



**Universidade do Estado do Rio de Janeiro**

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Luiz Otavio Soares de Oliveira

**Medição automática de pontos de costura para conjuntos de imagens e  
sua aplicação ao Projeto E-Foto**

Rio de Janeiro

2022

Luiz Otavio Soares de Oliveira

**Medição automática de pontos de costura para conjuntos de imagens e sua aplicação ao  
Projeto E-Foto**



Projeto Final apresentado, como requisito parcial para obtenção do grau de Engenheiro Cartógrafo, ao Departamento de Engenharia Cartográfica da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Me. Irving da Silva Badolato

Rio de Janeiro  
2022

Ficha elaborada pelo autor através do  
Sistema para Geração Automática de Ficha Catalográfica da Rede Sirius - UERJ

O48m Oliveira, Luiz Otavio Soares de .  
Medição automática de pontos de costura para ...  
imagens e sua aplicação ao Projeto E-Foto / Luiz  
Otavio Soares de Oliveira. - 2021.  
96 f.

Orientador: Irving da Silva Badolato.  
Projeto Final apresentado à Universidade do Estado  
do Rio de Janeiro, Faculdade de Engenharia, para  
obtenção do grau de bacharel em Engenharia  
Cartográfica.

1. Fotogrametria - Monografias. 2. Costura de  
imagens - Monografias. 3. Projeto E-Foto -  
Monografias. I. Badolato, Irving da Silva. II.  
Universidade do Estado do Rio de Janeiro. Faculdade  
de Engenharia. III. Título.

CDU 62+528

Luiz Otavio Soares de Oliveira

**Medição automática de pontos de costura para conjuntos de imagens e sua aplicação ao  
Projeto E-Foto**

Projeto Final apresentado, como requisito parcial para obtenção do grau de Engenheiro Cartógrafo, ao Departamento de Engenharia Cartográfica da Universidade do Estado do Rio de Janeiro.

Aprovada em 13 de maio de 2022.

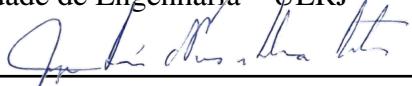
Banca Examinadora:

---



Prof. Me. Irving da Silva Badolato (Orientador)  
Faculdade de Engenharia – UERJ

---



Prof. Jorge Luis Nunes e Silva Brito, Ph.D.  
Faculdade de Engenharia - UERJ

---

JOAO ARAUJO RIBEIRO:88819329700

Assinado de forma digital por JOAO ARAUJO  
RIBEIRO:88819329700  
Dados: 2022.05.16 20:38:13 -03'00'

---

Prof. Dr. João Araújo Ribeiro  
Faculdade de Engenharia - UERJ

Rio de Janeiro  
2022

## **DEDICATÓRIA**

Aos meus queridos amigos e companheiros de jornada. Ela não foi longa, foi necessária para preparar o caminho. Que venham os próximos desafios! Sei que sempre poderei contar com todos vocês! Muito obrigado pelo tempo, fé e paciência em mim depositados. Amo vocês!

## **AGRADECIMENTOS**

Agradeço primeiramente a UERJ que apesar do tempo nunca duvidou do meu potencial, possibilitou minha formação e me ensinou a ser uma pessoa melhor do que quando ingressei. Ao meu incansável orientador que mesmo ocupado não me abandonou a deriva neste mar de direções a serem seguidas e a sua família por entender e aceitar nossos horários loucos e longos de reunião. Aos professores do departamento de Cartografia por seus ensinamentos.

A minha família que me apoiou nessa trajetória de pontos, costuras e ligações até formar uma imagem única e clara de quem sou e para onde vou. Em especial minha mãe Maria da Conceição, a pessoa mais forte que eu conheço que nunca se deixou levar pelas adversidades e me ensinou a ser a pessoa que sou hoje, minha irmã Maria Caroliny, que sempre me apoiou e me desafiou a ser melhor que eu fui ontem, meu pai Luiz Antônio, que me apoiou mesmo que as vezes eu não entendesse seu apoio, e meu namorado Renan, cujo apoio, amor incondicional e ombro amigo me salvaram diversas vezes nesses anos juntos.

Aos amigos que a vida trouxe e levou de acordo com seus caprichos, mas principalmente aos que se mantém que são amizades que pretendo cultivar para toda a vida. Especialmente, meus amigos mais antigos Thomaz, Ronald e Márcio. A Daniele que me apoiou em diversas descobertas da vida, e Renan Monteiro que apesar de sua jovem estadia em minha vida já é presença garantida no resto dela.

Aos profissionais da clínica Sinte em especial Jaqueline e Lilian, que cuidaram da minha coluna com seus diversos maquinários de tortura, independente de minhas reclamações, e assim possibilitaram as longas horas de trabalho que este trabalho demandou.

E finalmente agradeço a todos os meus cães, que me fizeram companhia nas longas horas dessa jornada.

A todos o meu mais profundo agradecimento.

”A tarefa não é tanto ver aquilo que ninguém viu, mas pensar o que ninguém ainda pensou sobre  
aquilo que todo mundo vê.”

*Arthur Schopenhauer*

## RESUMO

OLIVEIRA, LO *Medição automática de pontos de costura para conjuntos de imagens e sua aplicação ao Projeto E-Foto.* 2022. 96 f. Projeto Final (Graduação em Engenharia Cartográfica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2022.

A fotogrametria é uma das principais áreas de estudo em cursos de engenharia cartográfica. Ela é capaz de realizar a reconstrução métrica de espaços tridimensionais com base num par, ou mais, de fotografias de um mesmo objeto. Para alcançar esse objetivo tipicamente é necessário um conjunto de observações de pontos comuns nessas imagens. Este conjunto de medidas viabiliza a ligação ou costura de imagens, mas pode ser um processo extremamente custoso, pois aumenta em complexidade com número de imagens. Esta ligação foi por muitos anos feita manualmente, contudo tende a cair em desuso graças a rápida evolução tecnológica, que possibilita o uso de imagens digitais e processos de automação. Tendo em vista a aplicação para ensino de fotogrametria digital desenvolvida como software livre na Universidade do Estado do Rio de Janeiro no Projeto E-Foto, o e-foto, este trabalho foca-se no estudo de ferramentas de visão computacional, outra área do saber intimamente ligada à fotogrametria, que permita a automação de medidas ou pontos de costura para conjuntos de imagens. Para mais, a visão computacional é uma das partes da área de inteligência artificial e assim preocupa-se com a interpretação, além da geometria, ao lidar com imagens. Logo, é vital estudar bibliotecas para desenvolvimento de aplicações que implementem trabalhos desta área, como o OpenCV, e comparar as ferramentas disponíveis que possam ser úteis ao e-foto. Deste modo, foram definidos critérios de análise distintos, tais como tempo de execução, consumo de memória, qualidade do ajustamento, quantidade de respostas, sua distribuição e capacidade de restringir ou filtrar tais resultados. Foi codificada uma aplicação de linha de comando, de modo que possa ser integrada ao e-foto, que pode fazer uso de diversos algoritmos, a escolha do usuário, para a detecção e descrição de pontos chave para feições em imagens. Estas feições descritas podem ser caracterizadas por grande invariância de termos de rotação, escala e iluminação, de modo que pode ser feita a correlação de pares de imagens mesmo quando não há estabilidade da plataforma fotográfica usada. Contudo, faz-se importante a conferência das correlações adotando uma solução geométrica, como a da transformação homográfica, para robustecer os resultados. Para demonstrar nossa aplicação foram usadas imagens de exemplo do Projeto E-Foto, além de conjuntos maiores em casos específicos, onde o conjunto não poderia atender as necessidades dos testes planejados. Foram tabeladas as respostas comparando os algoritmos SIFT, SURF, ORB e AKAZE, tendo todos estes demonstrado capacidade de produzir um número de resultados satisfatórios. Foram usadas diferentes estratégias de correlação para entendimento dos possíveis impactos ao fluxo da aplicação quando variada a volumetria de feições. Destacam-se entre os algoritmos estudados para extrair feições o ORB e SIFT, tendo estes se destacado em tempo e qualidade, respectivamente. Para estratégia de correlação fica recomendado uso de correlação cruzada apenas em pequenos conjuntos de feições. Os resultados deste trabalho estão disponíveis em <https://github.com/tatorj/Tcc>.

Palavras-chave: Fotogrametria. Costura de imagens. Projeto E-Foto. OpenCV.

