

BRUNO ALEXANDRE KRINSKI

DETECTOR E DESCRITOR ORB EM APLICATIVO ANDROID PARA
LOCALIZAÇÃO E INCLUSÃO DE INFORMAÇÕES EM IMAGENS POR
REALIDADE AUMENTADA EM CAMPUS UNIVERSITÁRIO

Trabalho apresentado como requisito parcial à conclusão do Curso de Ciência da Computação - Bacharelado, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo Todt.

CURITIBA, PARANÁ
2016

Resumo

Este trabalho tem como objetivos estudar conceitos de Visão Computacional, Realidade Aumentada e desenvolver um aplicativo móvel, destinado a *smartphones* que utilizam sistema operacional *Android*, com a função de reconhecer os edifícios dos Campi Centro Politécnico e Jardim Botânico da Universidade Federal do Paraná. Para cada edifício, o aplicativo exibe um conjunto de informações na tela do aparelho celular.

Para fazer o reconhecimento de um edifício, foi utilizada a técnica de Visão Computacional de extração e descrição de pontos de interesse da imagem. Nesta, é necessário ter ao menos uma imagem de referência de cada edifício previamente carregada na base de dados do aplicativo. O algoritmo de detecção irá extrair pontos relevantes de cada imagem de referência e da imagem da câmera e, obter descritores de características associados a estes pontos. Após isto, uma comparação entre esses descritores é realizada. Caso os descritores das imagens de referência sejam similares aos descritores da imagem da câmera, o aplicativo exibe informações referentes àquele edifício na tela do aparelho celular.

Experimentos práticos foram realizados através de circuitos nos campi mencionados, contendo 13 edifícios na base de dados do aplicativo. O algoritmo utilizado obteve sucesso em reconhecer os edifícios, mesmo quando há rotação e translação do *smartphone*. Porém, quando há alterações na iluminação do ambiente, quase não há reconhecimento dos edifícios.

Palavras-chave: Realidade Aumentada, Visão Computacional, Android.

Abstract

This work aims to study concepts of Computational View, Augmented Reality and to develop a mobile application, destined to smartphones that use Android operating system, with the function of recognizing the buildings of the Campi Polytechnic Center and Botanical Garden of the Federal University of Paraná. For each building, the application displays a set of information on the mobile device's screen.

To make the recognition of a building, the technique of Computational View of extraction and description of points of interest of the image was used. In this, you need to have at least one reference image of each building previously loaded in the application database. The detection algorithm will extract relevant points from each reference image and the camera image and obtain characteristic descriptors associated with these points. After this, a comparison between these descriptors is performed. If the reference image descriptors are similar to the camera image descriptors, the application displays information about that building on the mobile device screen.

Practical experiments were carried out through circuits in the mentioned campuses, containing 13 buildings in the application database. The algorithm used succeeded in recognizing the buildings, even when there is rotation and translation of the smartphone. However, when there are changes in ambient lighting, there is almost no recognition of buildings.

Keywords: Augmented Reality, Computer Vision, Android.

Sumário

1	Introdução	1
2	Revisão da Literatura	2
2.1	Trabalhos Correlatos	2
2.2	Detecção de Pontos de Interesse	7
2.3	Harris Corner Detector	8
2.4	FAST Corner Detector	10
2.5	BRIEF: Binary Robust Independent Elementary Features	11
2.6	ORB: Oriented FAST and Rotated BRIEF	13
2.7	Hamming Distance	15
2.8	Considerações	16
3	Desenvolvimento	17
3.1	OpenCV	17
3.2	Identificando um edifício	18
4	Implementação	20
4.1	ImageDescriptor	21
4.2	Build	25
4.3	MainActivity	26
4.4	InitFragment	29
4.5	CameraFragment	30
4.6	ShowInformationsFragment	31
4.7	Execução do Algoritmo	32
4.8	Edifícios no Banco de Dados	33
4.8.1	Edifícios do Campus Centro Politécnico	33
4.8.2	Edifícios do Campus Jardim Botânico	38
5	Resultados e Discussão	39
5.1	Testes	39
6	Conclusão	42

Referências Bibliográficas 43

A Códigos 45

A.0.1	MainActivity.java	45
A.0.2	ImageDescriptor.java	53
A.0.3	Build.java	56
A.0.4	ShowInformationsFragment.java	57
A.0.5	CameraFragment.java	59
A.0.6	InitFragment.java	60
A.0.7	activity_main.xml	60
A.0.8	fragment_camera.xml	61
A.0.9	fragment_init.xml	61
A.0.10	fragment_show_informations.xml	61

Lista de Figuras

2.1	Reconhecimento de peças de museu.(Fonte: [Rattanakrungsri and White, 2014])	3
2.2	Localização em ambientes fechados(Fonte: [Delail et al., 2013])	3
2.3	Inserção de objetos virtuais no mundo real(Fonte: [Lin et al., 2015])	4
2.4	Detecção de plano dominante(Fonte: [Lin et al., 2015])	4
2.5	Cenário original e cenário com objeto virtual(Fonte: [Du et al., 2014])	5
2.6	Projeção de objetos 3D em uma cena do mundo real(Fonte: [Ufkes and Fiala, 2013])	5
2.7	Imagem em situação normal(Fonte: [Xia and Zhou, 2013])	6
2.8	ORB funcionando com diferente escala e rotação entre as imagens compara- das(Fonte: [Xia and Zhou, 2013])	6
2.9	Pokémon inserido no mundo real através da tela do <i>smartphone</i> (Fonte: https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo).	7
2.10	Extração de pontos de interesse.(Fonte: [Muhammad, 2015])	8
2.11	Exemplo de extração de cantos através do Harris Corner Detector (Fonte: http://docs.opencv.org/trunk/harris_result.jpg).	10
2.12	FAST Corner Detector.(Fonte: [Rosten et al., 2008])	11
2.13	Exemplo de extração de cantos através do FAST Corner Detector (Fonte: http://docs.opencv.org/3.0-beta/_images/fast_kp.jpg).	11
2.14	Diferença entre a escolha dos locais de teste BRIEF.(Fonte: [Calonder et al., 2010])	12
2.15	Exemplo de reconhecimento de objetos com o ORB(Fonte: [Rublee et al., 2011]).	14
2.16	Extração de pontos de interesse utilizando o ORB(Fonte: http://docs.opencv.org/3.0-beta/_images/orb_kp.jpg).	15
3.1	Exemplo da tela de informações exibida (Fonte: Autor)	19
4.1	Diagrama com as classes do aplicativo e o relacionamento entre elas (Fonte: Autor)	21
4.2	Diagrama de Sequência do Aplicativo (Fonte: Autor)	33
4.3	Prédio da Administração(Fonte: Autor).	33
4.4	Prédio da Biblioteca(Fonte: Autor).	34
4.5	Prédio da Biológicas(Fonte: Autor).	34
4.6	Prédio das PA's(Fonte: Autor).	35

4.7	Prédio de Engenharia Bioprocessos e Biotecnologia (Fonte: Autor).	35
4.8	Prédio de Laboratório de Estudos Sedimentológicos e Petrologia Sedimen- tar(Fonte: Autor).	36
4.9	Restaurante Universitário(Fonte: Autor).	36
4.10	Prédio das PC's(Fonte: Autor).	36
4.11	Prédio das CT's(Fonte: Autor).	37
4.12	Prédio Departamento de Informática(Fonte: Autor).	37
4.13	Prédio de Engenharia Industrial Madeireira(Fonte: Autor).	38
4.14	Prédio de Farmácia (Fonte: Autor).	38
4.15	Prédio de Sociais Aplicadas(Fonte: Autor).	38

Lista de Tabelas

5.1	Resultado obtido através dos testes (Fonte: Autor)	40
-----	--	----

Lista de Acrônimos

RA	Realidade Aumentada
RV	Realidade Virtual
ORB	<i>Oriented FAST and Rotated BRIEF</i>
SIFT	<i>Scale-Invariant Feature Transform</i>
SURF	<i>Speeded Up Robust Features</i>
GPS	<i>Global Position System</i>
KLT	Kanade–Lucas–Tomasi
FAST	<i>Features from Accelerated Segment Test</i>
BRIEF	<i>Binary Robust Independent Elementary Features</i>
OpenCV	<i>Open Source Computer Vision Library</i>

Capítulo 1

Introdução

A localização de edifícios no campus universitário é um problema enfrentado por novos alunos e professores. Isto se deve pela falta de identificação nos edifícios. A grande quantidade de edifícios e o fato de as aulas de um mesmo curso ocorrerem em prédios diferentes são fatores que agravam o problema.

Nos dias atuais, *smartphones* são geralmente equipados com câmeras e sensores de movimento, o que os tornam boas plataformas para aplicações de localização. No desenvolvimento deste trabalho, foram utilizados tanto a câmera quanto o GPS embarcados.

O objetivo deste trabalho é desenvolver um aplicativo de RA para *smartphones*. O aplicativo deve reconhecer os edifícios dos Campi Centro Politécnico e Jardim Botânico através da câmera do celular e, exibir um conjunto de informações destes edifícios na tela do *smartphone*. Espera-se que estas informações sejam úteis para novos alunos e professores obterem informações referentes ao setor da universidade que aquele edifício representa e também, se localizar nos campi mencionados.

Os próximos capítulos estão organizados da seguinte forma: primeiro é realizada uma revisão da literatura sobre trabalhos relacionados com o tema e também sobre os algoritmos utilizados no desenvolvimento deste trabalho. O terceiro capítulo explica como esses algoritmos foram empregados e como é o processo de identificação de um edifício. O quarto capítulo explica como o aplicativo foi desenvolvido, revisando o código e apresentando a base de dados utilizada. Para encerrar, são exibidos os testes realizados, resultados obtidos e quais conclusões podemos retirar destes testes, além de serem comentados os principais problemas e dificuldades enfrentados; e como tais problemas afetam o processo de reconhecimento dos edifícios.

Capítulo 2

Revisão da Literatura

Neste capítulo é apresentada uma revisão da literatura recente sobre o tema em desenvolvimento. O capítulo inicia com alguns exemplos de trabalhos envolvendo a Realidade Aumentada em aplicativos para dispositivos móveis, os algoritmos utilizados pelo autor e a metodologia adotada. Também são explicados os algoritmos da Visão Computacional que foram o substrato tecnológico para o processo de reconhecimento dos edifícios.

A Realidade Aumentada consiste na inserção de objetos e/ou informações virtuais em imagens do mundo real. Ela é uma variação da Realidade Virtual, tecnologia que permite a inserção do usuário em um ambiente sintético substituindo a visão do mundo real. Em contrapartida, a RA permite que o usuário veja o mundo real com objetos virtuais sobrepostos ou compondo sua estrutura. Portanto, a RA complementa a realidade sem substituí-la por completo. Idealmente, deve dar ao usuário a impressão de que objetos virtuais e reais coexistem no mesmo espaço [Azuma, 1997].

2.1 Trabalhos Correlatos

No artigo [Rattanaarungrot and White, 2014], temos um exemplo de aplicação de Realidade Aumentada para dispositivos móveis, o qual utiliza a câmera do *smartphone* para realizar o reconhecimento de peças de museu, exibindo várias informações dessas peças na tela do *smartphone*. Na figura 2.1 é exibido o resultado da aplicação após realizar o reconhecimento da peça e as informações que foram exibidas na tela do celular.



Figura 2.1: Reconhecimento de peças de museu.(Fonte: [Rattanaungrot and White, 2014])

Já no artigo [Delail et al., 2013], os autores propõem um sistema de navegação e localização dentro de ambientes fechados. O sistema proposto foi desenvolvido para *smartphones* e utiliza a Realidade Aumentada para exibir informações da localização do usuário na tela do aparelho celular. Os autores empregam os sensores já disponíveis nos aparelhos celulares para realizar a localização do usuário dentro do ambiente. Também é explicado pelos autores que é possível utilizar técnicas baseadas em visão para fazer o reconhecimento de marcas nas localizações onde se deseja obter algum tipo de informação. Um problema citado pelos autores, é que em lugares muito grandes o *smartphone* pode não conseguir processar toda a base de dados, o que pode ser solucionado movendo o processamento das imagens no banco de dados para um servidor externo. Na figura 2.2 é mostrado um exemplo de informações da localização do usuário, dentro de um ambiente interno, que são exibidas na tela do celular.



Figura 2.2: Localização em ambientes fechados(Fonte: [Delail et al., 2013])

Em [Lin et al., 2015], os autores propõem um sistema de rastreamento para a Realidade Aumentada que utiliza os sensores disponíveis nos aparelhos móveis, tais como o giroscópio, o acelerômetro e o sensor de campo magnético. Também fazem uso da câmera que detecta e segue um plano na imagem sem conhecimento inicial do ambiente, como mostrado na figura 2.4. O sistema proposto também faz a estimativa da posição do aparelho e renderiza objetos virtuais no mundo real, como mostrado na figura 2.3.



Figura 2.3: Inserção de objetos virtuais no mundo real(Fonte: [Lin et al., 2015])



Figura 2.4: Detecção de plano dominante(Fonte: [Lin et al., 2015])

No artigo [Du et al., 2014], os autores propõem um sistema de Realidade Aumentada para fazer a restauração de cenários naturais, permitindo visualizar como o cenário era antes de ser destruído. O método descrito pelos autores utiliza o algoritmo *ORB*(*Oriented FAST and Rotated BRIF*) para fazer a extração e a descrição dos pontos de interesse da imagem. O método utilizado para a comparação de pontos é o do vizinho mais próximo. Utilizam a matriz de homografia para encontrar a relação de mapeamento entre os pontos da imagem. Por fim, objetos

virtuais são inseridos no cenário. A figura 2.5 mostra um exemplo de cenário de referencia no lado esquerdo, e a inserção de uma estrutura virtual dentro deste cenário no lado direito.



Figura 2.5: Cenário original e cenário com objeto virtual(Fonte: [Du et al., 2014])

Outro artigo que propõe o algoritmo *ORB*(*Oriented FAST and Rotated BRIEF* para aplicações de Realidade Aumentada em dispositivos móveis é [Ufkes and Fiala, 2013]. O ORB é utilizado para realizar o reconhecimento de um objeto através da câmera do *smartphone*, e após o reconhecimento, objetos 3D são projetados na imagem, como mostra a figura 2.6.

