no dispositivo móvel. Todo o processo levou uma questão de segundos. Em segundo lugar, após testar a aplicação em várias estátuas e fachadas, surgiram situações onde o reconhecimento do objeto falhava, apresentando informação correspondente ao objeto que não aquele que estava a ser avaliado ou simplesmente não fazendo qualquer reconhecimento.

Para avaliar os efeitos da luminosidade, os casos de teste foram elaborados durante o período da manhã e final da tarde. Sendo que um objeto que foi registado no período da manhã foi depois avaliado no período da tarde. No final, verificou-se que as alterações na luminosidade tiveram um impacto negativo muito significativo no reconhecimento dos objetos, cerca de 84% dos casos não resultaram na identificação correta dos objetos em análise.

Para perceber o impacto que a geometria dos objetos tem no reconhecimento, foram feitos novos testes, aos mesmos monumentos, nas mesmas condições de luz. Os resultados demonstraram que objetos com um número reduzido de características distintas, ou com uma geometria mais angular, fazem com que o reconhecimento seja pouco exato. Neste cenário, houve uma taxa de sucesso de aproximadamente 46%.

Para colmatar estes desafios, a solução passa por guardar o registo das características dos objetos adquiridos a partir de várias imagens obtidas de diferentes ângulos, diferentes escalas e diferentes condições de luz. Com mais informação relativa a cada objeto o reconhecimento melhora.

Quanto à aplicação Web, a página não apresentou problemas a carregar o mapa, a carregar os objetos contidos na base de dados e a gerar os marcadores nas localizações corretas. Os testes feitos consistiram em clicar nos marcadores, acrescentar e atualizar conteúdo relativo aos monumentos, e abrir e fechar janelas. Em todos os casos, a aplicação apresentou ser fluída e consistente. Não surgiram problemas na funcionalidade básica da aplicação. Os marcadores abriram sempre a janela com a informação descritiva atualizada do objeto em questão. Não houve

problemas em gravar as alterações feitas à descrição dos objetos na base de dados nem depois a carregar estas novas atualizações.

Capítulo 4 - Conclusões e Trabalho Futuro

Durante a elaboração do projeto foram abordados conhecimentos e tecnologias os quais estava pouco ou nada familiarizado. Isto levou a que durante a fase de desenvolvimento tenham sido adquiridas novas competências e conhecimentos. Dado à falta de experiência, nunca foi claro no início do projeto, se os objetivos que foram propostos seriam cumpridos com sucesso, o que gerou um estímulo adicional.

A primeira abordagem do projeto consistiu no desenvolvimento da componente funcional do sistema: o serviço Web. Aqui, as dificuldades que surgiram foram relacionadas com, em primeiro lugar, explorar as ferramentas necessárias e mais adequadas à criação do serviço Web: o Java EE, do IDE Eclipse e a API JAX-RS implementada pela *framework* Jersey. Segundo, perceber como usar estas ferramentas para criar um serviço Web segundo a arquitetura REST.

Com o serviço web operacional procedeu-se ao desenvolvimento da aplicação móvel para o sistema operativo *Android*. Esta fase, revelou-se muito mais complicada e morosa, do que foi inicialmente planeado. Isto deveu-se, naturalmente, à falta de conhecimentos e experiência no desenvolvimento de aplicações *Android,* contudo, a maior dificuldade deveu-se em grande medida, à criação da interface com a câmara fotográfica, onde surgiram complicações a criar o ecrã que apresenta ao utilizador o que a câmara está a captar. Previa-se que esta fase do projeto não levaria mais do que 2 dias a concluir, acabou por levar perto de uma semana.

Por fim, foi criada a aplicação Web. Esta, revelou-se a componente mais simples de desenvolver. No processo de conceção foram adquiridas competências relativamente à utilização da API *Maps JavaScript*, que serviu para a criação de um mapa que automaticamente adiciona marcadores, correspondentes aos objetos contidos na base de dados, os quais apresentam informação relativa aos mesmos.

O sistema foi testado e a aplicação móvel foi avaliada em várias estátuas e algumas fachadas em diferentes ângulos de observação e diferentes condições de luminosidade. Em muitas situações, a aplicação detetou com sucesso estes monumentos, outras não. O que se percebeu foi que se o objeto não tem elementos característicos distintos suficientes, o reconhecimento deste torna-se difícil, o mesmo acontece se objeto a ser reconhecido, estiver sobre condições de luz diferentes daquelas a que foi registado originalmente. Estes problemas podem ser atenuados com o registo de várias imagens de cada objeto, adquiridas de diferentes ângulos, diferentes escalas e diferentes condições de luz.