## ABSTRACT

OLIVEIRA, LO *Automatic measurement of stitch points for image sets and their application to the E-Foto Project..* 2022. 96 f. Projeto Final (Graduação em Engenharia Cartográfica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2022.

Photogrammetry is one of the main areas of study in cartographic engineering courses. It's capable of performing the metric reconstruction of three-dimensional spaces based on a pair, or more, of photographs of the same object. To achieve this objective, a set of observations of common points in these images is typically required. This set of measurements makes it possible to link or stitch images, but it can be an extremely expensive process, as it increases in complexity with the number of images. This connection was made manually for many years, however, it tends to fall into disuse thanks to rapid technological developments, which allow the use of digital images and automation processes. Considering the application for teaching digital photogrammetry developed as free software at the State University of Rio de Janeiro in the E-Foto Project, e-foto, this work focuses on the study of computer vision tools, another area of knowledge closely linked to photogrammetry, which allows for the automation of measurements or stitch points for sets of images. Furthermore, computer vision is only one part of the field of artificial intelligence and thus it is concerned with interpretation, in addition to geometry, when dealing with images. Therefore, it is vital to study libraries for the development of applications that implement works in this area, such as OpenCV, and to compare the available tools that may be useful to e-foto. In this way, different analysis criteria were defined, such as execution time, memory consumption, quality of adjustment, quantity of responses, their distribution and the ability to restrict or filter such results. A command line application was coded so that it can be integrated into e-foto, which can make use of different algorithms, chosen by the user, for detection and description of key points for features in images. The features described can be characterized by high invariance in terms of rotation, scale and illumination, so that image pairs can be correlated even when the photographic platform used is not stable. However, it is important to check the correlations by adopting a geometric solution, such as the homographic transformation, in order to strengthen the results. In order to demonstrate our application, example images from the E-Foto Project were used, as well as larger sets in specific cases, where the set could not meet the needs of the planned tests. The answers were tabulated comparing the SIFT, SURF, ORB and AKAZE algorithms, all of which demonstrated the ability to produce a number of satisfactory results. Different correlation strategies were used to understand the possible impacts on the application flow when the feature volume was varied. Among the algorithms studied to extract features, the ORB and SIFT stand out, and they do so in terms of time and quality, respectively. For correlation strategy, it is recommended to use cross-correlation only in small sets of features. The results of this work are available at [⟨https://github.com/tatorj/Tcc⟩](https://github.com/tatorj/Tcc).

Keywords: Photogrammetry. Image stitch. E-Foto Project. OpenCV.

## LISTA DE FIGURAS

Figura 1 - Esquema de captação de imagens em aerolevantamentos. . . . .	15
Figura 2 - Reconstrução das coordenadas 3D a partir de imagens 2D. . . . .	16
Figura 3 - Diagrama de atividades do e-foto . . . . .	19
Figura 4 - Exemplo de representação para pontos chave . . . . .	22
Figura 5 - Exemplo de correlação . . . . .	24
Figura 6 - Sequência de imagens adotadas em UERJ_IO.epp . . . . .	32
Figura 7 - Regiões de interesse para distribuição de pontos de gruber . . . . .	34
Figura 8 - Fases da execução da solução de linha de comando proposta . . . . .	35
Figura 9 - Diagrama de classe da solução . . . . .	36
Figura 10 - Diagrama de atividades para o processamento de imagens . . . . .	39
Figura 11 - Diagrama de atividades para o processamento de pares . . . . .	40
Figura 12 - Diagrama de atividades para o processamento de medidas . . . . .	42
Figura 13 - Exemplo do arquivo de persistência de pontos. . . . .	43
Figura 14 - Exemplo do arquivo de persistência de medidas. . . . .	43
Figura 15 - Volume e distribuição dos resultados obtidos . . . . .	51
Figura 16 - Seleção de recortes das imagens para verificação da qualidade . . . . .	52
Figura 17 - Imagem ilustrativa da resposta ao utilizar máscara. . . . .	53
Figura 18 - Gráfico da diferença entre métodos de correlação BFM e FLANN. . . . .	55
Figura 19 - Padrões interpolados para imagens de 360x360 pixels . . . . .	93

## **LISTA DE TABELAS**

Tabela 1 - Módulos da OpenCV 4.5.5 . . . . .	25
Tabela 2 - Implementações da interface Feature2D . . . . .	30
Tabela 3 - Implementações da interface DescriptorMatcher . . . . .	31
Tabela 4 - Resultados numéricos de detecção e descrição dos métodos analisados. . . . .	46
Tabela 5 - Resultados numéricos da correlação e verificação geométrica. . . . .	48
Tabela 6 - Comparação entre o número de <i>inliers</i> nos pares e o total de pontos de costura	49
Tabela 7 - Resposta aos diferentes valores de resíduo máximo aplicáveis . . . . .	56

## LISTA DE ABREVIATURAS E SIGLAS

API:	Application Programming Interface
3D:	Espaço tridimensional
2D:	Espaço bidimensional
e-foto:	Estação fotogramétrica digital educacional
UERJ:	Universidade do Estado do Rio de Janeiro
LFSR:	Laboratório de Fotogrametria e Sensoriamento Remoto
DSM:	Digital Surface Model
DEM:	Digital Elevation Model
GIS:	Geographic Information Sistem
IBGE:	Instituto Brasileiro de Geografia e Estatística
IBM:	International Business Machines
IA:	Inteligencia Artificial
BFM:	Brute-Force Matcher
SURF:	Speeded Up Robust Features
LSM:	Least Squares Method
RANSAC:	Random Sample Consensus
LMEDS:	Least-Median of Squares
RHO:	Progressive Sample Consensus
OpenCV:	Open Computer Vision
SIFT:	Scale Invariant Feature Transform
AKAZE:	Accelerated Kaze
FED:	Fast Explicit Diffusion
ORB:	Oriented FAST and Rotated BRIEF
FAST:	Features from Accelerated Segment Test
BRIEF:	Binary Robust Independent Elementary Features
ODM:	Open Drone Maps
OSGeo:	Open Source Geospatial Foundation
CUDA	Computer Unified Device Architecture
GPU:	Graphics Processing Unit
FLANN:	Fast Library for Approximate Nearest Neighbors
UML:	Unified Modeling language
KD-Tree:	Árvore de K-dimensões
STL:	Standard Template Library
ISPRS:	International Society for Photogrammetry and Remoto Sensing

## SUMÁRIO

<b>INTRODUÇÃO</b>	12
<b>1 FUNDAMENTAÇÃO TEÓRICA</b>	14
<b>1.1 Fotogrametria</b>	14
<b>A Estação Fotogramétrica Digital E-Foto</b>	17
<b>1.2 Visão Computacional</b>	20
<b>A biblioteca OpenCV</b>	24
<b>2 METODOLOGIA</b>	27
<b>2.1 Estudo do cenário</b>	27
<b>Levantamento das necessidades do e-foto</b>	27
<b>Seleção de ferramentas aplicáveis na solução</b>	28
<b>Conjuntos de dados para testes</b>	32
<b>2.2 Definição dos critérios de análise das opções disponíveis</b>	33
<b>Tempo de execução e memória alocada</b>	33
<b>Distribuição e volume das respostas</b>	33
<b>Verificação da qualidade</b>	34
<b>Viabilidade de filtragem de melhores pontos</b>	34
<b>2.3 Planejamento de testes comparativos</b>	35
<b>2.4 Codificação da solução</b>	35
<b>Escolha de argumentos</b>	37
<b>Processamento de imagens</b>	39
<b>Processamento de pares</b>	39
<b>Processamento de medidas</b>	41
<b>Persistência de dados</b>	43
<b>3 RESULTADOS</b>	45
<b>3.1 Resultados numéricos dos testes</b>	45
<b>Custo de detecção e descrição</b>	45
<b>Custo de correlação e Verificação geométrica</b>	47
<b>Taxa de obtenção dos mesmos pontos em diferentes pares</b>	49
<b>3.2 Análise visual dos resultados pela carga no e-foto</b>	50
<b>3.3 Impacto do uso de diferentes parâmetros</b>	54
<b>Variação dos métodos de correlação</b>	54
<b>Variação dos resíduos</b>	56
<b>CONCLUSÃO</b>	57
<b>REFERÊNCIAS</b>	59
<b>APÊNDICE A – Código fonte</b>	62
<b>APÊNDICE B – Padrões para definir regiões de interesse de gruber</b>	90



## INTRODUÇÃO

A história da cartografia e do mapeamento se entrelaça com a da humanidade, visto que este meio de informação é essencial para nos situarmos geograficamente. No entanto, uma visão real, a nadir, para geração de mapas melhores foi por muito tempo um feito quase impossível. Mesmo com o advento dos voos controlados usando balões de ar quente, ainda era complexo registrar a informação a ser trabalhada. Então com a invenção e popularização da câmera fotográfica, na primeira metade do século XIX, o inventor francês Aimé Laussedat, percebeu as aplicações que essa nova tecnologia poderia ter no campo da cartografia. Uma das maiores inovações viria no campo do mapeamento, assim criando uma nova área na cartografia que viria a ser conhecida como fotogrametria.