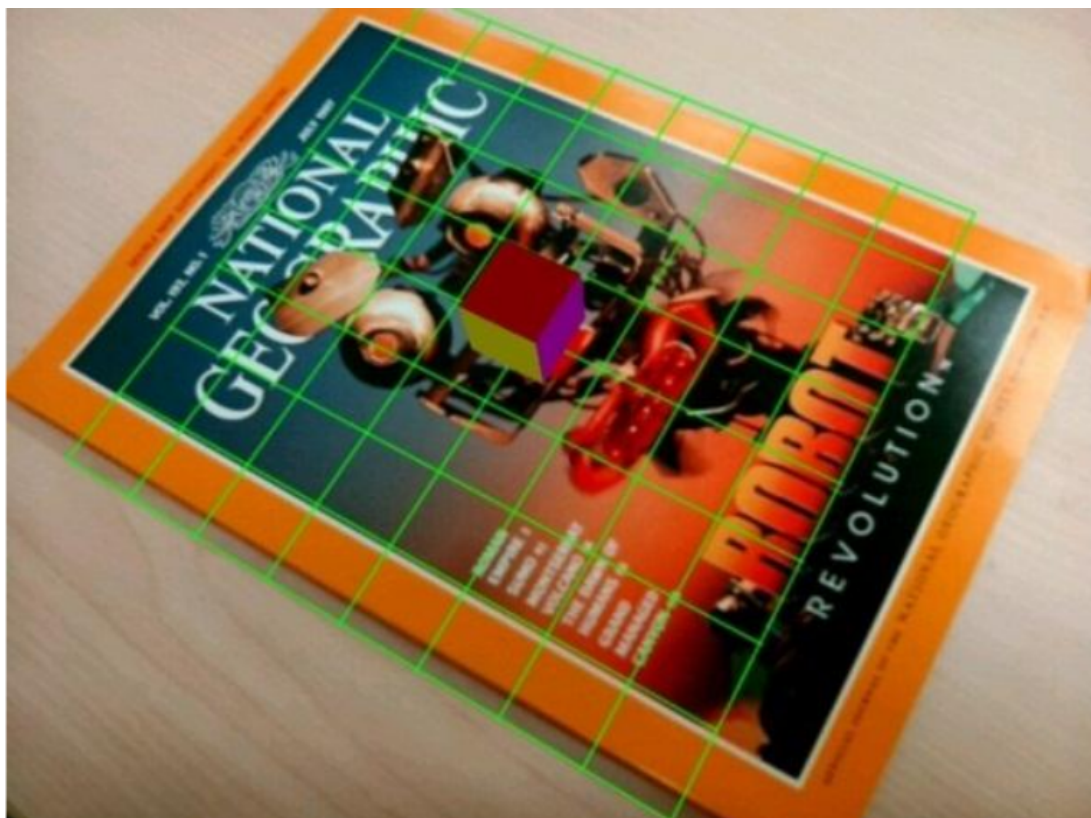


Figura 2.6: Projeção de objetos 3D em uma cena do mundo real(Fonte: [Ufkes and Fiala, 2013])

No artigo [Xia and Zhou, 2013] os autores demonstram o uso do ORB juntamente com algoritmos KLT(Kanade–Lucas–Tomasi) *feature tracker* para o rastreamento e identificação de objetos para sistemas de Realidade Aumentada. Os autores também demonstram neste artigo

o funcionamento do ORB quando há variação de escala e rotação, como mostrado nas figuras 2.7, que mostra a imagem na posição original e 2.8 mostra o algoritmo funcionando quando há alteração na escala e rotação.

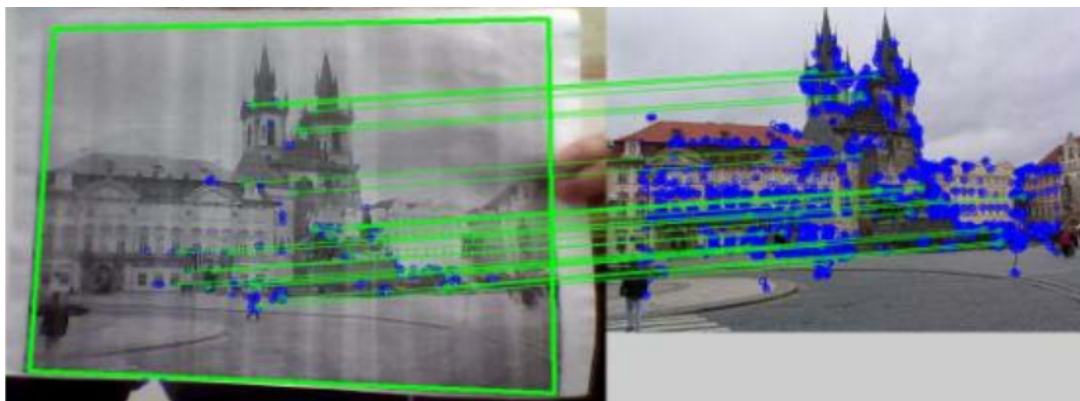


Figura 2.7: Imagem em situação normal(Fonte: [Xia and Zhou, 2013])

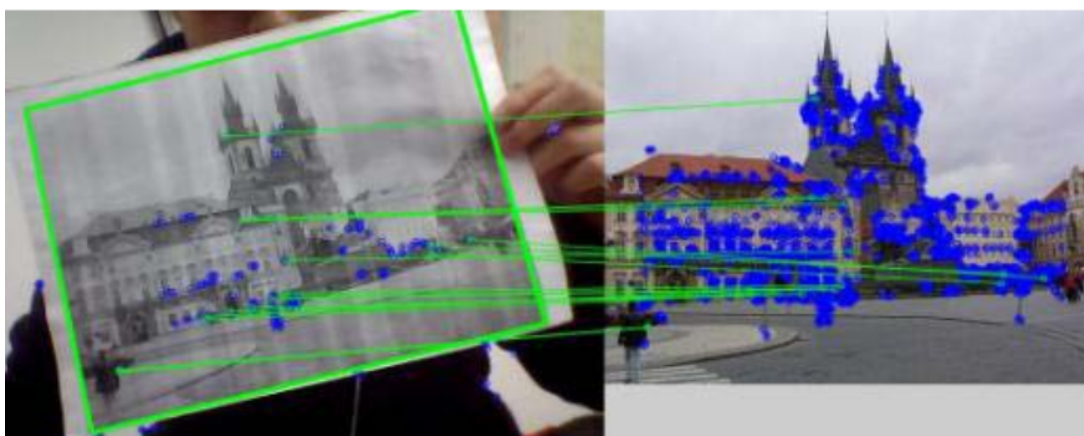


Figura 2.8: ORB funcionando com diferente escala e rotação entre as imagens comparadas(Fonte: [Xia and Zhou, 2013])

Pokémon GO é um jogo de Realidade Aumentada desenvolvido para *smartphones* pelas empresas Niantic, Inc., Nintendo e a The Pokémon Company, para as plataformas iOS e Android. O jogo faz uso do GPS e câmera de dispositivos compatíveis e permite aos jogadores capturar, batalhar, e treinar criaturas virtuais, chamadas Pokémons, que aparecem nas telas de dispositivos como se fossem no mundo real [Niantic, 2016]. Na figura 2.9 é mostrado um exemplo de um pokémon e outras informações sendo inseridas no mundo real através da câmera do *smartphone*.



Figura 2.9: Pokémon inserido no mundo real através da tela do *smartphone* (Fonte: <https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo>).

2.2 Detecção de Pontos de Interesse

Para fazer o reconhecimento de uma imagem ou de um objeto em uma imagem, uma primeira abordagem é comparar todos os pontos desta imagem. Porém, este é um processo muito dispendioso computacionalmente e não são todos os pontos da imagem que trazem algum tipo de informação relevante. Para contornar tal problema é usual utilizar a extração de cantos e pontos de interesse da imagem [Muhammad, 2015].

Pontos de interesse são padrões ou características destacadas em uma imagem. Devem ser facilmente rastreados e comparados. São utilizados para encontrar conjuntos de locais correspondentes em diferentes imagens. Estas correspondências são empregadas no reconhecimento e detecção de objetos em uma imagem. Uma vantagem da extração de pontos de interesse é que, ela encontra correspondência mesmo quando existem oclusão parcial e mudanças na orientação da imagem [Muhammad, 2015] e [Szeliski, 2011].

Na figura 2.10, temos um exemplo de extração de pontos de interesse em uma imagem. A sub-imagem representada pela janela 2 não é um bom ponto de interesse pois, existe uma grande quantidade de pontos similares na imagem, o que gera dificuldades na hora de fazer a comparação. A sub-imagem 1.1 também não é um ponto adequado pois, não há como identificá-lo unicamente. Descolando a sub-imagem 1.1 ao longo da linha há muitas opções nas quais a sub-imagem delimitada pela janela é igual à anterior e não há como diferenciá-la da sub-imagem 1.2 por exemplo. Enquanto o ponto 3 é a melhor opção para ser classificado como ponto de interesse. Movendo a janela 3 em qualquer direção, a área abaixo será diferente e única [Muhammad, 2015].

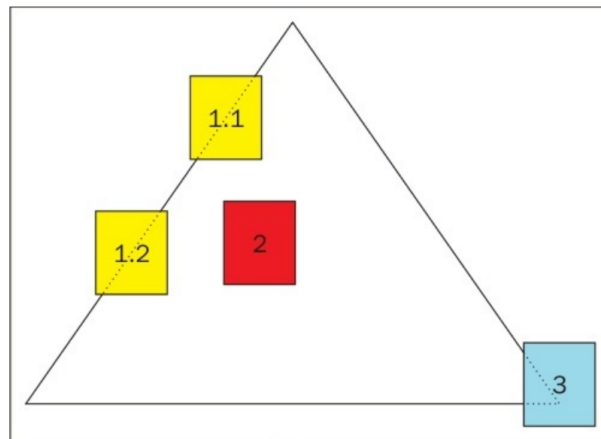


Figura 2.10: Extração de pontos de interesse.(Fonte: [Muhammad, 2015])

2.3 Harris Corner Detector

A interseção de duas arestas, denominada "canto", representa um ponto no qual as direções das duas arestas mudam. E como consequência, o gradiente da imagem em ambas direções ortogonais tem grande variação e pode ser utilizado para ser detectado como um ponto de referência.

O algoritmo *Harris Corner Detector* apresentado em [Harris and Stephens, 1988], utiliza essa ideia para varrer uma imagem I , em tons de cinza, com janelas em cada ponto $w(x,y)$ para calcular a diferença de intensidade.

$$E(u, v) = \sum_{k=x,y} w(x, y) [I(x + u, y + v) - I(x, y)]^2 \quad (2.1)$$

onde:

$w(x,y)$ é a posição da janela em (x,y)

$I(x,y)$ é a intensidade em (x,y)

$I(x+u,y+v)$ é a intensidade movendo a janela $(x+u,y+v)$

Se estivermos olhando para uma janela com um canto, a mesma terá grande variação de intensidade.

$$\sum_{x,y} [I(x+u, y+v) - I(x, y)]^2 \quad (2.2)$$

Usando a expansão de Taylor:

$$E(u, v) \approx \sum_{x,y} [I(x, y) + uI_x + vI_y - I(x, y)]^2 \quad (2.3)$$

Expandindo a equação e cancelando propriamente:

$$E(u, v) \approx \sum_{x,y} u^2 I_x^2 + 2uv I_x I_y + v^2 I_y^2 \quad (2.4)$$

Denotando em forma de matriz:

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \quad (2.5)$$

Usando uma variável auxiliar:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad (2.6)$$

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \quad (2.7)$$

Um escore R é calculado para cada janela, e caso este escore R seja maior que certo valor, então este ponto é um canto.

$$R = \det(M) - k(\text{trace}(M))^2 \quad (2.8)$$

onde:

$$\det(M) = \lambda_1 \lambda_2$$

$$\text{trace}(M) = \lambda_1 + \lambda_2$$

A imagem 2.11 mostra exemplos de extração de cantos e pontos de interesse utilizando o algoritmo Harris Corner Detector. A parte superior da imagem, os pontos em vermelho representam os cantos extraídos da imagem. Na parte inferior da imagem, esta informação é apresentada pelos pontos em verde.

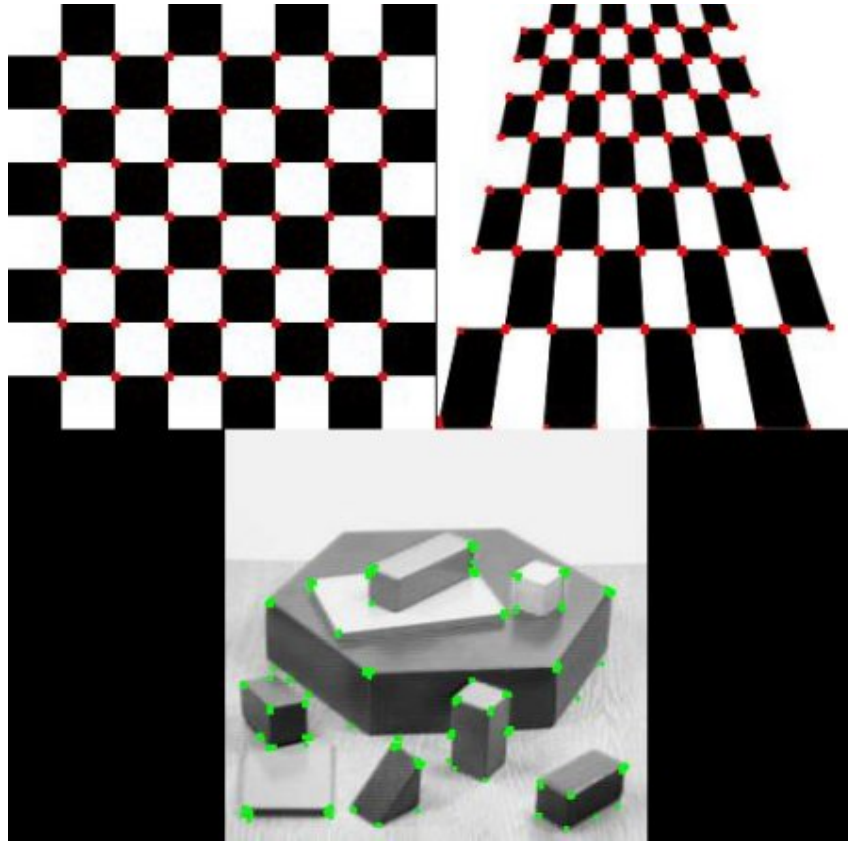


Figura 2.11: Exemplo de extração de cantos através do Harris Corner Detector (Fonte: http://docs.opencv.org/trunk/harris_result.jpg).

2.4 FAST Corner Detector

Fundamentado na ideia de [Smith and Brady, 1997], os autores de [Rosten et al., 2008] propuseram um algoritmo para a extração de pontos de interesse em uma imagem, baseando-se na diferença de intensidade de seus vizinhos.

O algoritmo funciona da seguinte forma: Para cada potencial ponto de interesse ou canto P, são testados os 16 pixels da vizinhança circular do ponto, como mostrado na figura 2.12. Caso 12 desses vizinhos tenham intensidade maior ou menor que a intensidade do ponto P mais/menos um limite, então este ponto é de interesse.

O limite é definido levando em consideração o número de pontos de interesse que se deseja obter na imagem. Quanto maior o limite, maior é a diferença de intensidade entre o ponto de interesse e seus vizinhos.

Uma versão mais acelerada do algoritmo testa a intensidade dos pixels 1, 5, 9 e 13 primeiro, e se a intensidade deles for maior ou menor que a intensidade do ponto P mais/menos um limite, então ele olha os outros 8 vizinhos [Muhammad, 2015].