O campo de estudos dedicado ao reconhecimento de objetos tem evoluído lentamente ao longo dos anos, dada a complexidade envolvida neste processo. Recentemente, tem havido um

crescente interesse no assunto, em parte, devido à popularização da realidade aumentada, ao aumento do poder computacional e novas técnicas de reconhecimento que envolvem o uso de inteligência artificial.

Referências

[1] AI Shack. http://aishack.in/tutorials/sift-scale-invariant-feature-transform-features/ [Acedido: 18-Jun-2018]

[2] Apple Developer. Augmented Reality. Disponível em: https://developer.apple.com/arkit/ [Acedido: 10-Jun-2018.]

[3] Boston Geographic Information Systems. Part 1: Getting Started With PostGIS: An almost Idiot's Guide (PostGIS 2.2). Disponível em; https://www.bostongis.com/PrinterFriendly.aspx?content_name=postgis_tut01 [Acedido: 23-Jun-2018]

[4] Eclipse Foundation. Eclipse IDE for Java EE Developers. Disponível em: http://www.eclipse.org/downloads/packages/release/oxygen/3/eclipse-ide-java-ee-developers [Acedido: 17-Jun-2018]

[5] Fridrich, Jiri. Goljan, Miroslav. Robust Hash Functions for digital watermarking.

[6] Google Developers. android.hardware.camera2. Disponível em: https://developer.android.com/reference/android/hardware/camera2/package-summary [Acedido: 9-Jul-2018]

[7] Google Developers. ARCore. Disponível em: https://developers.google.com/ar/ [Acedido: 11-Jun-2018]

[8] Google Developers. Get the last known location. Disponível em: https://developer.android.com/training/location/retrieve-current [Acedido:19-Jul-2018]

[9] Google Developers. Maps JavaScript API. Disponível em: https://developers.google.com/maps/documentation/javascript/tutorial [Acedido: 25-Jul-2018]

[10] Harris, C. e Stephens, M. A combined corner and edge detector, 1988

[11] Lowe, David G. Distinctive Image Features from Scale-Invariant Keypoints, 2004.

[12] Microsoft. Visual Studio Code. Disponível em: https://code.visualstudio.com/?wt.mc_id=DX_841432 [Acedido: 17-jun-2018]

[13] Oracle. The Java EE 6 Tutorial. Disponível em: https://docs.oracle.com/javaee/6/tutorial/doc/giqsx.html#gkcdg [Acedido: 16-Jun-2018]

[14] PostGIS. Spatial and Geographic objects for PostgreSQ. Disponível em: https://postgis.net/ [Acedido: 22-jun-2018]

[15] PostgreSQL. Disponível em: https://www.postgresql.org/ [Acedido: 22-jun-2018]

[16] Saalfeld, Stephan. JavaSIFT. Disponível em: http://fly.mpi-cbg.de/~saalfeld/Projects/javasift.html [Acedido: 12-Jun-2018]

[17] Salesforce Tutorials. What is Web Service and What are different types of webservices? Disponível em http://sfdcsrini.blogspot.com/2015/09/what-is-web-service-and-what-are.html [Acedido: 21-jun-2018]

[18] The Apache Software Foundation. Apache Tomcat. Disponível em: http://tomcat.apache.org/ [Acedido: 17-Jun-2018]

[19] Tutorialspoint. RESTful Web Services - Java (JAX-RS). Disponível em: https://www.tutorialspoint.com/restful/restful_jax_rs.htm [Acedido: 19-Jun-2018]

[20] Tutorialspoint. RESTful Web Services – Introduction. Disponível em: https://www.tutorialspoint.com/restful/restful_introduction.htm [Acedido: 19-Jun-2018]

[21] Wikitude. Wikitude AR SDK. Disponível em: https://www.wikitude.com/ [Acedido: 11-Jun-2018]

[22] Zauner, Christoph. Implementation and benchmarking of perceptual image hash functions. 2010

[23] googlesamples. android-Camera2Basic. Disponível em: https://github.com/googlesamples/android-Camera2Basic [Acedido: 14-Jul-2018]

[24] jersey. RESTful Web Services in Java. https://jersey.github.io/ [Acedido: 2-Jul-2018]

[25] pgAdmin. PostgreSQL Tools. Disponível em: https://www.pgadmin.org/ [Acedido: 23-Jun-2018]