Com o início da utilização de fotografias para mapeamento, surgiu a questão sobre como conectar várias imagens, umas às outras, para tornar possível a aquisição de mais informações sobre o objeto fotografado. Dentre as práticas adotadas tornou-se comum a escolha de pontos homólogos, que são partes dos mesmos objetos na cena fotografada visíveis em fotos diferentes, para prender estas imagens umas às outras usando-os como pontos de costura. Por sua vez, costurar conjuntos de imagens manualmente é um processo extremamente longo que aumenta em complexidade rapidamente conforme cresce o número de imagens disponíveis.

Tal costura era inicialmente realizada fisicamente ao se revelar as imagens e criar o mosaico literalmente costurando fisicamente as fotos para que o modelo se tornasse íntegro. Com a necessidade de medidas mais rigorosas e a evolução tecnológica foram criadas máquinas de restituição óptico-mecânicas, nas quais as imagens, que anteriormente eram costuradas, passaram a serem visualizadas por meio de diafilmes e com elas o restituidor passou a poder ajustá-las para obter visão estereoscópica e, assim, realizar as medidas. Com suportes mecânicos era possível inclusive plotar pontos e vetorizar outras geometrias sem a necessidade de costurar fisicamente as imagens para realização desses procedimentos.

Recentemente, com a evolução da computação e das câmeras, passou-se a utilizar imagens digitais em ambientes computadorizados. Por sua vez, a costura de imagens digitais já é um processo bem documentado e automatizado na área de visão computacional. Esse campo compartilha métodos computacionais de aprendizado de máquina, do campo de inteligência artificial, para realizar entre outras coisas a análise de entradas visuais com interpretação de imagens e a reconstrução da geometria das cenas.

Para fins acadêmicos, no contexto de desenvolvimento de ferramentas para ensino de fotogrametria, o Projeto E-foto possui entre seus produtos um software, denominado e-foto, que visa ser uma plataforma de fotogrametria digital em software livre, ele é desenvolvido no Laboratório de Fotogrametria e Sensoriamento Remoto da Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro, desde 2004, tendo sido implementado por professores e alunos desde seu início. Em sua trajetória observa-se que esse tipo de implementação gera

lacunas, onde certas automações não são incluídas, por não serem imediatamente necessárias para o funcionamento do projeto ou para que os passos intermediários possam ser estudados e praticados formando pensamento crítico sobre os resultados obtidos para análise de resultados finais. Logo, trabalhos repetitivos como o descrito acima ainda são feitos manualmente, não existindo alternativa viável para acelerar a produção depois de compreender o assunto, e, como consequência, aumentando a carga horária da realização de trabalhos.

Este trabalho se propõe a realizar um comparativo entre diversas técnicas comuns no processo de obtenção automática de pontos de costura, tais como detecção e descrição de pontos chave, correlação e verificação geométrica de pontos correlatos, utilizando-se das atuais tecnologias no campo de visão computacional. Estima-se que tal ensaio deve servir de balizador para determinar quais destas técnicas melhor se adéquam ao Projeto E-foto do ponto de vista da qualidade, desempenho e escalabilidade usando imagens aéreas. Entre os resultados materializáveis está uma proposta de solução para o problema que possa ser integrada ao software em versões futuras.

Além desta introdução e da conclusão, o desenvolvimento deste trabalho apresenta divisão em 3 capítulos, a saber: A fundamentação teórica, que apresenta as áreas de fotogrametria e visão computacional; a metodologia, que apresenta os passos necessários à confecção da solução para o problema proposto; e os resultados, onde diversos dados comparativos são analisados.

## 1 FUNDAMENTAÇÃO TEÓRICA

Este trabalho se propõe à análise da adoção de tecnologias da visão computacional em processos de fotogrametria, em especial, no software e-foto<sup>1</sup>. Por este motivo, as seções deste capítulo abordarão principalmente os conceitos de Fotogrametria e Visão Computacional.

### 1.1 Fotogrametria

A fotogrametria é definida pela Sociedade Internacional para Fotogrametria e Sensoriamento Remoto (*ISPRS – International Society for Photogrammetry and Remote Sensing*) como “a ciência e tecnologia de extrair informações geométricas e temáticas tridimensionais confiáveis, muitas vezes ao longo do tempo, de objetos e cenas a partir de imagens e dados espaciais”(ISPRS, 2021). Outra definição útil, como fizeram Coelho e Brito (2007) em seu livro, é para o objetivo principal da fotogrametria, que pode ser enunciado como “a reconstrução de um espaço tridimensional, chamado de espaço-objeto, a partir de um conjunto não vazio de imagens bidimensionais, chamado de espaço-imagem”.

Desde o advento da fotografia o estudo de sua aplicação para a documentação do espaço em que vivemos evoluiu bastante. Imediatamente foram descritos trabalhos para a documentação de edifícios e posteriormente, com o advento do avião, foram descritos métodos para fotografias aéreas. Modernamente, temos sensores em órbita da terra e de outros planetas, aeronaves leves não-tripuladas, além de sensores distintos do fotográfico como imageadores ativos ou sensores multi-espectrais.

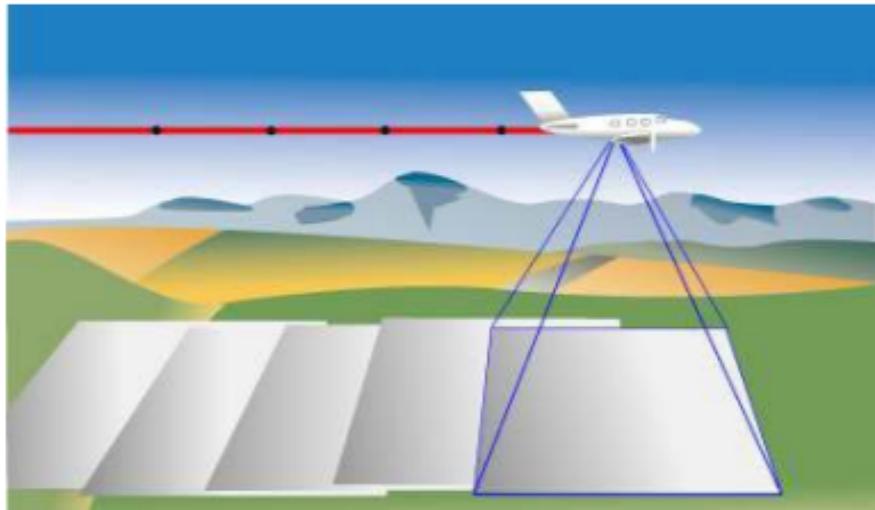
Podemos então classificar a fotogrametria, numa primeira abordagem, segundo o ponto de vista das imagens utilizadas, a saber:

- **Fotogrametria terrestre** que consiste na utilização das imagens primariamente geradas por sensores câmeras terrestres, ela é utilizada primariamente no campo da arquitetura, por exemplo, em restauração de fachadas históricas.
- **Fotogrametria aérea** também tratada como aerofotogrametria, se utiliza de imagens captadas por sensores ou câmaras a bordo de aeronaves, conforme ilustra a Figura 1, e tem como produto resultante a reconstrução da superfície terrestre.
- **Fotogrametria orbital** compartilha inúmeras características com a aerofotogrametria, mas se distingue principalmente pelo uso de sensores orbitais, sendo capazes de observar

---

<sup>1</sup> Para mais informações acesse [www.efoto.eng.uerj.br](http://www.efoto.eng.uerj.br)

Figura 1 - Esquema de captação de imagens em aerolevantamentos.