A imagem 2.13 é um exemplo da extração de pontos de interesse em uma imagem. Os pontos destacados em azul são os pontos de interesse encontrados pelo algoritmo.

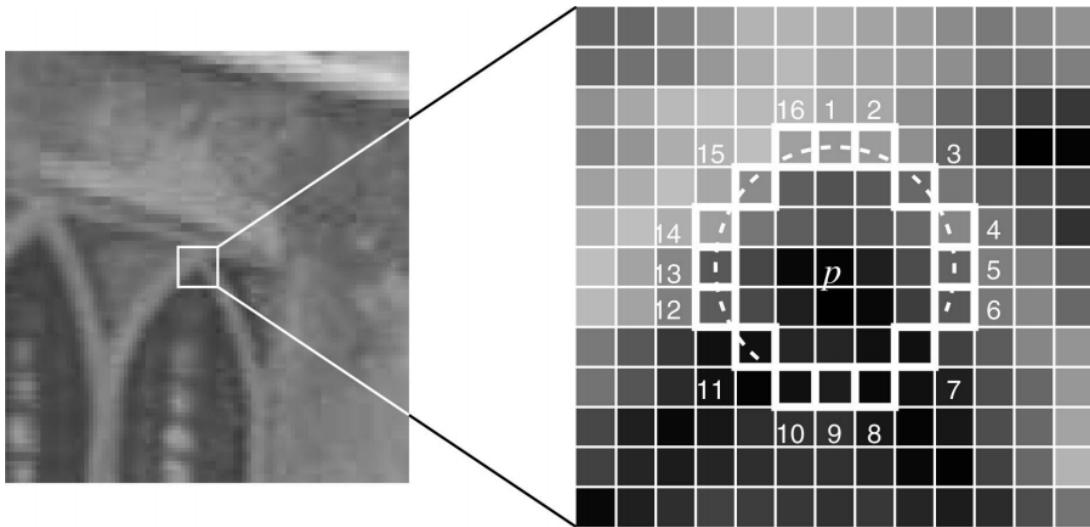


Figura 2.12: FAST Corner Detector.(Fonte: [Rosten et al., 2008])



Figura 2.13: Exemplo de extração de cantos através do FAST Corner Detector (Fonte: http://docs.opencv.org/3.0-beta/_images/fast_kp.jpg).

2.5 BRIEF: Binary Robust Independent Elementary Features

Reconhecimento de objetos, reconstrução 3D, recuperação de imagens e localização da câmera são exemplo de algumas tecnologias da Visão Computacional que nos dias de hoje, devem lidar com um volume de dados cada vez maior ou rodar em dispositivos móveis com recursos computacionais limitados. Para solucionar este problema, descritores locais são cada vez mais necessários, pois são rápidos de computar, fáceis de comparar e têm acesso a memória eficiente [Calonder et al., 2010].

Descritores de características são utilizados para se obter mais informações ao redor do ponto de interesse e são calculados sobre a vizinhança do ponto detectado.

Descritores como *SIFT*(*Scale-Invariant Feature Transform*) [Lowe, 2004] e *SURF*(*Speeded Up Robust Features*) [Bay et al., 2006] são baseados em histogramas de gradien-

tes, e os gradientes de cada pixel na área de interesse precisam ser calculados, e tem alto custo computacional e podem não ser suficientes para algumas aplicações.

O BRIEF [Calonder et al., 2010], utiliza somente a informação única do local do pixel para construir o descritor, e para deixá-lo menos sensível a ruídos, um filtro Gaussiano é utilizado para tornar a imagem mais suave. Para construir um descritor BRIEF de comprimento n , é necessário determinar n pares de pontos (x_i, y_i) , denotados por x e y os vetores dos pontos x_i e y_i respectivamente. Cinco métodos são utilizados pelos autores para encontrar os vetores x e y e o resultado de cada método é ilustrado pela figura 2.14:

G I) As localizações (x_i, y_i) são amostrados por uma distribuição aleatoriamente uniforme.

G II) Os testes são escolhidos de uma distribuição Gaussiana em torno do centro da área de interesse.

G III) As escolhas envolvem dois passos. Primeiro, a localização x_i é escolhida por uma Gaussiana centrada na origem, enquanto a segunda localização, é escolhida por outra Gaussiana centrada em x_i .

G IV) (x_i, y_i) são randomicamente escolhidos a partir de locais discretos de um grid local grosso, introduzindo uma quantização espacial.

G V) para cada i , x_i é $(0,0)$ a y_i pega todos os possíveis valores em um grid polar grosso.

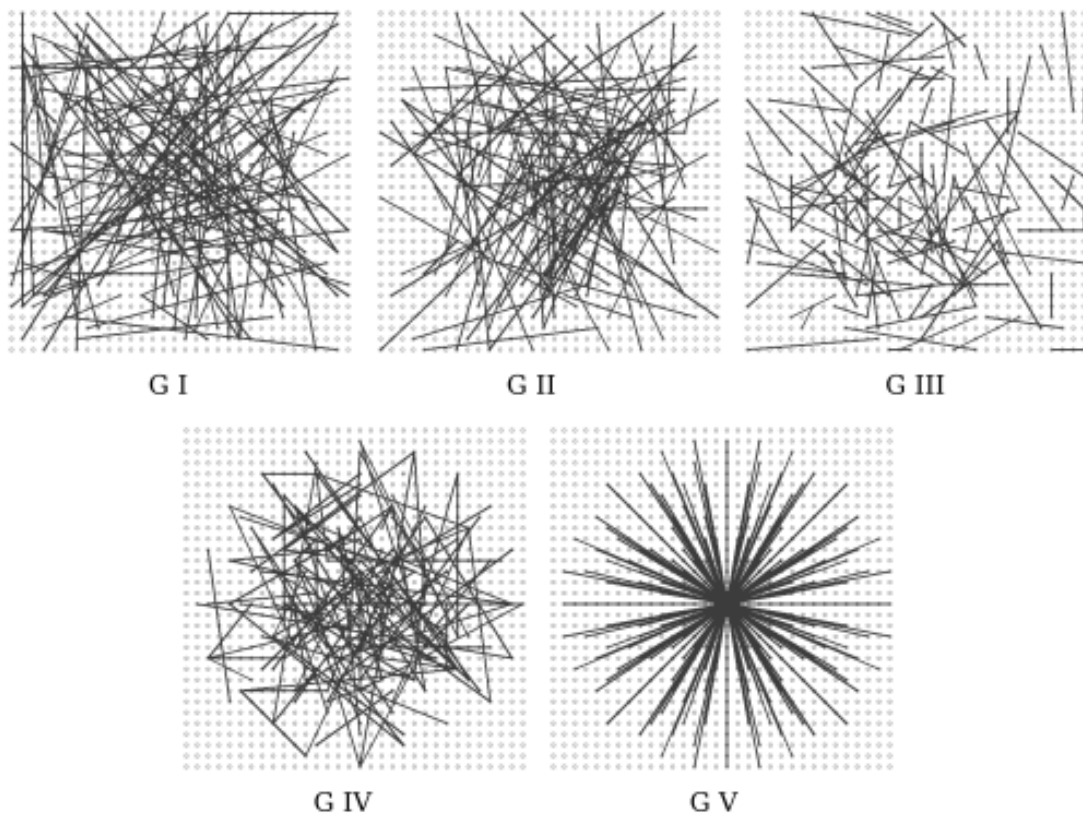


Figura 2.14: Diferença entra a escolha dos locais de teste BRIEF.(Fonte: [Calonder et al., 2010])

2.6 ORB: Oriented FAST and Rotated BRIEF

Em [Ruble et al., 2011], os autores propõem um descritor baseado em BRIEF, que seja rápido, invariante à rotação, resistente à ruídos e que sirva como uma alternativa para os já conhecidos SIFT "Scale-invariant feature transform" e SURF "Speeded up robust features".

O primeiro passo do algoritmo ORB, é utilizar o algoritmo FAST para fazer a detecção dos pontos de interesse da imagem. O FAST usa como parâmetro o limite da intensidade entre o pixel central e seus vizinhos em um anel circular sobre o centro. A versão utilizada pelos autores é o FAST-9 que olha apenas para 9 vizinhos em um anel circular sobre o centro.

Tendo os pontos de interesse identificados, é aplicado o algoritmo de Harris, que atribui um escore a estes pontos. Para N pontos chaves, um limite suficientemente baixo é definido para obter mais de N pontos. Estes pontos são ordenados de acordo com a medida de Harris e os N pontos superiores são escolhidos. A pirâmide de escala da imagem é utilizada para produzir as características do FAST em cada nível da mesma.

Para calcular a orientação do canto, os autores utilizam a centroide de intensidade, a qual assume que a intensidade de um canto está deslocada do seu centro, e este vetor pode ser utilizado para imputar uma orientação.

Na parte da descrição, os autores utilizam como base o BRIEF para calcular as strings binárias. Porém, é utilizado uma variação chamada de BRIEF dirigido, que adiciona uma direção ao BRIEF com base na orientação dos pontos de interesse. O BRIEF é construído considerando um conjunto de testes binários de intensidade. Levando em conta a região de interesse de uma imagem suavizada p , um teste binário τ é definido como:

$$\tau(p; x; y) = \begin{cases} 1 : p(x) < p(y) \\ 0 : p(x) \geq p(y) \end{cases} \quad (2.9)$$

Onde $p(x)$ é a intensidade do ponto p na localização x . A característica é definida como um vetor de n testes binários:

$$f_n(p) = \sum_{1 \leq i \leq n} u^{2-i} \tau(p, x_i, y_i) \quad (2.10)$$

O método de distribuição do BRIEF utilizado pelos autores foi a distribuição Gaussiana em torno do centro da área de interesse.

Para tornar o resultado do BRIEF invariante à rotação, os autores direcionam o BRIEF de acordo com a orientação dos pontos de interesse através dos seguintes métodos:

Para cada conjunto de características de N testes binários na localização (x_i, y_i) , uma matriz de $2 \times n$ é definida.

$$S = \begin{bmatrix} x_1, \dots, x_n \\ y_1, \dots, y_n \end{bmatrix} \quad (2.11)$$

Empregando a orientação θ dos pontos e a correspondente rotação da matriz R_θ , é construído uma versão direcionada S_θ de S :

$$S_\theta = R_\theta S \quad (2.12)$$

O operador do BRIEF direcionado fica da seguinte forma:

$$g_n(p, \theta) = f_n|(x_i, y_i) \in S_\theta \quad (2.13)$$

O ângulo é discretizado para incrementar $27\pi/30$ (12 graus) e construir uma tabela de pesquisa de padrões BRIEF pré computados. Na medida que a orientação θ do ponto de interesse é consistente através das visualizações, o conjunto correto de pontos S_θ é utilizado para calcular seu descritor.

A imagem 2.15 mostra o ORB reconhecendo objetos a partir de uma imagem de referencia, que é apresentado no lado direito da figura. No lado esquerdo da figura é mostrado os pontos de interesse identificados da imagem de referencia e a correspondência destes pontos em uma imagem onde o objeto está inserido. A imagem 2.16 mostra a extração de pontos de interesse utilizando o ORB. Os pontos destacados em verde são os pontos de interesse encontrados pelo algoritmo.

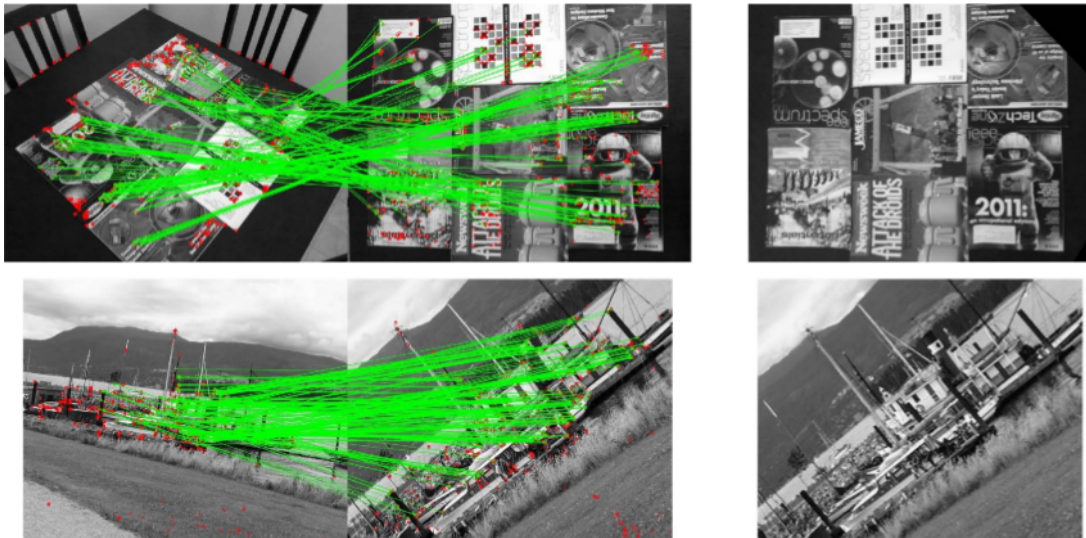


Figura 2.15: Exemplo de reconhecimento de objetos com o ORB(Fonte: [Rublee et al., 2011]).

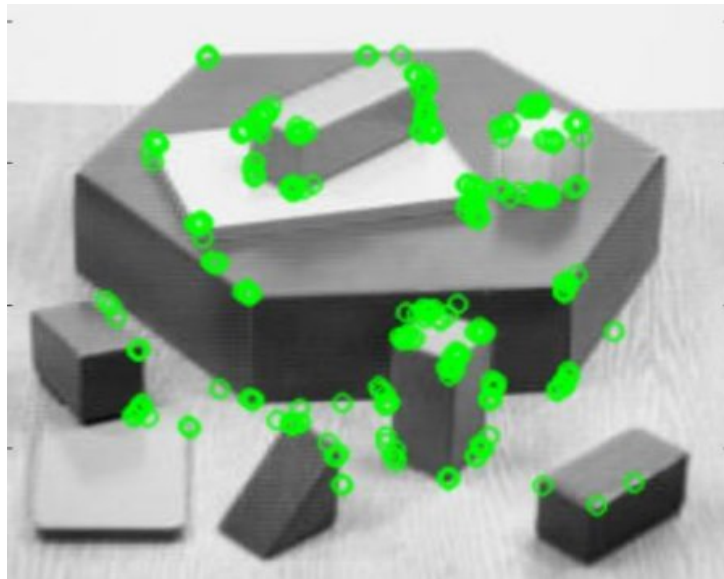


Figura 2.16: Extração de pontos de interesse utilizando o ORB(Fonte: http://docs.opencv.org/3.0-beta/_images/orb_kp.jpg).

2.7 Hamming Distance

A distância de *Hamming* entre duas strings de mesmo comprimento é o número de posições nas quais elas diferem entre si. Ela corresponde ao menor número de substituições necessárias para transformar uma *string* em outra, ou o número de erros que transformaram uma na outra [Hamming, 1950].

Seguem alguns exemplos do cálculo da distancia de Hamming:

A distancia de *Hamming* entre "elabore" e "melhore" é 4.