Fonte: Adaptado de Fricker e Rohrbach (2005)

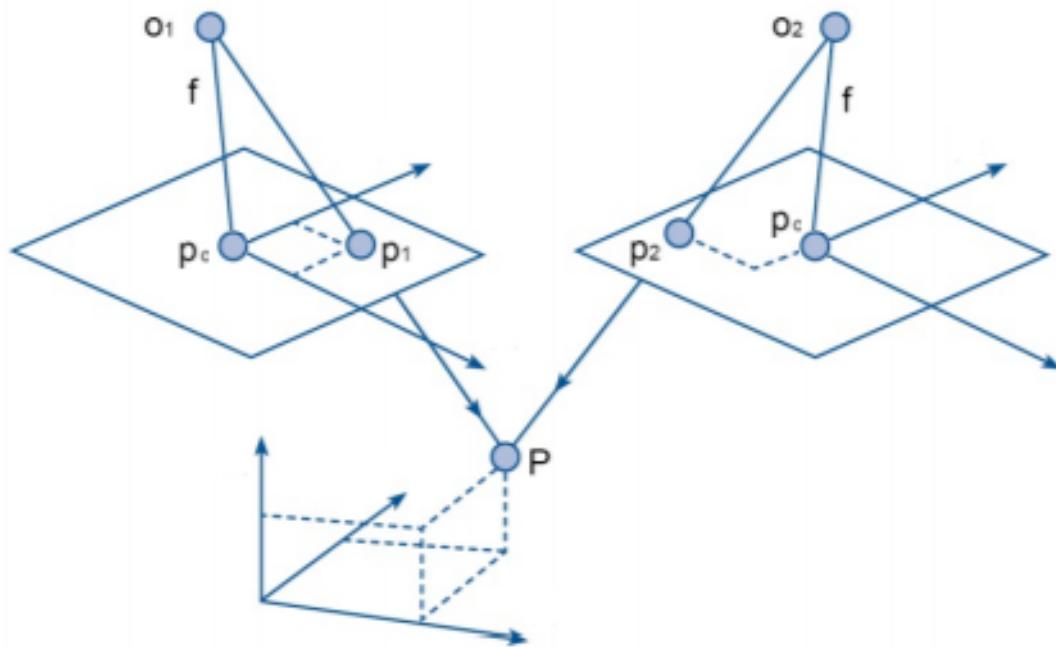
uma porção maior de terreno numa única linha de varredura, com pequenas distorções.

Estruturalmente, o processo de criação de produtos fotogramétricos pode ainda ser dividido em fases de trabalho bem definidas, como as que destacamos:

- **Trabalho de planejamento:** onde ocorre a determinação do objeto ou região de estudo (espaço-objeto), são avaliadas informações locais, como acervos de dados pré-existentes e/ou são traçados planos para um novo imageamento.
- **Trabalho de campo:** onde é realizado o levantamento de pontos notáveis que sirvam como pontos de apoio de campo e/ou controle da qualidade, além do sobrevoo para capturar novas imagens, quando necessários.
- **Trabalho de laboratório:** onde são realizados o ajustamento para os parâmetros de orientação das imagens e a extração dos produtos desejados como modelos digitais de superfície, restituições tridimensionais ou ortomosaicos.

O ajustamento de parâmetros é fundamental para viabilizar todos os produtos que a fotogrametria pode gerar. Este ajustamento só é possível pela identificação de pontos do terreno no conjunto de imagens. Além desse fato, em grandes conjuntos de imagens provenientes de aerolevantamentos é mais eficiente ter pontos fotogramétricos, além dos pontos de controle, os quais representam o mesmo local do espaço objeto em imagens diferentes, sem que seja necessário o conhecimento prévio de suas coordenadas de mundo. São os pontos de controle que associam imagens ao terreno. Logo, estes são importantes para o ajuste absoluto do bloco de

Figura 2 - Reconstrução das coordenadas 3D a partir de imagens 2D.



Legenda:  $O_1$  e  $O_2$  fornecem distintos valores  $(X_0, Y_0, Z_0)$ , pois são, expressos em 3D no sistema de referência do terreno assim como o ponto  $P = (X, Y, Z)$ . O termo  $f$  é a distância focal da câmara usada na produção das imagens. Por sua vez, o ponto  $p_c = (x_0, y_0)$  é o centro óptico da câmara expresso em 2D, pois está sobre espaço das imagens, assim como os pontos  $p_1$  e  $p_2$  com seus respectivos valores  $(x, y)$ .

Fonte: Adaptado de <https://ez-pdh.com/aerial-photogrammetry-help/>

imagens. Por sua vez, os pontos fotogramétricos atuam como pontos de costura para um bloco de imagens seja ajustado relativamente.

Pontos fotogramétricos podem ainda ter suas coordenadas de mundo determinadas por consequência do ajustamento do bloco de imagens. A determinação destas coordenadas é possível pelo processo da triangulação espacial, ilustrado na Figura 2, que pode ser derivado rearranjando os termos da Equação 1, denominada equação de colinearidade. A triangulação de coordenadas 3D só é possível nas regiões de sobreposição imagens, pois é necessário obter ao menos duas observações no espaço das imagens a fim de atender o número de equações que permitam resolver o problema com 3 incógnitas (X, Y e Z).

$$\begin{aligned} x &= x_0 - f \frac{m_{11}(X-X_0) + m_{12}(Y-Y_0) + m_{13}(Z-Z_0)}{m_{31}(X-X_0) + m_{32}(Y-Y_0) + m_{33}(Z-Z_0)} \\ y &= y_0 - f \frac{m_{21}(X-X_0) + m_{22}(Y-Y_0) + m_{23}(Z-Z_0)}{m_{31}(X-X_0) + m_{32}(Y-Y_0) + m_{33}(Z-Z_0)} \end{aligned} \quad (1)$$

Os termos  $m_{ij}$  na equação de colinearidade expressam a rotação entre os planos das imagens e os eixos do sistema de coordenadas adotado para o terreno,  $(X_0, Y_0, Z_0)$  expressa a coordenada do ponto focal da lente da câmera no momento da captura da foto,  $(x_0, y_0)$  expressa o ponto central das imagens registradas e  $f$  expressa a distância focal do sistema sensor.

### 1.1.1 A Estação Fotogramétrica Digital E-Foto

Segundo seus desenvolvedores, o e-foto é uma iniciativa acadêmica de desenvolvimento de software livre para Fotogrametria Digital. Este é fruto principal do Projeto E-Foto, que é desenvolvido no Laboratório de Fotogrametria e Sensoriamento Remoto (LFSR) da Faculdade de Engenharia (FEN) da Universidade do Estado do Rio de Janeiro (UERJ), desde 2004. Através do software o projeto reduz as barreiras de custo para interessados em Fotogrametria (E-FOTO, 2022). É importante também ressaltar sua relevância frente pacotes de softwares proprietários, em geral oferecidos em um modelo caixa preta, que revelam aos usuários pouquíssimos detalhes sobre os cálculos que são feitos, o que é aceitável para produção em larga escala, mas em um ambiente educacional tira a chance de inovação e de capacitação independente de plataformas por parte da comunidade estudantil.

Além do software o Projeto E-Foto possui outros desdobramentos significativos como o fomento para produção de diversos trabalhos de conclusão (como Correa e Laranja em 2011, Ribeiro em 2012 e Regal em 2013), incluindo este trabalho. São exemplos de produtos do projeto cursos, palestras, tutoriais (em texto e vídeo) e apresentações em fóruns nacionais e internacionais. Hoje o Projeto E-Foto é conhecido internacionalmente e atrai entusiastas do mundo inteiro tendo alguns produtos da comunidade usuária com o caso do livro produzido por Vermeer e Ayehu (2018).

Entre os tutoriais para estudo do e-foto mais atuais é observada a subdivisão segundo

os seguintes módulos com foco no trabalho laboratorial de um projeto fotogramétrico: criação e gerenciamento de projetos (*project creation and management*), orientação interior (*interior orientation*), orientação exterior por ressecção espacial (*exterior orientation by space resection*), fototriangulação por feixes perspectivos (*exterior orientation by phototriangulation*), restituição fotogramétrica 3D (*Stereoplotter*), extração do modelo digital de elevações (*DSM Extraction*), ortorretificação (*Orthorectification*) e integração com o QGIS (*Integration with QGIS*).