A distancia de *Hamming* entre 2173896 e 2233796 é 3.

A distancia de *Hamming* entre 11011 e 10011 é 1.

Neste terceiro exemplo, se a cadeia de bits A (11011) fosse emitida e chegasse ao receptor como B (10011), poderia ser usada a operação XOR da seguinte maneira:

$$11011 \text{ XOR } 10011 = 01000$$

A quantidade de bits encontrados nessa operação, é a distância de *Hamming* entre a palavra transmitida e a recebida. Deste modo, conclui-se que a distância de *Hamming* entre as strings do exemplo é 1, pois apenas 1 bit foi encontrado após a operação XOR.

Hamming Distance é empregado neste trabalho para medir a diferença entre duas strings binárias produzidas pelo descritor de características. A operação é muito eficiente e rápida, porque pode ser executada através de instruções em linguagem de máquina ou utilizar uma operação XOR seguido de uma contagem de bits [Muhammad, 2015].

2.8 Considerações

Para fazer o reconhecimento de um objeto em uma imagem, uma alternativa é realizar a extração de pontos de interesse desta imagem, fazer a descrição destes pontos e compará-los com a base de dados. No desenvolvimento do aplicativo proposto neste trabalho foi utilizado o *ORB* (*Oriented FAST and Rotated BRIEF*) para realizar o processo de extração e descrição dos pontos de interesse. O motivo para escolher este algoritmo, foi em virtude de algumas vantagens que o ORB tem em comparação aos outros algoritmos do mesmo gênero, como por exemplo, ser invariante a rotação da imagem, além de ter um custo computacional médio. Já no processo de comparação da imagem com o banco de dados, foi utilizado o algoritmo *Hamming Distance*, o qual compara as strings binárias geradas pelo algoritmo ORB.

Capítulo 3

Desenvolvimento

Este capítulo explica como os algoritmos são utilizados no desenvolvimento do aplicativo. O aplicativo é destinado a *smartphones* que rodam Android, um sistema operacional *open source* baseado em kernel linux, projetado principalmente para dispositivos móveis e, atualmente é desenvolvido pela empresa de tecnologia Google [Google, 2016a] e [Google, 2016c]. A principal IDE para desenvolvimento de aplicativos para o Android é o *Android Studio*, também desenvolvido pela empresa de tecnologia Google. [Google, 2016b]. Os principais sensores utilizados foram a câmera e o GPS. A câmera é utilizada para realizar o recebimento das imagens para a comparação com a base de dados do aplicativo. Já o GPS, nos fornece a localização do usuário que, é utilizada para guiar a busca na base de dados do aplicativo; excluindo assim a necessidade de comparar cada imagem recebida pela câmera com a base de dados inteira.

3.1 OpenCV

Todos os algoritmos utilizados no aplicativo foram retirados da biblioteca *OpenCV* (*Open Source Computer Vision Library*), uma biblioteca aberta de Visão Computacional e Aprendizagem de Máquina. A mesma foi construída para fornecer infra-estrutura comum nas aplicações de Visão Computacional e acelerar o uso de percepção de máquina em produtos comerciais.

O OpenCV tem mais de 2500 algoritmos otimizados, o que inclui um conjunto abrangente de algoritmos clássicos de visão computacional de aprendizado de máquina. Estes algoritmos são utilizados para detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, unir imagens para produzir outra imagem de alta resolução de uma cena inteira, encontrar imagens similares de um banco de imagens, remover olhos vermelhos de imagens tiradas com flash, seguir o movimento dos olhos, reconhecer um cenário e estabelecer marcadores para sobrepor-lo com Realidade Aumentada, entre outros [itseez, 2016].

3.2 Identificando um edifício

O reconhecimento de um edifício começa no momento em que a câmera do celular é apontada para o mesmo. Para reconhecer um objeto, ou no caso desse trabalho, um edifício, é necessário ter ao menos uma imagem de referência previamente tirada e carregada na aplicação. O algoritmo basicamente irá comparar cada *frame*(imagem) da câmera com as imagens de referência na tentativa de encontrar uma correspondência entre elas.

Existem diversos métodos de reconhecimento de objetos dentro de uma imagem. Este processo pode ser realizado através de algoritmos simples que reconhecem a forma geométrica do objeto, ou até mesmo, seu contorno; até algoritmos mais elaborados que decompõem o objeto em várias partes e tenta fazer o reconhecimento individual de cada uma delas. O método utilizado neste trabalho faz a extração e descrição dos pontos de interesse das imagens para compará-las por meio destes pontos.

Para fazer a comparação entre um *frame* da câmera e a imagem de referência, é necessário extrair os pontos de interesse e fazer a descrição desses pontos em ambas as imagens. Os processo de extração e descrição dos pontos de interesse da imagem de referência é feito uma única vez, apenas quando o usuário se aproxima pela primeira vez do edifício. O aplicativo utiliza o GPS do aparelho para saber sua localização.

Já o processo de extração e descrição dos pontos de interesse dos *frames* da câmera é necessário realizar diversas vezes durante a execução do aplicativo pois, cada *frame* da câmera é diferente e é jogado fora após a comparação com as imagens de referência, encontrando alguma correspondência entre elas ou não.

O algoritmo utilizado para fazer a extração e descrição dos pontos de interesse é o ORB (*Oriented FAST and Rotated BRIEF*), que é uma alternativa para os algoritmos *SIFT*(*Scale-Invariant Feature Transform*) [Lowe, 2004] e *SURF*(*Speeded Up Robust Features*) [Bay et al., 2006] que, têm exatamente a mesma função do ORB, sendo até um pouco mais eficientes em sua tarefa, e por isso, são a preferência de muitos autores para este tipo de aplicação. Porém, ambos o SIFT e SURF são patenteados. Já o ORB é aberto e livre para uso e por esse motivo foi escolhido no desenvolvimento deste trabalho. Com os pontos de interesses extraídos e tendo feito sua descrição, o algoritmo *Hamming Distance* é usado para verificar a correspondência entre as imagens.

A busca no banco de dados da aplicação por um edifício que corresponda ao *frame* da câmera é através da força bruta, ou seja, ele compara o *frame* com todos os edifícios contidos na base de dados. A medida que a base de dados aumenta, esse processo pode se tornar custoso e até mesmo inviável. Inicializar as imagens de referência e calcular seus pontos de interesse no momento em que a aplicação é iniciada, já estava se mostrando ser um processo extremamente lento. Para resolver esse problema, foi optado pelo uso do GPS contido no aparelho. Cada edifício tem associado um par de coordenadas geográficas (latitude, longitude). Assim, tanto o

calculo dos pontos de interesse de uma imagem de referência, quanto a comparação com o *frame* vindo da câmera, são executados apenas quando o usuário se aproxima do edifício em questão.

Após todo o processo de reconhecimento do edifício, caso ele encontre algum edifício na base de dados que combine com a imagem vinda da câmera, será exibido na tela do celular um conjunto de informações referentes aquele edifício como mostrado na figura 3.1. Essas informações são referentes ao seu nome, endereço, site e telefone.

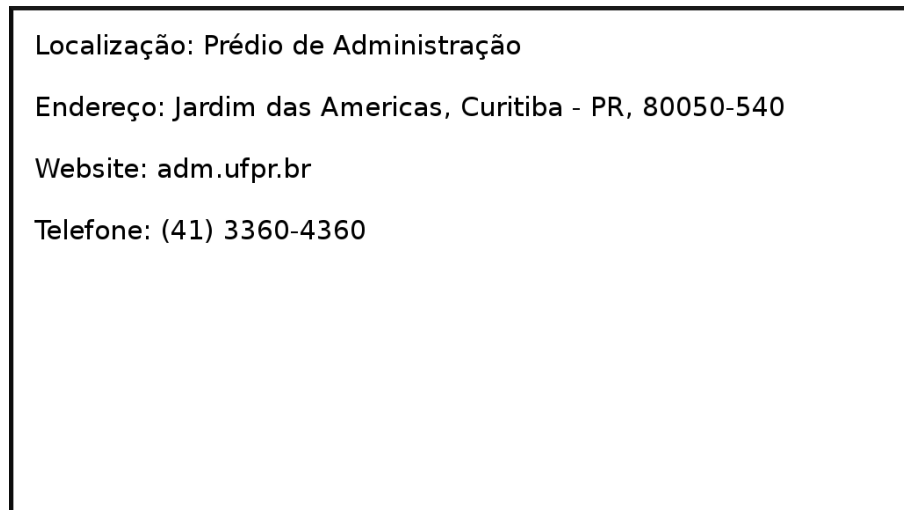


Figura 3.1: Exemplo da tela de informações exibida (Fonte: Autor)

Capítulo 4

Implementação

Neste capítulo é explicado a implementação dos algoritmos no aplicativo e são apresentados detalhes sobre os objetos e classes utilizadas no processo. A imagem 4.1 mostra um panorama geral de todas as classes utilizadas no desenvolvimento do aplicativo e quais são os relacionamentos existente entre estas classes. A primeira classe é a *ImageDescriptor* que armazena todas as informações de uma dada imagem, seja essa imagem de referência ou uma imagem da câmera. Esta classe é responsável por chamar as funções do OpenCV que executam os algoritmos de extração e descrição dos pontos de interesse da imagem. Ela também chama a função do OpenCV que executa o algoritmo de comparação entre duas imagens. A próxima classe é a *Build*, que representa um edifício na base de dados do aplicativo. Nesta classe encontramos as coordenadas geográficas do edifício em questão e um vetor de objetos da classe *ImageDescriptor* para representar as imagens de referência do mesmo. Já a classe *MainActivity* é responsável por iniciar a biblioteca OpenCV e executar o algoritmo que compara os *frames* da câmera com as imagens de referência na tentativa de identificar um edifício.

Existem três classes que herdam da classe *Fragment* do *Android* e são responsáveis por exibirem as informações na tela do *smartphone*. A primeira classe é a *InitFragment*, responsável por exibir a tela de inicialização do aplicativo. A classe *cameraFragment* é responsável por exibir os *frames* da câmera em tempo real. E a classe *ShowInformationsFragment* é responsável por exibir as informações do edifício identificado.

A seguir, é explicado detalhadamente o código de cada classe da aplicação, mostrando seus atributos e métodos e também, o processo de extração e descrição dos pontos de interesse e a identificação de um edifício na base de dados.

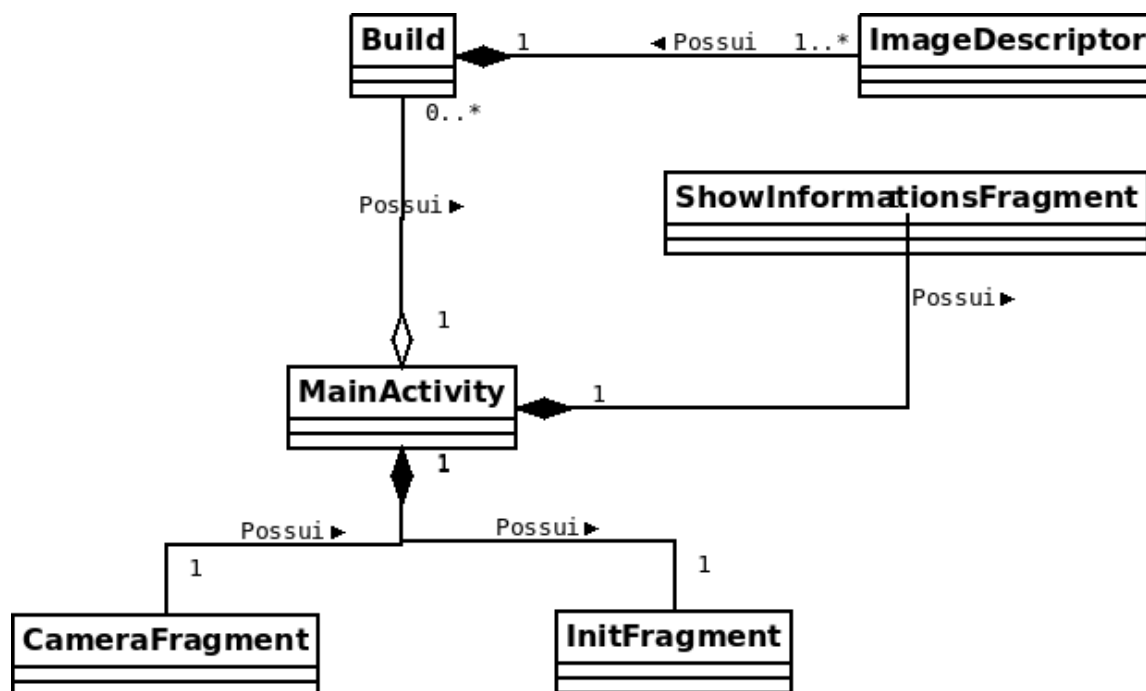


Figura 4.1: Diagrama com as classes do aplicativo e o relacionamento entre elas (Fonte: Autor)

4.1 ImageDescriptor

Os processos de extração, descrição e comparação dos pontos de interesse foram encapsulados dentro desta classe. Cada objeto dessa classe representa uma imagem, seja ela um *frame* da câmera ou uma imagem de referência. Esta classe é responsável por guardar as principais informações de uma imagem. Tem os objetos retirados da biblioteca OpenCV e que são necessários para a execução do algoritmo.

O primeiro atributo é uma matriz utilizada para guardar uma versão da imagem em tons de cinza.

```
1 Mat imgGray = new Mat();
```

Outro atributo é o tipo `MatOfKeyPoint` para guardar os pontos de interesse que serão calculados pelo ORB.

```
1 MatOfKeyPoint imgKeypoints = new MatOfKeyPoint();
```

Um atributo do tipo `imgDescriptor` guarda as strings binárias calculadas também pelo algoritmo ORB, porém na parte de descrição dos pontos de interesse.

```
1 Mat imgDescriptor = new Mat();
```

Devido a necessidade de calcular os quatro cantos da imagem, foi utilizada uma matriz para verificar se a imagem de referência está contida ou não dentro de um dado *frame* da câmera.

```
1 Mat imgCorners = new Mat(4, 1, CvType.CV_32FC2);
```

O tipo `CvType` define a profundidade da cor, número de canais e o layout do canal na imagem. Cada ponto é representado como uma matriz de 32-bits floats com dois canais.

Para representar as correspondências entre as imagens de referência e da câmera foi utilizado o objeto do tipo `MatOfDMatch`.

```
1 MatOfDMatch matches = new MatOfDMatch();
```

A classe tem os objetos necessários para realizar a extração e descrição dos pontos de interesse através do algoritmo ORB. Esses objetos são do tipo `FeatureDetector`, que faz a extração dos pontos de interesse e `DescriptorExtractor` responsável pela descrição destes pontos. Também é necessário um objeto do tipo `DescriptorMatcher` que utiliza o algoritmo *Hamming Distance* para calcular a correspondência entre as strings binárias geradas pelo passo de descrição dos pontos de interesse.

```
1 FeatureDetector featureDetector;  
2 DescriptorExtractor descriptorExtractor;  
3 DescriptorMatcher descriptorMatcher;
```

O processo de extração e descrição dos pontos de interesse de cada imagem é feito pelo construtor da classe `ImagemDescriptor`, que recebe como parâmetros um inteiro que representa a imagem de referência, o contexto da aplicação e uma matriz caso a imagem que se esteja instanciando seja a um *frame* da câmera.

Se a imagem não for da câmera, então temos que carregar a imagem de referência vinda dos recursos do aplicativo. Ela é carregada no formato BGR (blue, green, red), por motivo de eficiência.

```
1 img = Utils.loadResource(context, referenceImageID,  
2                               Imgcodecs.CV_LOAD_IMAGE_COLOR);
```

O próximo passo é calcular uma versão da imagem em tons de cinza, independentemente se a imagem é da câmera ou de referência; e então executar o processo de extração e descrição dos pontos de interesse. A extração dos pontos de interesse utiliza a imagem em tons de cinza e tem como resultado a matriz com os pontos de interesse. Já o processo de descrição, utiliza os pontos de interesse obtidos no passo anterior, juntamente com a imagem em tons de cinza e produz uma outra matriz com as strings binárias que representam o processo de descrição da imagem.

```
1 Imgproc.cvtColor(img, this.imgGray, Imgproc.COLOR_BGR2GRAY);  
2 featureDetector.detect(this.imgGray, this.imgKeypoints);  
3 descriptorExtractor.compute(this.imgGray, this.imgKeypoints,  
4                             this.imgDescriptor);
```

E por fim, ainda no construtor da classe, caso a imagem seja de referência, é chamado o método `findCorners()`, que tem duas funções: A primeira é caso a imagem em questão seja uma imagem de referência. Então, este método calcula os quatro cantos da imagem.

```

1 if (referenceImg == null){
2     this.imgCorners = new Mat(4, 1, CvType.CV_32FC2);
3     this.imgCorners.put(0, 0, 0.0, 0.0);
4     this.imgCorners.put(1, 0, this.imgGray.cols(), 0.0);
5     this.imgCorners.put(2, 0, this.imgGray.cols(), this.imgGray.rows());
6     this.imgCorners.put(3, 0, 0.0, this.imgGray.rows());
7     return;
8 }

```

A segunda função deste método é caso a imagem seja um *frame* da câmera. Primeiro, ele irá verificar se há uma quantidade mínima de pontos de interesse identificados na imagem.

```

1 List<DMatch> matchesList = this.matches.toList();
2 if (matchesList.size() < 4) {
3     return;
4 }

```

Caso haja um número mínimo de pontos de interesse, calcula a distância mínima entre estes pontos.

```

1 double minDist = Double.MAX_VALUE;
2 for (DMatch match : matchesList) {
3     double dist = match.distance;
4     if (dist < minDist) {
5         minDist = dist;
6     }
7 }

```

Verifica-se então, se a distância mínima é maior que um certo limite.

```

1 if (minDist > 25.0){
2     return;
3 }

```

O próximo passo é utilizar a distância mínima para identificar bons pontos de interesse.

```

1 ArrayList<Point> goodReferencePointsList = new ArrayList<>();
2 ArrayList<Point> goodScenePointsList = new ArrayList<>();
3 double maxGoodMatchDist = 1.75 * minDist;
4 for (DMatch match : matchesList) {
5     if (match.distance < maxGoodMatchDist) {
6         goodReferencePointsList.add(referenceKeypointsList.
7                                     get(match.trainIdx).pt);
8         goodScenePointsList.add(sceneKeypointsList.get(match.queryIdx).pt);
9     }
10 }

```

E verificar se ainda há um mínimo de pontos para calcular a matriz de homografia.

```

1 if (goodReferencePointsList.size() < 4 || goodScenePointsList.size() < 4) {
2     return;
3 }

```

O próximo passo é converter as matrizes de correspondência da imagem de referência e da câmera para o formato MatOfPoint2f, como requerido pela função Calib3d.findHomography e encontrar a matriz de homografia.

```

1 Mat homography=Calib3d.findHomography(goodReferencePoints,goodScenePoints);

```

Utiliza-se então, a matriz de homografia para projetar os cantos da imagem de referência na imagem de câmera.

```

1 Core.perspectiveTransform(referenceImg.imgCorners, candidateSceneCorners,
2                             homography);

```

Converte-se os cantos da imagem da câmera para o formato inteiro, como requerido pela função Imgproc.isContourConvex.

```

1 candidateSceneCorners.convertTo(intSceneCorners, CvType.CV_32SC2);

```

E para finalizar, verifica se os cantos formam um polígono convexo. Caso contrário, se os cantos formam um polígono côncavo, o resultado da detecção é inválido porque nenhuma perspectiva real pode tornar os cantos de uma imagem retangular parecidos com um polígono côncavo. E atribui verdadeiro para uma variável que identifica que a imagem da câmera e de referência são compatíveis.

```

1 if (Imgproc.isContourConvex(intSceneCorners)) {
2     candidateSceneCorners.copyTo(this.imgCorners);
3     this.exibeInformacao = true;
4 }

```

Nesta classe, ainda temos uma função responsável por verificar as correspondências entre as imagens da câmera e de referência. Essa função é a tryMatcher, que recebe como parâmetro a imagem de referência e calcula a correspondência dos pontos de interesse vindos do descritor. Para calcular a correspondência entre as imagens é utilizado a matriz com os descritores da imagem de referência e a matriz com os descritores da imagem da câmera, tendo como resultado uma matriz que guarda essas correspondências. No final deste método, é chamado o método imageCorners, passando como parâmetro a imagem de referência e é neste momento em que o método irá executar a comparação entre as duas imagens como explicado acima.

```

1 descriptorMatcher = DescriptorMatcher.create(
2     DescriptorMatcher.BRUTEFORCE_HAMMING);
3 descriptorMatcher.match(this.imgDescriptor,referenceImg.imgDescriptor,
4     this.matches);

```

4.2 Build

Outra classe importante para o processo de identificação de um prédio é a classe Build. Esta classe representa um prédio propriamente dito, e tem como atributos um par de coordenadas geográficas para identificar a localização do prédio através do sistema de GPS do aparelho, a distância máxima em que o usuário deve estar do prédio para que seja possível o reconhecimento do mesmo e quatro strings para armazenar informações de nome do edifício, endereço, site e telefone. Todas estas informações são inicializadas pelo construtor da classe. E também um vetor de objetos da classe ImageDescriptor para armazenar as imagens de referência do edifício. Como as informações de cada imagem são armazenadas na classe ImageDescriptor, ou seja, cada instancia da classe ImageDescriptor representa uma imagem de referência. Cada edifício pode ter várias imagens de referência armazenadas em sua base de dados. Logo, é necessário um vetor para armazenar todas elas.

```

1  boolean init;
2      int dist;
3      double latitude;
4      double longitude;
5      Vector<ImageDescriptor> imageDescriptors;
6      String local;
7      String end;
8      String site;
9      String fone;
10
11  Build(double lat,double lng,int d,String l,String e,String s,String f){
12      this.init = false;
13      this.latitude = lat;
14      this.longitude = lng;
15      this.dist = d;
16      this.local = l;
17      this.end = e;
18      this.site = s;
19      this.fone = f;
20  }
```

Para fazer a inicialização das imagens do edifício temos um método chamado InitImages, o qual recebe um vetor de inteiros que representa as imagens nos recursos da aplicação, o contexto da aplicação e uma matriz caso estejamos inicializando uma imagem da câmera. Este método inicializa um objeto da classe ImageDescriptor caso a imagem que esteja sendo inicializada seja da câmera, ou um vetor de objetos da classe ImageDescriptor caso esteja inicializando as imagens de referência.

```

1  void initImages(Vector<Integer> referencesImagesIds, Context context,
2                  Mat image) throws IOException{
3      this.init = true;
```



```

4      imageDescriptors = new Vector<>();
5      if (image != null){
6          ImageDescriptor img = new ImageDescriptor(0,context,image);
7          imageDescriptors.add(img);
8      } else {
9          for (int i = 0; i < referencesImagesIds.size(); i++) {
10             ImageDescriptor img = new ImageDescriptor(
11                 referencesImagesIds.get(i),context,null);
12             imageDescriptors.add(img);
13         }
14     }
15 }

```

4.3 MainActivity

A atividade principal é responsável por agrupar todas as classes, inicializar o OpenCV e rodar a aplicação. É a *thread* principal do programa.

Para iniciar a aplicação, primeiro há a necessidade de instanciar alguns objetos dentro da atividade principal para obter acesso à câmera do aparelho através das funções fornecidas pelo OpenCV.

O primeiro objeto utilizado é o CameraBridgeViewBase da biblioteca OpenCV. Este objeto irá avisar para a aplicação que ela deverá utilizar a câmera do celular, e será mostrada no JavaCameraView definido no arquivo de layout activity_main.xml.

```

1 <org.opencv.android.JavaCameraView
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:id="@+id/cameraView"/>

```

E o código necessário para inicializar o OpenCV assincronamente através do gerenciador do OpenCV Android é o seguinte:

```

1 private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
2     @Override
3     public void onManagerConnected(int status) {
4         switch (status) {
5             case LoaderCallbackInterface.SUCCESS: {
6                 Log.i("OpenCV", "OpenCV loaded successfully");
7                 mOpenCvCameraView.enableView();
8                 mOpenCvCameraView.setOnTouchListener(MainActivity.this);
9             }
10            break;
11
12            default: {
13                super.onManagerConnected(status);

```

```

14     }
15     break;
16     }
17 }
18 };

```

Há a necessidade de implementar algumas funções para fazer o gerenciamento da câmera. A primeira função é a `onCameraViewStarted` que, apenas instancia um objeto do tipo `Mat()`, ou seja, uma matriz, para receber a imagem da câmera no momento em que a aplicação é iniciada.

E a função `onCameraViewStopped` libera essa matriz quando a visualização é parada.

```

1 @Override
2 public void onCameraViewStarted(int width, int height) {
3     cameraImg = new Mat();
4 }
5
6 @Override
7 public void onCameraViewStopped() {
8     cameraImg.release();
9 }

```

Outra função muito importante é a `onCameraFrame`. Essa função dá acesso a cada *frame* da câmera e é nessa função em que é executado todo o processo de comparar o *frame* da câmera com as imagens do banco de dados.

Primeiro é necessário armazenar o *frame* através do comando:

```

1 cameraImg = inputFrame.rgba();

```

Então o algoritmo faz uma verificação se não está sendo exibida nenhuma informação na tela do aparelho. Em caso positivo, ele apenas retorna o *frame* e sai da função. Em caso negativo, ele irá instanciar um objeto do tipo `Build` para armazenar o *frame* da câmera e fazer o cálculo de extração e descrição dos pontos de interesse deste *frame*.

```

1 try {
2     cameraImageDescriptor = new Build(myLatitude, myLongitude, 0, "", "", "", "");
3     cameraImageDescriptor.initImages(null, MainActivity.this, cameraImg);
4 } catch (IOException e) {
5     Log.e("ERRO", "Erro ao iniciar imagem da camera");
6     e.printStackTrace();
7 }

```

Após a inicialização da imagem da câmera, ele percorre um vetor contendo todos os edifícios da base de dados. Para cada edifício, é medido a distância entre o usuário e este edifício. Em caso negativo, ele tenta outro edifício e, em caso positivo, é verificado se aquele edifício já foi inicializado, ou seja, se os pontos de interesse já foram extraídos e descritos pelo algoritmo. Em caso negativo, o edifício é inicializado através do método `initImages` da classe `Build`. Tendo

as imagens inicializadas, para cada imagem de referência do prédio, ou seja, para cada objeto da classe ImageDescriptor dentro da classe Build, é feita a comparação entre a imagem e o *frame* da câmera. E se ele encontrar alguma correspondência, a aplicação inicializa um *fragment* com as informações daquele edifício.

```

1  for (int k = 0; k < predios.size(); k++){
2      if (distFrom(myLatitude, myLongitude, predios.get(k).latitude,
3                  predios.get(k).longitude) < predios.get(k).dist){
4          if (!predios.get(k).init){
5              try {
6                  predios.get(k).initImages(referenceImgs.get(k),
7                                              MainActivity.this,null);
8              } catch (IOException e) {
9                  Log.e("ERROR", "Erro ao iniciar prédio de id: " +
10                      Integer.toString(k));
11                  e.printStackTrace();
12              }
13          }
14          test = false;
15          for(int i = 0; i < referenceImgs.get(k).size(); i++){
16              cameraImageDescriptor.imageDescriptors.get(0).tryMatcher(
17                  predios.get(k).imageDescriptors.get(i));
18              test = test || cameraImageDescriptor.imageDescriptors
19                  .get(0).exibeInformacao;
20          }
21          if (test) {
22              ShowInformationsFragment showFragment
23                  = new ShowInformationsFragment();
24              showFragment.setString(predios.get(k).local, predios.get(k).end,
25                                   predios.get(k).site, predios.get(k).fone);
26              FragmentTransaction transaction = getSupportFragmentManager()
27                  .beginTransaction();
28              transaction.replace(R.id.fragment_container, showFragment);
29              transaction.addToBackStack(null);
30              transaction.commit();
31              exibindoInformacao = true;
32              return cameraImg;
33          }
34      }
35  }

```

A distância entre duas coordenadas geográficas é medida através da função distFrom.

```

1  public static double distFrom(double lat1, double lng1,
2                                double lat2, double lng2) {
3      double earthRadius = 6371000; //meters
4      double dLat = Math.toRadians(lat2-lat1);
5      double dLng = Math.toRadians(lng2-lng1);

```

```

6      double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
7          Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
8          Math.sin(dLng/2) * Math.sin(dLng/2);
9      double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
10
11      return (float) (earthRadius * c);
12  }

```

As alterações das coordenadas atuais do aparelho são medidas pela função `onLocationChanged`, que é um *listening*, que atualiza as coordenadas do aparelho sempre que há alguma mudança.

```

1  @Override
2  public void onLocationChanged(Location location) {
3      myLatitude = location.getLatitude();
4      myLongitude = location.getLongitude();
5  }

```

Há também uma função para verificar o toque na tela. Após ser exibida as informações na tela do celular, há a necessidade de dar um toque na tela para que ela volte a exibir as imagens da câmera e prosseguir com o processo de identificação dos edifícios.

```

1  @Override
2  public boolean onTouch(View view, MotionEvent motionEvent) {
3      CameraFragment mCameraFragment = new CameraFragment();
4      FragmentTransaction transaction =
5          getSupportFragmentManager().beginTransaction();
6      transaction.replace(R.id.fragment_container, mCameraFragment);
7      transaction.addToBackStack(null);
8      transaction.commit();
9      exibindoInformacao = false;
10     return false;
11 }

```

4.4 InitFragment

Esta classe herda da classe *Fragment* do *Android* e é responsável por exibir a tela de início quando o aplicativo é iniciado. Esta classe tem os métodos `onCreate`, responsável por iniciar o *fragment*, `onAttach` responsável por associar o *fragment* a uma *Activity*, `onDetach` responsável por desassociar o *fragment* da *Activity* e `onCreateView` responsável por criar e retornar a hierarquia de visões associada ao *fragment*.

```

1  @Override
2  public void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4  }