Nos módulos de criação e gerenciamento de projetos, orientação interior, ressecção espacial e fototriangulação são realizadas atividades que têm por objetivo cadastrar, recuperar ou ajustar os parâmetros internos e externos ao sistema sensor durante o voo e obtenção do conjunto de imagens. São também fornecidos através destes módulos dados extras que possuem relevância para o processo, como sistema de referência adotado, características do sensor e do voo, pontos do terreno e suas medidas no conjunto de imagens que deve ser orientado. Num projeto com o e-foto o usuário só está apto a prosseguir aos demais módulos se todo o conjunto de imagens carregado recebeu parâmetros de orientação interior e exterior.

Está incluso no material de estudos um diagrama, similar ao adaptado pela Figura 3, de onde pode-se inferir que, apesar de cada módulo realizar possuir foco em atividades diferentes, alguns deles geram resultados análogos como, por exemplo, nos módulos de fototriangulação e de ressecção espacial, pois ambos podem ser utilizados em conjunto ou individualmente para finalizar a inicialização dos projetos, gerando os parâmetros de orientação exterior do bloco.

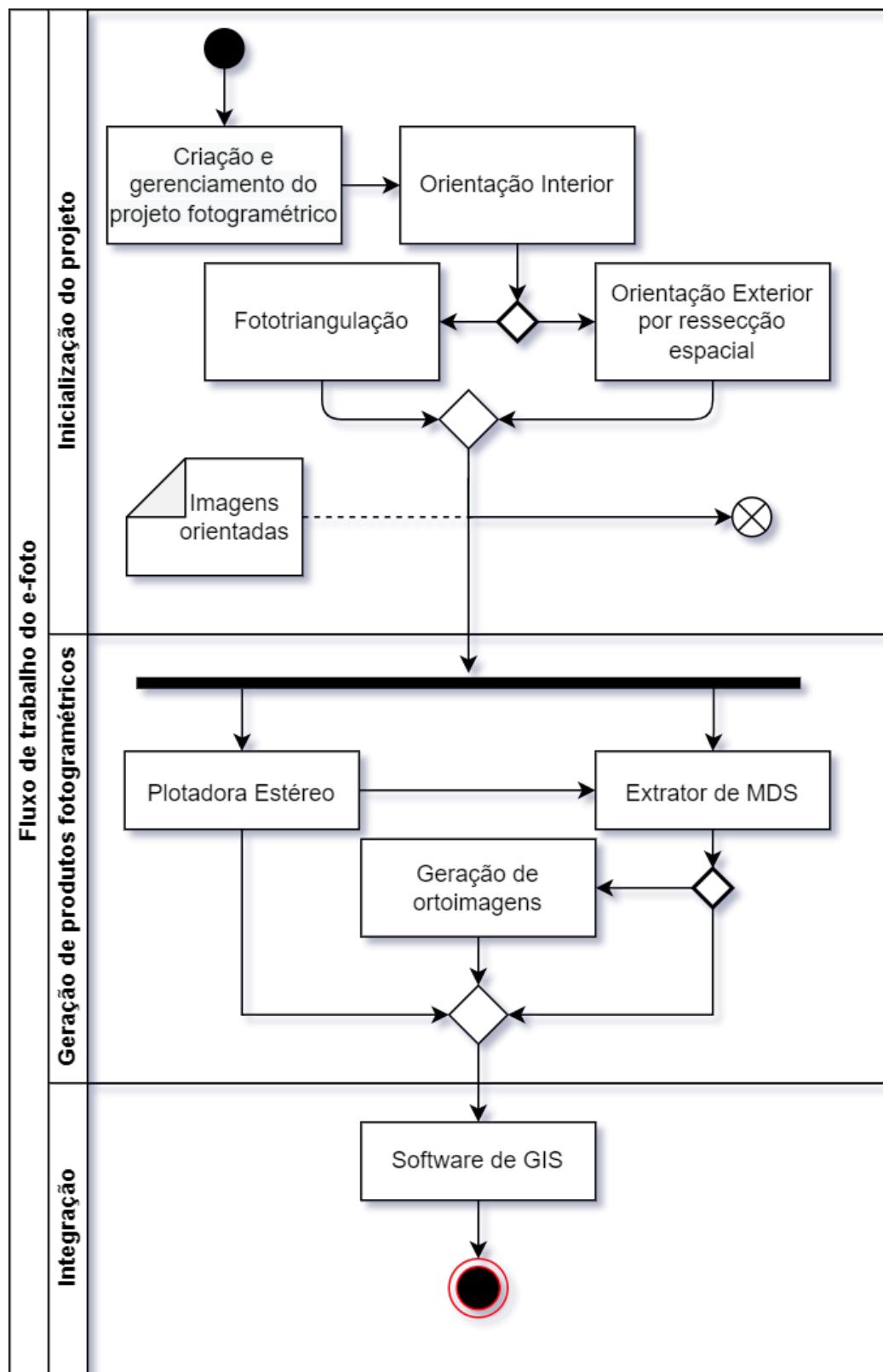
Além dos módulos referidos anteriormente há alguns outros que destinam-se a geração dos produtos fotogramétricos finais. Esses módulos se utilizam das imagens orientadas para criar diversos produtos como: restituição de vetores em 3D, modelos digitais de elevação (MDE) ou orto-mosaicos.

Por sua vez, o processo de medição de pontos homólogos, que preservaram as mesmas feições do terreno em distintas imagens, é realizado em diferentes módulos. Notavelmente isto ocorre tanto durante a atividades da fase de inicialização, como na geração de produtos finais. De modo geral a obtenção destas medidas é seguida pela interseção espacial que obtém as coordenadas 3D da feição no terreno. No entanto, deve ser destacado que estas medidas podem ser automatizadas, como no caso da extração de MDE, ou manuais, como no caso da restituição 3D e da fototriangulação.

Em fato, alguma automação para obter estas medidas seria bem vinda em todas as etapas do software. Por tais motivos, vale destacar a função e estado de desenvolvimento de cada um destes módulos conforme segue:

**Fototriangulação** Este módulo permite o cálculo simultâneo dos parâmetros de orientação exterior do bloco de imagens num projeto. Isto requer que todas as imagens possuam orientação interior prévia. Ele oferece uma interface para visualização e medição de pontos de controle e fotogramétricos. Após a inserção do mínimo de pontos necessários, que depende do número de imagens no projeto, é habilitada a função de executar os cálculos de ajustamento do bloco. Com o ajustamento são obtidas as coordenadas geográficas

Figura 3 - Diagrama de atividades do e-foto



Fonte: Adaptação de [http://www.efoto.eng.uerj.br/images/Documentos/0The\\_workflow\\_of\\_the\\_e-foto\\_software-16.06-v01.pdf](http://www.efoto.eng.uerj.br/images/Documentos/0The_workflow_of_the_e-foto_software-16.06-v01.pdf).

de imagens e pontos fotogramétricos. Apesar de funcional, esse módulo ainda pode receber melhorias já previstas por seus autores, como a automação das medições de pontos fotogramétricos, a indicação de regiões de von gruber para orientar a medição de pontos fotogramétricos e a exclusão de pontos inseridos.

**Restituição 3D** Este módulo permite a visualização estereoscópica de pares de imagens previamente orientadas e a vetorização das feições observadas, isto é, produção de pontos, linhas e polígonos em 3D que possam ser transportados para outras aplicações de produção cartográfica, como o QGIS, pela exportação de dados em formatos de arquivos vetoriais (shapefile). Na exportação, as feições preservam seus rótulos e classes (escolhidas entre um conjunto predefinido pelo e-foto), mas não levam informações extras como perímetro e área. Entende-se que estes atributos são computados no módulo para comodidade do usuário, ainda que possam ser computados externamente nas aplicações de SIG, pois são derivados do dado geométrico em si. Na vetorização, o módulo requer ajustes manuais para que os marcadores de medição sejam posicionadas sobre uma feição, tanto pela necessidade de corrigir a altura da medida quando o usuário obtém vértices em uma superfície desnivelada, quanto pelas distorções notáveis nos limites de modelos, normalmente fruto da falta de normalização das imagens usadas. Logo, a indicação automática de homólogos poderia acelerar a vetorização ao atrair o marcador para a superfície do terreno.

**Extração de MDE** Este módulo foca-se na produção de modelos numéricos para superfícies imageadas, que são definidos segundo o Instituto Brasileiro de Geografia e Estatística (IBGE) como “uma representação das altitudes da superfície topográfica agregada aos elementos geográficos existentes sobre ela, como cobertura vegetal e edificações.” (IBGE, 2022). Para tanto, são utilizados os pontos que foram medidos em módulos anteriores, adotando-os como sementes para realizar um algoritmo de crescimento de regiões (*region growing*), isto é, são buscados novos pontos homólogos na vizinhança das sementes e computadas as alturas usando-se de intersecção espacial. Em seu estado atual este módulo é usado com poucas sementes, por contar apenas com a medição manual dos pontos de controle e fotogramétricos. Isto implica no crescimento regiões muito amplo e, por vezes, impossível devido as descontinuidades naturais do terreno. Algo que leva usuários a terem de medir mais sementes manualmente antes da execução do módulo.

## 1.2 Visão Computacional

A visão computacional é definida pela International Business Machines Corporation (IBM) como, “Um campo da inteligência artificial (IA) que permite que computadores e sistemas obtenham informações significativas de imagens digitais, vídeos e outras entradas visuais e tomem ações ou façam recomendações com base nessas informações” (IBM, 2022). Este

campo evolui desde o início da década de 1960, tendo se concentrado na busca de padrões para interpretar imagens e para recuperar a geometria da cena. Ele está intimamente ligada aos modelos de aprendizado de máquina (*Machine Learning*) e a melhora do hardware básico para este fim proporcionou um salto na precisão e no número de aplicações desde a década de 2010 (ACM, 2022). Deste modo, em seu estado atual, visão computacional permite avaliar relações de nível superior como quais objetos estão interagindo uns com os outros, em qual contexto e como a cena provavelmente irá progredir.

Pela inovação e eficácia que as aplicações desta área do conhecimento trazem, elas vêm sendo adotadas em diversas outras áreas, sendo cada vez mais essencial em análises de imagens laboratoriais (GAO et al., 2018), na construção civil (FANG et al., 2020; RUIZ; LERMA; GIMENO, 2002), na indústria alimentícia (GOMES; LETA, 2012) entre outros.

Um dos artigos iniciais mais notáveis sobre o assunto foi intitulado o “*Receptive fields of single neurones in the cat's striate cortex*” (HUBEL; WIESEL, 1959) no qual os autores descrevem as respostas dos neurônios corticais visuais de um gato ao exporem ele a diferentes imagens. Como descrito no artigo os pesquisadores observaram melhores resultados quando, ao trocar as imagens projetadas, eles perceberam que um neurônio específico respondia. Mais tarde eles descobriram ser as bordas das transparências que formava uma sombra nítida. Assim eles estabeleceram que o processamento visual sempre se inicia com estruturas mais simples, o que pode ser considerado um dos primeiros grandes passos para a visão computacional.

Em 1982 com a publicação do livro “*Vision: A computational investigation into the human representation and processing of visual information*” (republicado em 2010 por Marr) utiliza-se a base criada por Hubel e Wiesel e é estabelecido que o mecanismo da visão é hierárquico. Então, ele impõe que a principal função desse mecanismo é criar representações 3D a partir das cenas capturadas do ambiente para que possamos interagir com ele. Assim foi estabelecida uma base na qual são tratadas questões de baixo custo computacional, se comparadas aos problemas de detecção direta, para fomentar novos algoritmos de detecção de mais alto nível.

Notavelmente um grande salto na área de reconstrução geométrica resultou de um estudo de reconhecimento de objetos baseado em feições pontuais, isto é, foi devido principalmente ao foco em interpretação que a criação de modelos 3D pode experimentar rápida evolução. Isto é observado com a publicação do artigo “*Object Recognition from Local Scale-Invariant Features*” (LOWE, 1999b), no qual são descritos sistemas de reconhecimento visual que usa feições invariáveis a escala. Desde a publicação deste artigo diversos outros seguiram abordagens estruturais semelhantes, o que possibilitou a formalização de um *pipeline* típico para a construção de aplicações baseadas em correlação de imagens.

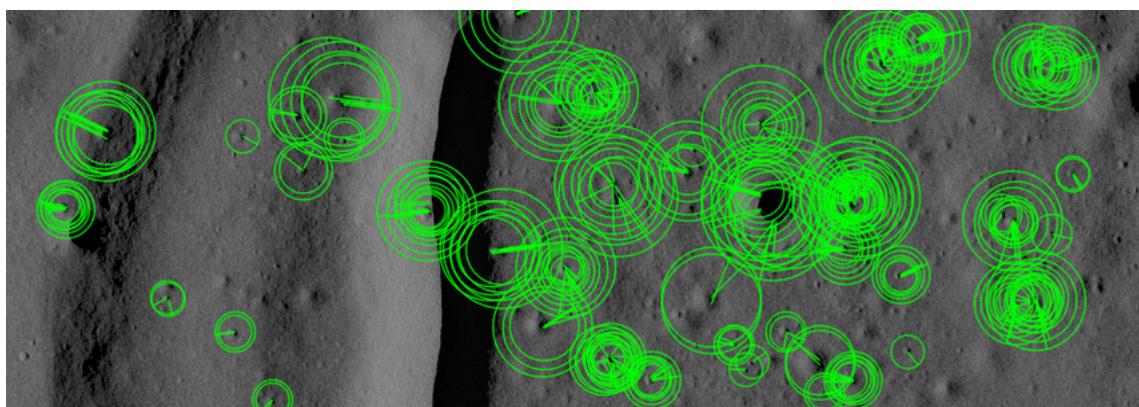
Em aplicações que atuem prioritariamente no espaço 2D, como estabilização de vídeo, alinhamento de rostos ou criação de imagens panorâmicas este *pipeline* é adotado, mas este pode servir ainda a criação de modelos 3D. Isso se justifica, uma vez que todo o processamento típico se inicia no espaço das imagens e permanece neste até que a relação do conjunto esteja muito bem definida. Neste contexto, os métodos de visão computacional para feições 2D em sua

maioria estão focados na produção de respostas confiáveis em 4 ou mais estágios, a saber:

- Detecção de pontos chave em imagens
- Extração de descritores para pontos
- Correlação de descrições de pontos
- Restrição da solução por geometria

A detecção de pontos chaves é o primeiro momento de qualquer método de análise de feições 2D, onde cada imagem é analisada e é criada uma coleção massiva de coordenadas do espaço das imagens, como ilustrado na Figura 4, que podem estar atreladas a outras informações, como rotações e escalas de obtenção. O que distingue um bom ponto chave é sua invariância, isto é, capacidade de ser diferenciado de outros pontos em sua vizinhança, sejam estes obtidos por deslocamento, rotação ou escala no espaço da imagem. Trabalhos preliminares neste sentido focaram principalmente imagens de uma mesma escala (HARRIS; STEPHENS et al., 1988), mas isto não impede que sejam aplicados em diferentes escalas se o usuário destes algoritmos constrói uma pirâmide de imagens e preserva a informação do nível onde cada ponto chave foi detectado.

Figura 4 - Exemplo de representação para pontos chave



Legenda: Os círculos estão centrados nos pontos chaves que representam; a orientação é indicada com a linha que parte do ponto chave em direção ao círculo que o representa; e o raio do círculo é proporcional a redução de necessária da imagem para sua obtenção.