```

```

5
6 @Override
7 public View onCreateView(LayoutInflater inflater, ViewGroup container,
8                             Bundle savedInstanceState) {
9     return inflater.inflate(R.layout.fragment_init, container, false);
10 }
11
12 @Override
13 public void onAttach(Context context) {
14     super.onAttach(context);
15 }
16
17 @Override
18 public void onDetach() {
19     super.onDetach();
20 }

```

O `initFragment` tem um arquivo de *layout* associado. Neste arquivo, há um `TextView` para exibir um texto de inicialização da aplicação.

```

1 <TextView
2     android:text="@string/start"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:textSize="200sp"
6     android:textColor="#ffffff"
7     android:textAlignment="center"
8     android:id="@+id/textView" />

```

4.5 CameraFragment

Esta classe é responsável por exibir os *frames* da câmera na tela do celular. É composta pelos mesmos métodos da classe `InitFragment`, porém há uma diferença no arquivo de *layout* associado a ela. O *layout* desta classe contém o elemento `JavaCameraView` que exibe os *frames* da câmera na tela do celular.

```

1 <org.opencv.android.JavaCameraView
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:id="@+id/cameraView"/>

```

4.6 ShowInformationsFragment

Esta é uma classe que herda da classe `Fragment` e é utilizada pra exibir as informações na tela do celular. Além dos métodos de `onCreate`, `onAttach` e `onDetach` herdados de `Fragment` e que são necessários para fazer a exibição do fragment na tela do celular, há também os métodos `setString` responsável por receber as informações que deverão ser exibidas, e também o método `onCreateView`, responsável por criar a exibição com as informações e mostrá-las na tela do celular.

```

1 public void setString(String t1,String t2,String t3,String t4) {
2     s1 = t1;
3     s2 = t2;
4     s3 = t3;
5     s4 = t4;
6 }

1 @Override
2     public View onCreateView(LayoutInflater inflater, ViewGroup container,
3                             Bundle savedInstanceState) {
4         ViewGroup rootView = (ViewGroup) inflater.inflate(
5             R.layout.fragment_show_informations, container, false);
6
7         TextView local = (TextView) rootView.findViewById(R.id.local);
8         local.setText("Localização:" + this.s1);
9
10        TextView end = (TextView) rootView.findViewById(R.id.end);
11        end.setText("Endereço:" + this.s2);
12
13        TextView site = (TextView) rootView.findViewById(R.id.site);
14        site.setText("Website:" + this.s3);
15
16        TextView fone = (TextView) rootView.findViewById(R.id.fone);
17        fone.setText("Telefone:" + this.s4);
18
19        return rootView;
20    }

```

No *layout* associado, existem quatro `TextView` para exibir as informações do edifício.

```

1 <TextView
2     android:layout_width="match_parent"
3     android:layout_height="359dp"
4     android:id="@+id/local" />
5
6     <TextView
7         android:layout_marginTop="20dp"
8         android:layout_width="match_parent"

```

```

9      android:layout_height="match_parent"
10     android:id="@+id/end" />
11
12     <TextView
13         android:layout_marginTop="60dp"
14         android:layout_width="match_parent"
15         android:layout_height="match_parent"
16         android:id="@+id/site" />
17
18     <TextView
19         android:layout_marginTop="80dp"
20         android:layout_width="match_parent"
21         android:layout_height="match_parent"
22         android:id="@+id/fone" />

```

4.7 Execução do Algoritmo

A figura 4.2 mostra o diagrama de sequência da execução do algoritmo. Com a aplicação em execução, a primeira ação do usuário é tocar na tela para que a aplicação mude o *fragment* que está sendo exibido na tela. A MainActivity recebe esta ação, e então inicia a câmera do aparelho através das funções do OpenCV e deixa de exibir o InitFragment para exibir o CameraFragment.

No momento em que o usuário se aproxima de um edifício pela primeira vez, a MainActivity inicializa o objeto da classe Build referente ao edifício em questão. Por sua vez, a classe Build irá inicializar as imagens de referência do edifício que são objetos da classe ImageDescriptor e executar as funções de extração e descrição dessas imagens.

Para cada *frame* da câmera, será criado um novo objeto da classe Build, e o *frame* da câmera será um objeto da classe ImageDescriptor. Neste passo, será executado os algoritmos de extração e descrição do *frame* da câmera. O *frame* será exibido na tela do *smartphone* para o usuário enquanto seus pontos de interesse são comparados com a base de dados. Encontrando um edifício correspondente, a MainActivity irá atualizar a tela do celular e mostrar o ShowInformationsFragment contendo as informações do edifício. Quando o usuário tocar novamente na tela, a MainActivity exibe o CameraFragment na tela do celular e o processo de identificação do edifício se repete.

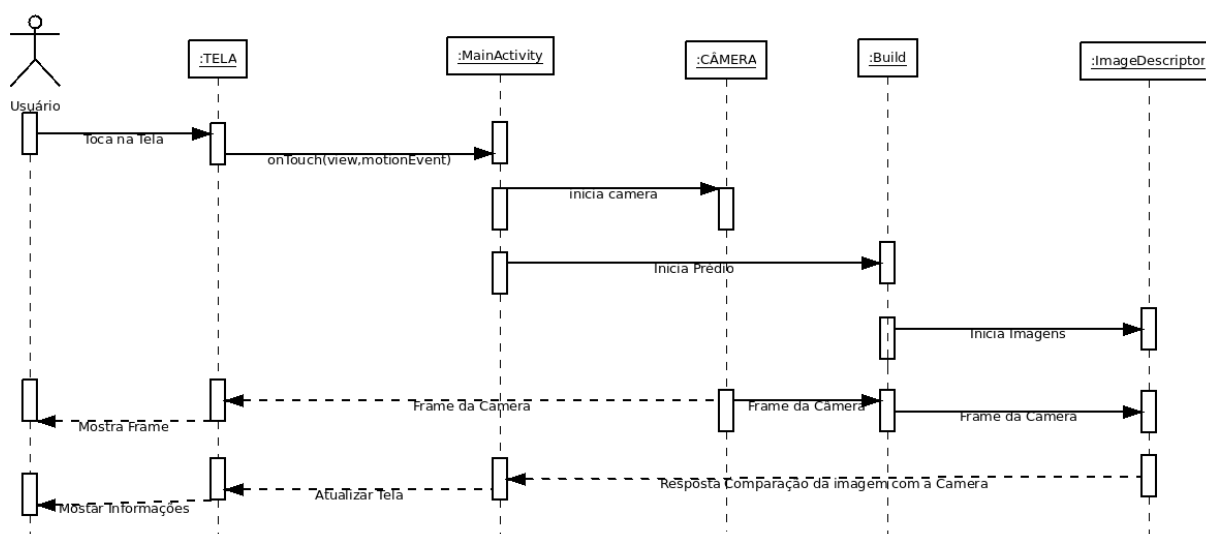


Figura 4.2: Diagrama de Sequência do Aplicativo (Fonte: Autor)

4.8 Edifícios no Banco de Dados

O aplicativo ainda está na fase de testes; e até agora foram incluídos e testados 13 edifícios em sua base de dados onde, 10 edifícios são do Campus Centro Politécnico e 3 edifícios são do Campus Jardim Botânico. A seguir são mostrados as imagens de referência utilizadas para realizar o processo de reconhecimento dos edifícios. As figuras de 4.3 à 4.12 são os edifícios do Campus Centro Politécnico e as figuras de 4.13 à 4.15 são dos edifícios do Campus Jardim Botânico.

4.8.1 Edifícios do Campus Centro Politécnico



Figura 4.3: Prédio da Administração(Fonte: Autor).



Figura 4.4: Prédio da Biblioteca(Fonte: Autor).



Figura 4.5: Prédio da Biológicas(Fonte: Autor).



Figura 4.6: Prédio das PA's(Fonte: Autor).



Figura 4.7: Prédio de Engenharia Bioprocessos e Biotecnologia (Fonte: Autor).



Figura 4.8: Prédio de Laboratório de Estudos Sedimentológicos e Petrologia Sedimentar(Fonte: Autor).



Figura 4.9: Restaurante Universitário(Fonte: Autor).



Figura 4.10: Prédio das PC's(Fonte: Autor).



Figura 4.11: Prédio das CT's(Fonte: Autor).



Figura 4.12: Prédio Departamento de Informática(Fonte: Autor).

4.8.2 Edifícios do Campus Jardim Botânico



Figura 4.13: Prédio de Engenharia Industrial Madeireira(Fonte: Autor).



Figura 4.14: Prédio de Farmácia (Fonte: Autor).



Figura 4.15: Prédio de Sociais Aplicadas(Fonte: Autor).

Capítulo 5

Resultados e Discussão

Neste capítulo são mostrados os testes realizados, quais os problemas apresentados pelos algoritmos, e quais são as dificuldades que estes problemas acarretaram no desempenho do algoritmo do aplicativo.

5.1 Testes

O método de avaliação do aplicativo foi através de percursos realizados nos Campi Centro Politécnico e Jardim Botânico verificando se os edifícios inclusos na base de dados conseguiam ou não ser reconhecidos pelos algoritmos. As imagens de referência foram tiradas em dia ensolarado e no total, o reconhecimento de cada edifício foi testado 20 vezes, 10 vezes em dia ensolarado, para testar o algoritmo em condições ideais de iluminação e 10 vezes em dia nublado para testar o algoritmo quando há mudança de iluminação no ambiente.

A seguir, a tabela 5.1 mostra os resultados obtidos. A tabela é composta por três colunas. A primeira coluna é o nome do edifício. A coluna do meio mostra a taxa de reconhecimento obtida pelo edifício em dias ensolarados e a ultima coluna mostra a taxa de reconhecimento obtida pelo edifício em dias nublados. A última linha da coluna é a taxa de reconhecimento médio que o algoritmo obteve. Este valor foi calculado através da média aritmética das taxas de reconhecimento dos edifício.

Edifício	Ensolarado	Nublado
Administração	100%	20%
Biblioteca	100%	60%
Biológicas	100%	30%
Bloco PA	100%	40%
Engenharia Bio-processos e Biotecnologia	100%	3%
Laboratório de Estudos Sedimentológicos e Petrologia Sedimentar	100%	40%
Restaurante Universitário	80%	10%
Engenharia Industrial Madeireira	100%	30%
Farmácia	80%	20%
Sociais Aplicadas	90%	40%
Departamento de Informática	70%	20%
Bloco PC	0%	0%
Bloco CT	40%	0%
Valor médio de reconhecimento	81%	26%

Tabela 5.1: Resultado obtido através dos testes (Fonte: Autor)

Na tabela 5.1, podemos ver que nas condições ideais de iluminação do ambiente, ou seja, nos dias ensolarados, a maioria dos edifícios obtiveram uma taxa de reconhecimento alta. Como vemos na tabela, o valor médio de reconhecimento dos edifícios foi de 81%. Porém, quando há variação na iluminação do ambiente, ou seja, os dias nublados, a taxa de reconhecimento é tem uma grande redução, decaindo de 81% para 26%. Isto se deve ao fato do reconhecimento do edifício ser muito dependente da imagem de referência, e alterações de iluminação acabam prejudicando o reconhecimento.

Há também os casos dos edifícios "Bloco PC" e "Bloco CT", que mesmo nas condições ideais de iluminação, ou seja, dias ensolarados, não obtiveram uma taxa de acerto aceitável. Este resultado se deve ao fato do algoritmo não conseguir realizar uma extração de pontos de interesse que seja boa para realizar o reconhecimento do edifício. Existem vários fatores para isto, o primeiro é caso o edifício contenha muitas janelas ou portas com vidro. O reflexo gerado pelo vidro gera pontos de interesse falsos. Outro fator que pode causar este problema é caso os edifícios não tenham pontos de interesse que possam ser identificados pelo algoritmo. Isto ocorre quando a estrutura do edifício é muito homogênea e não possui cantos suficientes para o reconhecimento. O mesmo problema pode ser encontrado quando há alteração na hora do dia em que o reconhecimento é feito e a posição do sol acaba gerando sombras que também acabam prejudicando o reconhecimento.

Para contornar o problema do reconhecimento quando há alteração na iluminação do ambiente, é necessário obter várias fotos de referência para cada edifício. Estas fotos devem ser tiradas em vários horários do dia e em vários dias diferentes quando há alteração de dias ensolarados e nublados. Essas fotos de referência também devem ser tiradas de vários ângulos e posições diferentes do edifício, caso se deseje obter um reconhecimento completo do edifício e não só a fachada do mesmo. Porém, devido as capacidades limitadas dos *smartphones*, é

necessário passar o processo de reconhecimento do edifício para um servidor externo que consiga lidar com o grande volume de dados que serão necessários analisar para realizar o reconhecimento. Outro problema associado a isto foi encontrado durante o processo de tirar fotos dos edifícios para serem utilizadas como base de dados da aplicação. Nem todos os prédios estão aptos para serem incluídos na aplicação. Esse problema é gerado por inúmeros objetos que ficam na frente dos edifícios, como árvores e placas, os quais acabaram atrapalhando o processo tanto de tirar as fotos de referencia do edifício, quanto no processo de identificação do mesmo. Edifícios muito próximos um dos outros ou com estacionamento na frente impossibilitaram que estes fossem incluídos.

Como resultado principal, podemos concluir que o ORB é uma algoritmo que realiza o que se propõem a fazer, porém em ambientes externos vários problemas são encontrados, problemas estes que demandarão algumas alterações futuras para o que aplicativo possa ser disponibilizado aos demais usuários.

Capítulo 6

Conclusão

Como principal resultado obtido através deste trabalho, temos uma primeira versão do aplicativo já em funcionamento. Porém, diversos problemas e dificuldades se apresentaram no processo do seu desenvolvimento. O principal deles foi que, o reconhecimento de um edifício se mostrou extremamente dependente da imagem de referência. Ou seja, qualquer modificação no ambiente, exige uma alteração da imagem de referência do edifício que se deseja reconhecer. Como foi visualizado nos testes, o reconhecimento foi de 81% nas condições adequadas de iluminação e posição do usuário. Entretanto, quando ocorreu mudança na iluminação, a taxa de acerto caiu para 26%.

Os próximos passos a serem adotados são, primeiramente estudar algoritmos que ajudem a melhorar o processo de identificação dos edifícios. Principalmente na parte que se refere a mudança de iluminação do ambiente em que o edifício está inserido, e também, para melhorar o reconhecimento de edifícios compostos por muitas janelas e/ou portas de vidro, que por causa do reflexo acabam prejudicando o algoritmo.