Fonte: Adaptado de Ciarambino et al. (2016)

A descrição é a aplicação de alguma função nas vizinhanças de um ponto chave previamente detectado e resulta na preservação das características invariantes do ponto. Descritores são desenvolvidos para estar aptos a verificação de semelhança segundo alguma técnica de busca de dados ou cálculo de distâncias entre pontos no espaço de descrição. Entre as abordagens mais

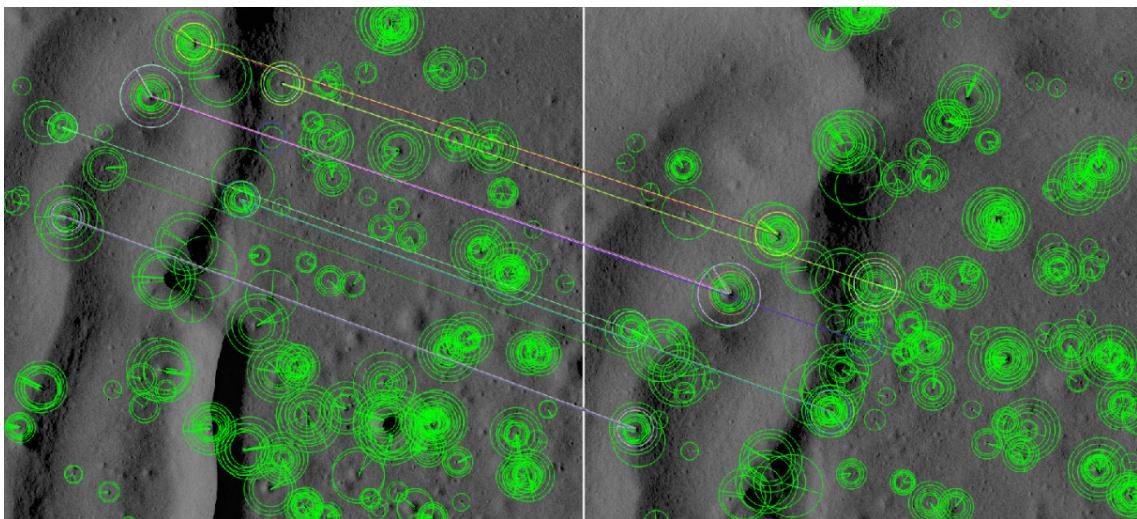
comuns de desenvolvimento de descritores estão as abordagens baseadas em histogramas de gradientes orientados (LOWE, 1999b; BAY; TUYTELAARS; GOOL, 2006), computados em derivadas de imagens, e a descrição de por strings binárias (ALCANTARILLA; NUOVO; BARTOLI, 2011; RUBLEE et al., 2011), geralmente computados diretamente sobre as intensidades de *pixels* nas imagens.

O estágio de correlação é o momento em que são comparadas as descrições dos pontos obtidos em uma imagem com o conjunto de descritores de uma outra imagem, ou de um conjunto de descritores de diversas imagens, a fim de encontrar as melhores correspondências para cada ponto. A estratégia adotada de forma mais ampla, para toda a imagem, é significativamente razoável se não há qualquer conhecimento prévio de onde ou entre quais imagens as correspondências devem ocorrer. Embora seja relativamente mais cara do que as técnicas de rastreamento de pontos, que perfazem buscas em conjuntos limitados de pontos e imagens, mas só estão disponíveis quando há como especular qual é a relação entre o conjunto de imagens. Por sua vez, as técnicas de correlação são frequentemente adotadas para inicializar (ou restaurar) o rastreamento, em aplicações de estabilização de vídeo por exemplo, sendo assim ambas as técnicas complementares entre si.

Ainda no contexto de correlações é possível considerar se cada hipótese de combinação entre pares de pontos precisa ser avaliada (abordagem de força bruta) ou se é possível explorar o espaço de descrição para realizar buscas por vizinhos mais próximos. A varredura extensiva de listas de descritores pode ser considerada uma solução precisa, mas pouco escalável, pois o custo de comparação de  $K$  pontos em uma lista com  $M$  outros pontos implica em  $K \times M$  operações. Dentre as soluções alternativas destacam-se o uso de estruturas de dados capazes de emular o espaço de  $N$  dimensões de um descritor, permitindo assim que algoritmos de busca por vizinhos mais próximos aproximados sejam usados tendo em vista que alguns podem levar a um número de operações que é aproximadamente  $K \times \log N$ .

Em ambos os casos de correlação a resposta para todos os pontos não é necessariamente o melhor resultado. Há que ser considerado que muitas correlações encontradas entre pares de pontos poderiam ser eliminadas por um limiar de distância. Estabelecer esse limiar ainda é um problema quando temos diferentes espaços de descrição a considerar. Outra questão importante é que a ordem de disposição das imagens num par analisado pode gerar resultados distintos. Algo aceitável em determinadas aplicações, como a busca por objetos, mas que deve ser evitado em aplicações como a costura de imagens. Por sua vez, esta diferença em ambos os sentidos pode ser explorada para que a verificação cruzada preserve apenas as melhores correlações, quando elas concordarem entre si. Isto é útil principalmente nas aplicações de costura de imagens. Conforme ilustrado na Figura 5 as melhores correlações, restritas por verificação cruzada, oferecem bom entendimento do movimento relativo entre as imagens num par. Outras alternativas podem ser previstas para que pontos chave que parecem correlacionar bem com mais de um ponto sejam deixados de fora, evitando assim o problema de pontos em regiões que podem ser considerados padrões repetitivos.

Figura 5 - Exemplo de correlação



Legenda: Linhas em cores distintas indicam diferentes feições correlatas entre a imagem da esquerda e a da direita.

Fonte: Adaptado de Ciarambino et al. (2016)

A melhor receita para robustecer a solução, e evitar que a aquisição automática de correlatos gere impactos ao processamento, é restringi-la usando o conhecimento prévio de como as imagens se formaram. Neste sentido, entre dois quadros obtidos por uma câmera, podemos encontrar uma transformação projetiva que leve os pontos de uma imagem em outra. Ao analisar os resíduos do transporte de pontos pela transformada, quando comparados com seus pares indicados pelas correlações, é possível destacar os conjuntos de pontos que permanecem na solução (*inliers*) dos conjunto de pontos tratados como erros grosseiros (*outliers*). Isto é feito em conjunto com técnicas de iteração e análise de consenso, para que possa ser verificado o grau de confiabilidade da resposta.

### 1.2.1 A biblioteca OpenCV

Lançado por Gary Bradski em 1999, enquanto trabalhava na Intel Corporation, na esperança de acelerar o desenvolvimento da visão computacional e da inteligencia artificial ao criar e fornecer uma infraestrutura sólida para todos que trabalham na área. Esta biblioteca foi escrita em C inicialmente, e portada ao C++ depois da versão 2.4.

Esta biblioteca é desenvolvida como software livre e utilizável em todos os principais sistemas operacionais (OPENCV, 2022). Em sua área de concentração ela é umas das soluções mais disponíveis na atualidade, com grande número de interfaces para desenvolvimento de aplicações (APIs), tendo compatibilidade com linguagens como o Python, o Java, o Javascript

e o C#, além da linguagem C++, na qual é desenvolvida. Estão disponíveis também interfaces para uso conjunto com aplicações como o MATLAB e o GNU Octave.

Devido a sua difusão não é difícil obter apoio para estudos dos conceitos deste campo, pois há muitos exemplos e tutoriais didáticos, questões e respostas em fóruns especializados, além de livros, cursos e códigos de aplicações *open source* que fazem uso ativo desta biblioteca. Sua documentação, embora rudimentar em alguns aspectos, está ligada aos diversos trabalhos acadêmicos em que se basearam, desde que disponibilizados diretamente por seus autores.

A OpenCV está dividida em módulos e estes podem ainda ser divididos entre: principais, que formam a base estável e completamente open source; e extras, onde encontramos códigos de contribuidores isolados, experimentais ou patenteados, mas que já fazem parte da biblioteca por sua grande contribuição para a evolução da área. Mesmo com alguns recursos patenteados a biblioteca pode ser utilizada para fins acadêmicos, estando vedada a distribuição de binários executáveis com tais recursos para outros fins. Os módulos principais das versões mais recentes são listados na Tabela 1.

Tabela 1 - Módulos da OpenCV 4.5.5

Módulo	Descrição
core	Funcionalidade central
imgproc	Processamento de imagens
imgcodecs	Leitura e escrita dos formatos de imagem
videoio	Leitura e escrita dos formatos de video
highgui	interface gráfica de alto nível
video	Analise de video
calib3d	Calibração de câmera e reconstrução 3D
features2d	estrutura de recursos 2D
objdetect	Detecção de objetos
dnn	rede neural profunda
ml	aprendizado de máquina
flann	Clustering e Pesquisa em Espaços Multidimensionais
photo	Fotografia computacional
stitching	costura de imagens
gapi	Interface de gráficos

Fonte: ‘O autor’.

O módulo *features2d* apresenta a maior parte dos recursos necessários para a realização do presente trabalho. Entre os algoritmos integrados encontram-se o SIFT<sup>2</sup> que há mais de 20 anos possibilitou um salto na detecção de objetos baseada em feições pontuais. Diversos outros

---

<sup>2</sup> SIFT refere-se a *Scale Invariant Feature Transform* e foi publicado por Lowe em 1999b

algoritmos mais atuais, como o AKAZE<sup>3</sup>, ORB<sup>4</sup> e SURF<sup>5</sup> que reivindicam melhorias específicas e surgem como alternativas a serem comparadas em diversos cenários.