Outra melhoria a ser feita, é em relação a adição de novos edifícios na base de dados. Essa adição ficou dependente de alterações diretamente no código, o que torna complicado a adição de novos edifícios. Para contornar a situação, haveria a necessidade de adicionar uma nova função ao aplicativo. Essa nova função permitiria ao usuário tirar uma foto de um possível novo edifício e ali mesmo inserir as informações referentes aquele edifício. Essa melhoria porém, exigiria a criação de um servidor externo, com a função de receber e repassar tais informações para outros usuários. E devido a grande quantidade de novos edifícios inseridos desta maneira no servidor, também haverá a necessidade de transferir o processo de identificação dos edifícios para este servidor, pois as capacidades de processamento em aparelhos celulares é limitada.

Referências Bibliográficas

- [Azuma, 1997] Azuma, R. T. (1997). A survey of augmented reality. *Teleoperators and Virtual Environments*, 6:355–385.
- [Bay et al., 2006] Bay, H., Tuytelaars, T., and Gool, L. V. (2006). Surf: Speeded up robust features. In *9th European Conference on Computer Vision*.
- [Calonder et al., 2010] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief: Binary robust independent elementary features. In *In European Conference on Computer Vision*.
- [Delail et al., 2013] Delail, B. A., Weruaga, L., and Zemerly, M. J. (2013). Indoor localization and navigation using smartphones augmented reality and inertial tracking. In *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*.
- [Du et al., 2014] Du, Y., Miao, Z., and Cen, Y. (2014). Markless augmented reality registration algorithm based on ORB. In *Signal Processing (ICSP), 2014 12th International Conference on*.
- [Google, 2016a] Google (2016a). Android. https://www.android.com/intl/pt-BR_br/. Acessado em 10/10/2016.
- [Google, 2016b] Google (2016b). Android studio. <https://developer.android.com/studio/index.html>. Acessado em 10/10/2016.
- [Google, 2016c] Google (2016c). Source android. <https://source.android.com/source/index.html>. Acessado em 10/10/2016.
- [Hamming, 1950] Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell System Technical Journal*, XXIX.
- [Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. *Alvey Vision Conference*, pages 147–151.
- [itseez, 2016] itseez (2016). Opencv. <http://opencv.org/>. Acessado em 10/10/2016.
- [Lin et al., 2015] Lin, C., Wang, L., Li, D., and Zhang, M. (2015). A visual-inertial fusion based tracking system for mobile augmented reality. In *Signal Processing, Communications and Computing (ICSPCC), 2015 IEEE International Conference on*.

- [Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, page 91–110.
- [Muhammad, 2015] Muhammad, A. (2015). *OpenCV Android Programming By Example*. Packt.
- [Niantic, 2016] Niantic (2016). Pokémon go. <http://www.pokemongo.com/en-us/>. Acessado em 10/10/2016.
- [Rattnarungrot and White, 2014] Rattnarungrot, S. and White, M. (2014). A service-oriented mobile augmented reality architecture for personalized museum environments. In *Virtual Systems & Multimedia (VSMM), 2014 International Conference on*.
- [Rosten et al., 2008] Rosten, E., Porter, R., and Drummond, T. (2008). Faster and better: A machine learning approach to corner detection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 32(1):105–119.
- [Rublee et al., 2011] Rublee, E., Konolige, K., Rabaud, V., and Bradski, G. (2011). Orb: an efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on*.
- [Smith and Brady, 1997] Smith, S. M. and Brady, J. M. (1997). Susan—a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78.
- [Szeliski, 2011] Szeliski, R. (2011). *Computer Vision Algorithms and Applications*. Springer.
- [Ufkes and Fiala, 2013] Ufkes, A. and Fiala, M. (2013). A markerless augmented reality system for mobile devices. In *Computer and Robot Vision (CRV), 2013 International Conference on*.
- [Xia and Zhou, 2013] Xia, Y. and Zhou, W. (2013). A tracking and registration method based on orb and klt for augmented reality system. In *Wireless and Optical Communication Conference (WOCC), 2013 22nd*.

Apêndice A

Códigos

A.0.1 MainActivity.java

```
1 package com.example.krinski.ufprbuildingdetector;
2
3 import android.content.pm.ActivityInfo;
4 import android.content.pm.PackageManager;
5 import android.location.Location;
6 import android.location.LocationListener;
7 import android.location.LocationManager;
8 import android.support.v4.app.ActivityCompat;
9 import android.support.v4.app.FragmentActivity;
10 import android.os.Bundle;
11 import android.support.v4.app.FragmentTransaction;
12 import android.util.Log;
13 import android.view.MotionEvent;
14 import android.view.SurfaceView;
15 import android.view.View;
16 import android.view.WindowManager;
17
18 import org.opencv.android.BaseLoaderCallback;
19 import org.opencv.android.CameraBridgeViewBase;
20 import org.opencv.android.LoaderCallbackInterface;
21 import org.opencv.android.OpenCVLoader;
22 import org.opencv.core.Mat;
23
24 import java.io.IOException;
25 import java.util.Vector;
26
27 public class MainActivity extends FragmentActivity implements
28                                     LocationListener,
29                                     View.OnTouchListener,
30                                     CameraBridgeViewBase.CvCameraViewListener2 {
31
```

```

32  double myLatitude, myLongitude;
33  boolean test;
34
35  private Mat cameraImg;
36  boolean exibindoInformacao = false;
37  private CameraBridgeViewBase mOpenCvCameraView;
38  private Build cameraImageDescriptor;
39
40  Vector<Build> predios;
41  Vector<Vector<Integer>> referenceImgs;
42
43  Build admBuild,
44      bibBuild,
45      pasBuild,
46      bioBuild,
47      ruBuild,
48      desBuild,
49      labeBuild,
50      dinfBuild,
51      farmBuild,
52      socApliBuild,
53      engIndMad;
54
55  Vector<Integer> referenceImgAdmBuild;
56  Vector<Integer> referenceImgBibBuild;
57  Vector<Integer> referenceImgPasBuild;
58  Vector<Integer> referenceImgBioBuild;
59  Vector<Integer> referenceImgRuBuild;
60  Vector<Integer> referenceImgDesBuild;
61  Vector<Integer> referenceImgLabeBuild;
62  Vector<Integer> referenceImgDinfBuild;
63  Vector<Integer> referenceImgFarmBuild;
64  Vector<Integer> referenceImgSocApliBuild;
65  Vector<Integer> referenceImgEngIndMadBuild;
66
67  private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
68      @Override
69      public void onManagerConnected(int status) {
70          switch (status) {
71              case LoaderCallbackInterface.SUCCESS: {
72                  Log.i("OpenCV", "OpenCV loaded successfully");
73                  mOpenCvCameraView.enableView();
74                  mOpenCvCameraView.setOnTouchListener(MainActivity.this);
75              }
76              break;
77
78              default: {

```

```

79     super.onManagerConnected(status);
80 }
81 break;
82 }
83 }
84 };
85
86 @Override
87 protected void onCreate(Bundle savedInstanceState) {
88     super.onCreate(savedInstanceState);
89     setContentView(R.layout.activity_main);
90
91     getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
92     getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
93         WindowManager.LayoutParams.FLAG_FULLSCREEN);
94     setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
95
96     mOpenCvCameraView = (CameraBridgeViewBase) findViewById(R.id.cameraView);
97     mOpenCvCameraView.setVisibility(SurfaceView.VISIBLE);
98     mOpenCvCameraView.setCvCameraViewListener(this);
99
100    LocationManager locationManager =
101        (LocationManager) getSystemService(LOCATION_SERVICE);
102    if (ActivityCompat.checkSelfPermission(
103        this, android.Manifest.permission.ACCESS_FINE_LOCATION)
104        != PackageManager.PERMISSION_GRANTED &&
105        ActivityCompat.checkSelfPermission(
106            this, android.Manifest.permission.ACCESS_COARSE_LOCATION) !=
107        PackageManager.PERMISSION_GRANTED) {
108        onDestroy();
109    }
110
111    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
112        this);
113
114    admBuild = new Build(-25.450027,
115        -49.232125, 20,
116        "Prédio de Administração",
117        "Jardim das Americas, Curitiba - PR, 80050-540",
118        "adm.ufpr.br",
119        "(41) 3360-4360");
120
121    bibBuild = new Build(-25.450819,
122        -49.232080, 20,
123        "Biblioteca Exatas UFPR",
124        "391, Rua Evaristo F. F. da Costa, 383 - Jardim das Américas,
125        Curitiba - State of Paraná",

```

```

126         "portal.ufpr.br", "");
127
128     pasBuild = new Build(-25.450278,
129         -49.231576, 20,
130         "Bloco de Aulas PA's", "", "", "");
131
132     bioBuild = new Build(-25.448105,
133         -49.233319, 20,
134         "Setor de Ciências Biológicas",
135         "Av. Cel. Francisco H. dos Santos, s/n - Jardim das Americas,
136         Curitiba - PR",
137         "bio.ufpr.br",
138         "(41) 3361-1799");
139     ruBuild = new Build(-25.449778,
140         -49.234964, 20,
141         "RU Centro Politécnico",
142         "Jardim das Americas, Curitiba - PR",
143         "pra.ufpr.br",
144         "(41) 3361-3248");
145
146     desBuild = new Build(-25.448604,
147         -49.234548, 20,
148         "Prédio Desconhecido", "", "", "");
149     labeBuild = new Build(-25.451984,
150         -49.233708, 20,
151         "Laboratório de Estudos Sedimentológicos e Petrologia Sedimentar",
152         "", "", "");
153     dinfBuild = new Build(-25.450894,
154         -49.232223, 20,
155         "Departamento de Informática - UFPR",
156     "Avenida Coronel Francisco Heráclito dos Santos, 100 - Jardim das Americas
157         - Jardim das Americas, Curitiba - PR, 81531-980",
158         "inf.ufpr.br",
159         "(41) 3361-3031");
160
161     farmBuild = new Build(-25.446967,
162         -49.239107, 20,
163         "Prédio de Farmácia", "", "", "");
164
165     socApliBuild = new Build(-25.444949,
166         -49.238805, 20,
167         "Setor de Ciências Sociais Aplicadas", "",
168         "sociais.ufpr.br",
169         "(41) 3360-4344");
170     engIndMad = new Build(-25.447881,
171         -49.238087, 20,
172         "Engenharia Industrial Madeireira",

```

```

173         "Avenida Prefeito Lothário Meissner, 632 -
174         Jardim Botânico, Curitiba - PR, 80210-170",
175         "madeira.ufpr.br",
176         "(41) 3360-4306");
177
178     predios = new Vector<>();
179     predios.add(admBuild);
180     predios.add(bibBuild);
181     predios.add(pasBuild);
182     predios.add(bioBuild);
183     predios.add(ruBuild);
184     predios.add(desBuild);
185     predios.add(labeBuild);
186     predios.add(dinfBuild);
187     predios.add(farmBuild);
188     predios.add(socApliBuild);
189     predios.add(engIndMad);
190
191     referenceImgAdmBuild = new Vector<>();
192     referenceImgBibBuild = new Vector<>();
193     referenceImgPasBuild = new Vector<>();
194     referenceImgBioBuild = new Vector<>();
195     referenceImgRuBuild = new Vector<>();
196     referenceImgDesBuild = new Vector<>();
197     referenceImgLabeBuild = new Vector<>();
198     referenceImgDinfBuild = new Vector<>();
199     referenceImgFarmBuild = new Vector<>();
200     referenceImgSocApliBuild = new Vector<>();
201     referenceImgEngIndMadBuild = new Vector<>();
202
203     referenceImgAdmBuild.add(R.drawable.admzero);
204     referenceImgAdmBuild.add(R.drawable.admum);
205
206     referenceImgBibBuild.add(R.drawable.bibzero);
207     referenceImgBibBuild.add(R.drawable.bibum);
208
209     referenceImgPasBuild.add(R.drawable.blocopaszero);
210
211     referenceImgPcsBuild.add(R.drawable.blocopcszero);
212     referenceImgPcsBuild.add(R.drawable.blocopcsum);
213
214     referenceImgCtsBuild.add(R.drawable.blococotszero);
215     referenceImgCtsBuild.add(R.drawable.blococotsum);
216
217     referenceImgBioBuild.add(R.drawable.biologicas);
218
219     referenceImgRuBuild.add(R.drawable.ru);

```



```

220
221 referenceImgDesBuild.add(R.drawable.desconhecido);
222
223 referenceImgLabeBuild.add(R.drawable.labesed);
224
225 referenceImgDinfBuild.add(R.drawable.dinf);
226
227 referenceImgFarmBuild.add(R.drawable.farmacia);
228
229 referenceImgSocApliBuild.add(R.drawable.socapli);
230
231 referenceImgEngIndMadBuild.add(R.drawable.engindmad);
232
233 referenceImgs = new Vector<>();
234 referenceImgs.add(referenceImgAdmBuild);
235 referenceImgs.add(referenceImgBibBuild);
236 referenceImgs.add(referenceImgPasBuild);
237 referenceImgs.add(referenceImgBioBuild);
238 referenceImgs.add(referenceImgRuBuild);
239 referenceImgs.add(referenceImgDesBuild);
240 referenceImgs.add(referenceImgLabeBuild);
241 referenceImgs.add(referenceImgDinfBuild);
242 referenceImgs.add(referenceImgFarmBuild);
243 referenceImgs.add(referenceImgSocApliBuild);
244 referenceImgs.add(referenceImgEngIndMadBuild);
245
246 InitFragment mInitFragment = new InitFragment();
247 FragmentTransaction transaction = getSupportFragmentManager()
248     .beginTransaction();
249 transaction.replace(R.id.fragment_container, mInitFragment);
250 transaction.addToBackStack(null);
251 transaction.commit();
252 }
253
254 @Override
255 public void onPause() {
256     super.onPause();
257     if (mOpenCvCameraView != null) {
258         mOpenCvCameraView.disableView();
259     }
260 }
261
262 public void onResume() {
263     super.onResume();
264     if (!OpenCVLoader.initDebug()) {
265         OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_1_0, this,
266             mLoaderCallback);

```

```

267     } else {
268         mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS);
269     }
270 }
271
272 public void onDestroy() {
273     super.onDestroy();
274     if (mOpenCvCameraView != null) {
275         mOpenCvCameraView.disableView();
276     }
277 }
278
279 @Override
280 public boolean onTouch(View view, MotionEvent motionEvent) {
281     CameraFragment mCameraFragment = new CameraFragment();
282     FragmentTransaction transaction =
283         getSupportFragmentManager().beginTransaction();
284     transaction.replace(R.id.fragment_container, mCameraFragment);
285     transaction.addToBackStack(null);
286     transaction.commit();
287     exibindoInformacao = false;
288     return false;
289 }
290
291 @Override
292 public void onCameraViewStarted(int width, int height) {
293     cameraImg = new Mat();
294 }
295
296 @Override
297 public void onCameraViewStopped() {
298     cameraImg.release();
299 }
300
301 public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
302     cameraImg = inputFrame.rgba();
303
304     if (!exibindoInformacao) {
305         Log.e("INFORMAÇÃO", "Não exibindo informação");
306         try {
307             cameraImageDescriptor = new Build(myLatitude, myLongitude, 0, "", "", "", "");
308             cameraImageDescriptor.initImages(null, MainActivity.this, cameraImg);
309         } catch (IOException e) {
310             Log.e("ERRO", "Erro ao iniciar imagem da camera");
311             e.printStackTrace();
312         }
313     }

```

```

314 for (int k = 0; k < predios.size(); k++){
315     if (distFrom(myLatitude, myLongitude, predios.get(k).latitude,
316               predios.get(k).longitude) < predios.get(k).dist){
317         Log.e("LOCALIZAÇÃO", "Entrei na localização");
318         if (!predios.get(k).init){
319             try {
320                 predios.get(k).initImages(referenceImgs.get(k), MainActivity.this, null);
321             } catch (IOException e) {
322                 Log.e("ERROR", "Erro ao iniciar prédio de id: " + Integer.toString(k));
323                 e.printStackTrace();
324             }
325         }
326
327         test = false;
328         for(int i = 0; i < referenceImgs.get(k).size(); i++){
329             cameraImageDescriptor.imageDescriptors.get(0).tryMatcher(
330                 predios.get(k).imageDescriptors.get(i));
331
332             test = test ||
333                 cameraImageDescriptor.imageDescriptors.get(0).exibeInformacao;
334         }
335         if (test) {
336             ShowInformationsFragment showFragment = new ShowInformationsFragment();
337             showFragment.setString(predios.get(k).local,
338                                   predios.get(k).end, predios.get(k).site, predios.get(k).fone);
339             FragmentTransaction transaction = getSupportFragmentManager().
340                 beginTransaction();
341             transaction.replace(R.id.fragment_container, showFragment);
342             transaction.addToBackStack(null);
343             transaction.commit();
344             exibindoInformacao = true;
345             return cameraImg;
346         }
347     }
348 }
349 return cameraImg;
350 }
351
352 @Override
353 public void onLocationChanged(Location location) {
354     myLatitude = location.getLatitude();
355     myLongitude = location.getLongitude();
356 }
357
358 @Override
359 public void onStatusChanged(String provider, int status, Bundle extras) {
360 }

```

```

361
362 @Override
363 public void onProviderEnabled(String provider) {
364 }
365
366 @Override
367 public void onProviderDisabled(String provider) {
368 }
369
370 public static double distFrom(double lat1, double lng1,
371                               double lat2, double lng2) {
372     double earthRadius = 6371000; //meters
373     double dLat = Math.toRadians(lat2-lat1);
374     double dLng = Math.toRadians(lng2-lng1);
375     double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
376               Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
377               Math.sin(dLng/2) * Math.sin(dLng/2);
378     double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
379
380     return (float) (earthRadius * c);
381 }
382 }

```

A.0.2 ImageDescriptor.java

```

1 package com.example.krinski.ufprbuildingdetector;
2
3 import android.content.Context;
4 import org.opencv.android.Utils;
5 import org.opencv.calib3d.Calib3d;
6 import org.opencv.core.Core;
7 import org.opencv.core.CvType;
8 import org.opencv.core.DMatch;
9 import org.opencv.core.KeyPoint;
10 import org.opencv.core.Mat;
11 import org.opencv.core.MatOfDMatch;
12 import org.opencv.core.MatOfKeyPoint;
13 import org.opencv.core.MatOfPoint;
14 import org.opencv.core.MatOfPoint2f;
15 import org.opencv.core.Point;
16 import org.opencv.features2d.DescriptorExtractor;
17 import org.opencv.features2d.DescriptorMatcher;
18 import org.opencv.features2d.FeatureDetector;
19 import org.opencv.imgcodecs.Imgcodecs;
20 import org.opencv.imgproc.Imgproc;
21

```

```

22 import java.io.IOException;
23 import java.util.ArrayList;
24 import java.util.List;
25
26 class ImageDescriptor {
27     private Mat imgGray;
28     private Mat imgCorners;
29     private Mat imgDescriptor;
30     private MatOfKeyPoint imgKeypoints;
31     private MatOfDMatch matches;
32     boolean exibeInformacao;
33
34     ImageDescriptor(int referenceImageID, Context context, Mat image)
35                                     throws IOException {
36         this.imgGray = new Mat();
37         this.imgDescriptor = new Mat();
38         this.imgKeypoints = new MatOfKeyPoint();
39
40         FeatureDetector featureDetector = FeatureDetector.create(
41                                     FeatureDetector.ORB);
42         DescriptorExtractor descriptorExtractor = DescriptorExtractor.create(
43                                     DescriptorExtractor.ORB);
44
45         Mat img;
46         if (image != null) {
47             img = image;
48         } else {
49             img = Utils.loadResource(context, referenceImageID,
50                                     Imgcodecs.CV_LOAD_IMAGE_COLOR);
51         }
52
53         Imgproc.cvtColor(img, this.imgGray, Imgproc.COLOR_BGR2GRAY);
54
55         featureDetector.detect(this.imgGray, this.imgKeypoints);
56         descriptorExtractor.compute(this.imgGray, this.imgKeypoints,
57                                     this.imgDescriptor);
58
59         if (image == null) {
60             this.imageCorners(null);
61         }
62     }
63
64     void tryMatcher(ImageDescriptor referenceImg) {
65         this.matches = new MatOfDMatch();
66         DescriptorMatcher descriptorMatcher = DescriptorMatcher.create(
67                                     DescriptorMatcher.BRUTEFORCE_HAMMING);
68         descriptorMatcher.match(this.imgDescriptor, referenceImg.imgDescriptor,

```

```

69                                     this.matches);
70     this.imageCorners(referenceImg);
71 }
72
73 private void imageCorners(ImageDescriptor referenceImg) {
74
75     if (referenceImg == null){
76         this.imgCorners = new Mat(4, 1, CvType.CV_32FC2);
77         this.imgCorners.put(0, 0, 0.0, 0.0);
78         this.imgCorners.put(1, 0, this.imgGray.cols(), 0.0);
79         this.imgCorners.put(2, 0, this.imgGray.cols(), this.imgGray.rows());
80         this.imgCorners.put(3, 0, 0.0, this.imgGray.rows());
81         return;
82     }
83     this.exibeInformacao = false;
84
85     MatOfPoint intSceneCorners = new MatOfPoint();
86     this.imgCorners = new Mat(0, 0, CvType.CV_32FC2);
87     Mat candidateSceneCorners = new Mat(4, 1, CvType.CV_32FC2);
88
89     List<DMatch> matchesList = this.matches.toList();
90
91     if (matchesList.size() < 4) {
92         return;
93     }
94
95     List<KeyPoint> referenceKeypointsList = referenceImg.imgKeypoints.toList();
96     List<KeyPoint> sceneKeypointsList = this.imgKeypoints.toList();
97
98     double minDist = Double.MAX_VALUE;
99     for (DMatch match : matchesList) {
100         double dist = match.distance;
101         if (dist < minDist) {
102             minDist = dist;
103         }
104     }
105
106     if (minDist > 25.0){
107         return;
108     }
109
110
111     ArrayList<Point> goodReferencePointsList = new ArrayList<>();
112     ArrayList<Point> goodScenePointsList = new ArrayList<>();
113     double maxGoodMatchDist = 1.75 * minDist;
114     for (DMatch match : matchesList) {
115         if (match.distance < maxGoodMatchDist) {

```

```

116     goodReferencePointsList.add(referenceKeypointsList.
117                               get(match.trainIdx).pt);
118     goodScenePointsList.add(sceneKeypointsList.get(match.queryIdx).pt);
119 }
120 }
121
122 if (goodReferencePointsList.size() < 4 || goodScenePointsList.size() < 4) {
123     return;
124 }
125
126 MatOfPoint2f goodReferencePoints = new MatOfPoint2f();
127 goodReferencePoints.fromList(goodReferencePointsList);
128
129 MatOfPoint2f goodScenePoints = new MatOfPoint2f();
130 goodScenePoints.fromList(goodScenePointsList);
131
132 Mat homography = Calib3d.findHomography(goodReferencePoints,
133                                         goodScenePoints);
134
135 if (homography.cols() == 3) {
136     Core.perspectiveTransform(referenceImg.imgCorners, candidateSceneCorners,
137                               homography);
138     candidateSceneCorners.convertTo(intSceneCorners, CvType.CV_32SC2);
139
140     if (Imgproc.isContourConvex(intSceneCorners)) {
141         candidateSceneCorners.copyTo(this.imgCorners);
142         this.exibeInformacao = true;
143     }
144 }
145 }
146 }

```

A.0.3 Build.java

```

1 package com.example.krinski.ufprbuildingdetector;
2
3 import android.content.Context;
4
5 import org.opencv.core.Mat;
6
7 import java.io.IOException;
8 import java.util.Vector;
9
10 class Build {
11
12     boolean init;

```

```

13  int dist;
14  double latitude;
15  double longitude;
16  Vector<ImageDescriptor> imageDescriptors;
17  String local;
18  String end;
19  String site;
20  String fone;
21
22  Build(double lat,double lng,int d,String l,String e,String s,String f){
23      this.init = false;
24      this.latitude = lat;
25      this.longitude = lng;
26      this.dist = d;
27      this.local = l;
28      this.end = e;
29      this.site = s;
30      this.fone = f;
31  }
32
33  void initImages(Vector<Integer> referencesImagesIds, Context context,
34                                                         Mat image)
35                                                         throws IOException{
36      this.init = true;
37      imageDescriptors = new Vector<>();
38      if (image != null){
39          ImageDescriptor img = new ImageDescriptor(0,context,image);
40          imageDescriptors.add(img);
41      } else {
42          for (int i = 0; i < referencesImagesIds.size(); i++) {
43              ImageDescriptor img = new ImageDescriptor(referencesImagesIds.get(i),
44                                                         context,null);
45              imageDescriptors.add(img);
46          }
47      }
48  }
49  }

```

A.0.4 ShowInformationsFragment.java

```

1  package com.example.krinski.ufprbuildingdetector;
2
3  import android.content.Context;
4  import android.os.Bundle;
5  import android.support.v4.app.Fragment;
6  import android.view.LayoutInflater;

```



```
7 import android.view.View;
8 import android.view.ViewGroup;
9 import android.widget.TextView;
10
11 public class ShowInformationsFragment extends Fragment {
12
13     String s1,s2,s3,s4;
14
15     public ShowInformationsFragment() {
16     }
17
18     public void setString(String t1,String t2,String t3,String t4) {
19         s1 = t1;
20         s2 = t2;
21         s3 = t3;
22         s4 = t4;
23     }
24
25     @Override
26     public void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28     }
29
30     @Override
31     public View onCreateView(LayoutInflater inflater, ViewGroup container,
32                             Bundle savedInstanceState) {
33         ViewGroup rootView = (ViewGroup) inflater.inflate(
34             R.layout.fragment_show_informations, container, false);
35
36         TextView local = (TextView) rootView.findViewById(R.id.local);
37         local.setText("Localização:" + this.s1);
38
39         TextView end = (TextView) rootView.findViewById(R.id.end);
40         end.setText("Endereço:" + this.s2);
41
42         TextView site = (TextView) rootView.findViewById(R.id.site);
43         site.setText("Website:" + this.s3);
44
45         TextView fone = (TextView) rootView.findViewById(R.id.fone);
46         fone.setText("Telefone:" + this.s4);
47
48         return rootView;
49     }
50
51     @Override
52     public void onAttach(Context context) {
53         super.onAttach(context);
```

```

54     }
55
56     @Override
57     public void onDetach() {
58         super.onDetach();
59     }
60 }

```

A.0.5 CameraFragment.java

```

1 package com.example.krinski.ufprbuildingdetector;
2
3 import android.content.Context;
4 import android.os.Bundle;
5 import android.support.v4.app.Fragment;
6 import android.view.LayoutInflater;
7 import android.view.View;
8 import android.view.ViewGroup;
9
10 public class CameraFragment extends Fragment {
11
12     public CameraFragment() {
13     }
14
15     @Override
16     public void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18     }
19
20     @Override
21     public View onCreateView(LayoutInflater inflater, ViewGroup container,
22                             Bundle savedInstanceState) {
23         return inflater.inflate(R.layout.fragment_camera, container, false);
24     }
25
26     @Override
27     public void onAttach(Context context) {
28         super.onAttach(context);
29     }
30
31     @Override
32     public void onDetach() {
33         super.onDetach();
34     }
35 }

```

A.0.6 InitFragment.java

```

1 package com.example.krinski.ufprbuildingdetector;
2
3 import android.content.Context;
4 import android.os.Bundle;
5 import android.support.v4.app.Fragment;
6 import android.view.LayoutInflater;
7 import android.view.View;
8 import android.view.ViewGroup;
9
10 public class InitFragment extends Fragment {
11
12     public InitFragment() {
13     }
14
15     @Override
16     public void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18     }
19
20     @Override
21     public View onCreateView(LayoutInflater inflater,
22                             ViewGroup container, Bundle savedInstanceState) {
23         return inflater.inflate(R.layout.fragment_init, container, false);
24     }
25
26     @Override
27     public void onAttach(Context context) {
28         super.onAttach(context);
29     }
30
31     @Override
32     public void onDetach() {
33         super.onDetach();
34     }
35 }

```

A.0.7 activity_main.xml

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <FrameLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:id="@+id/fragment_container"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"

```

```

7  android:orientation="horizontal">
8
9  <fragment
10     android:id="@+id/camera_fragment"
11     android:layout_width="wrap_content"
12     android:layout_height="wrap_content"
13     android:name="com.example.krinski.ufprbuildingdetector.CameraFragment">
14
15     <org.opencv.android.JavaCameraView
16         android:layout_width="match_parent"
17         android:layout_height="match_parent"
18         android:id="@+id/cameraView"/>
19     </fragment>
20 </FrameLayout>

```

A.0.8 fragment_camera.xml

```

1  <FrameLayout
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5  </FrameLayout>

```

A.0.9 fragment_init.xml

```

1  <FrameLayout
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:background="@color/colorPrimary">
6
7     <TextView
8         android:text="@string/start"
9         android:layout_width="fill_parent"
10        android:layout_height="fill_parent"
11        android:textSize="200sp"
12        android:textColor="#ffffff"
13        android:textAlignment="center"
14        android:id="@+id/textView" />
15
16 </FrameLayout>

```

A.0.10 fragment_show_informations.xml

```
1 <FrameLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:tools="http://schemas.android.com/tools"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   android:background="@color/white"
7   tools:context="com.example.krinski.ufprbuildingdetector.
8                               ShowInformationsFragment">
9
10  <TextView
11    android:layout_width="match_parent"
12    android:layout_height="359dp"
13    android:id="@+id/local" />
14
15  <TextView
16    android:layout_marginTop="20dp"
17    android:layout_width="match_parent"
18    android:layout_height="match_parent"
19    android:id="@+id/end" />
20
21  <TextView
22    android:layout_marginTop="60dp"
23    android:layout_width="match_parent"
24    android:layout_height="match_parent"
25    android:id="@+id/site" />
26
27  <TextView
28    android:layout_marginTop="80dp"
29    android:layout_width="match_parent"
30    android:layout_height="match_parent"
31    android:id="@+id/fone" />
32 </FrameLayout>
```