Discutiremos no Capítulo 2 alguns recursos estendidos que podem ser obtidos nos módulos *xfeatures2d* e *cudafeatures2d*. São também importantes para o produto final os módulos *core*, *imgproc* e *calib3d*. Os módulos *highgui* e *stitching* contribuem significativamente para este estudo, mas não são adotados como parte do produto final. Todavia, são igualmente relevantes para estudos futuros, pela proximidade com o assunto de fotogrametria, os módulos extras *ccalib* (padrões personalizados de calibração para reconstrução 3D), *optflow* (fluxo óptico), *reg* (registro de imagens), *rgbd* (processamento com sensores de profundidade), *sfm* (estruturação por movimento), *stereo* (correspondência densa), *structured\_light* (processamento de luz estruturada), *surface\_matching* (correlação de superfícies) e *tracking* (rastreamento).

---

<sup>3</sup> AKAZE refere-se a *Accelerated-KAZE* e foi publicado por Alcantarilla, Nuovo e Bartoli em 2011

<sup>4</sup> ORB refere-se a *Oriented FAST and Rotated BRIEF* e foi publicado por Rublee et al. em 2011

<sup>5</sup> SURF refere-se a *Speeded Up Robust Features* e foi publicado por Bay, Tuytelaars e Gool em 2006

## 2 METODOLOGIA

O fluxo de trabalho adotado incluiu etapas de estudo do problema e das ferramentas disponíveis para tratá-lo, definição de critérios de análise para a tomada de decisão entre as opções disponíveis, planejamento de testes comparativos e codificação de um programa com interface de linha de comando para concretizar uma solução minimamente viável que possa ser integrada ao Projeto E-Foto em suas versões futuras.

### 2.1 Estudo do cenário

Entre os objetos a serem estudados para a confecção do trabalho destacam-se o levantamento das necessidade do software (e-foto), das ferramentas computacionais que implementam partes essenciais da solução e dos conjuntos de dados disponíveis para realização de testes.

#### 2.1.1 Levantamento das necessidades do e-foto

No início do projeto, foi considerado o estado atual do software levando em consideração atualizações não publicadas aos usuários, seja por meio de suas documentações ou de seus lançamentos de versões executáveis, mais de conhecimento dos principais desenvolvedores ativos no projeto. Foram identificados ainda quais módulos se beneficiariam das possíveis respostas deste trabalho e escolhido um módulo alvo, como prova de conceito.

Para tal, foi preciso realizar um análise mais aprofundada destes módulos visando o entendimento dos dados que eles necessitam para funcionar e como estes são entregues. O alvo escolhido foi aquele com menor dependência de acoplamento, isto é, que permite o mínimo de alteração no software, atuando assim como um provedor de dados isolado a ser integrado posteriormente ao considerar uso dos formatos de arquivos suportados atualmente.

Cabe ressaltar que existe um método pouco documentado do e-foto, elaborado especificamente para o desenvolvimento e testes do módulo de aerotriangulação (PUPIM, 2012), que permite a importação de pontos através de 2 arquivos de texto “.txt”. Estes arquivos são preenchidos com as informações de ENH dos pontos, ou seja, suas informações geográficas, e a localização desses pontos quando medidos nas imagens. Isto possibilitou o foco do trabalho na comparação de técnicas, deixando para o software conjuntos de dados que possam ser importados em seu ambiente.

O módulo alvo escolhido realiza a aerotriangulação. Ele adota pontos fotogramétricos, que equivalem aos pontos de costura até o momento em que o módulo efetua seus cálculos, pois estes pontos adquirem coordenadas de terreno como consequência do processo de ajustamento

do bloco de imagens. Atualmente estes pontos devem ser medidos manualmente nos conjuntos de imagens de um bloco fotogramétrico, antes que o processo possa ser executado.

Dentre as opções deixadas fora do escopo deste trabalho, mas consideradas na etapa de estudos, estão os módulos de restituição estereoscópica e de extração de modelos digitais de superfície. Ressalta-se que estes módulos não poderiam ser abastecidos com os dados deste trabalho sem alteração direta do e-foto. Contudo, os resultados da fase de estudo para estes módulos serão compartilhadas no Capítulo 3 como referências para trabalhos futuros. Outra opção, também descartada, foi a confecção de um novo módulo completo para seleção de pares e visualização de foto-índices. Mais uma vez, cabe ressaltar que este trabalho pode ser considerado como uma das bases para o novo módulo.

### 2.1.2 Seleção de ferramentas aplicáveis na solução

Foram consideradas as bibliotecas de visão computacional que ofertassem métodos para detecção e descrição de pontos chaves, bem como para realizar correlação de feições e correções geométricas. Uma qualidade avaliada foi a portabilidade, ou seja, capacidade de funcionar nos principais sistemas operacionais da atualidade. Também foram consideradas as linguagens de programação disponíveis para que os resultados pudessem ser incorporados ao e-foto futuramente.

Muitas opções de APIs (interfaces de programação de aplicações) para visão computacional foram publicadas nos últimos anos. Desde opções abertas à comunidade desenvolvedora como é o caso de OpenCV (apresentada em detalhes no Capítulo 1), SimpleCV<sup>6</sup>, BoofCV<sup>7</sup> e Scikit-image<sup>8</sup>, até soluções corporativas em redes, como é o caso da Amazon<sup>9</sup>, da Google<sup>10</sup> e da Microsoft<sup>11</sup>. Dentre tantas opções, merecem destaque o Orfeo-Toolbox e o Opticks, pois tratam-se de projetos de código aberto focados no processamento de imagens de sensoriamento remoto com suporte da Open Source Geospatial Foundation (OSGeo), uma organização sem fins lucrativos cuja missão é promover a adoção de tecnologias geoespaciais abertas à comunidade.

Optou-se por utilizar neste trabalho o OpenCV, uma das mais completas bibliotecas de visão computacional, pois ela é desenvolvida em linguagem C++ (tendo usado interface em C até

<sup>6</sup> SimpleCV: plataforma de aplicativos de visão computacional em Python disponível em <<https://simplecv.org>>

<sup>7</sup> BoofCV: biblioteca aberta de visão computacional de tempo real em Java disponível em <<https://boofcv.org/>>

<sup>8</sup> Scikit-image: coleção de algoritmos para processamento de imagens elaborados pela comunidade Python disponível em <<https://scikit-image.org>>

<sup>9</sup> AWS Rekognition disponível em <<https://aws.amazon.com/pt/rekognition>>

<sup>10</sup> Google Cloud Vision disponível em <<https://cloud.google.com/vision>>

<sup>11</sup> Azure Computer Vision disponível em <<https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision>>

sua versão 2.4), além de possuir um número considerável de exemplos. Ela é também base para outros aplicativos de processamento de imagens aéreas como o Open Drone Maps<sup>12</sup> (ODM).

Outras características interessantes do OpenCV são uma boa portabilidade entre sistemas operacionais, além de interfaces com outras linguagens de programação como o Python, permitindo assim seu uso na extensão de aplicações. Deste modo, entende-se que ela se alinha com critérios básicos de seleção de dependências para o e-foto e pode vir a ser uma ferramenta de apoio em suas versões futuras. Logo, estima-se que tal escolha auxilie na integração dos resultados com o e-foto, mesmo considerando diferentes cenários de curso do seu desenvolvimento, principalmente por este também ser desenvolvido utilizando-se da linguagem C++, mas também por este já adotar chamadas para outros programas de linha de comando, como o *xsltproc* (RIBEIRO, 2012) e pelo potencial para a adesão de novos alunos ao projeto interessados em trabalhar com outras linguagens de programação.

A versão escolhida de OpenCV foi a 4.5.4, que não é a versão mais recente da biblioteca, mas inclui diversas melhorias consideradas importantes neste trabalho. O principal motivador para a escolha da versão foi a observação de que os repositórios de arquivos binários para diversas distribuições Linux já dispõem desta versão<sup>13</sup>. Ela ainda pode ser utilizada em Windows e Mac através de binários disponibilizados em repositórios do tipo Conda<sup>14</sup>.

Em OpenCV estão muitos dos trabalhos mais recentes de pesquisa da área de visão computacional. No que toca este trabalho, a Tabela 2 indica os algoritmos do módulo de análise de feições em 2D, disponíveis na versão escolhida da biblioteca. São listadas as implementações disponíveis para diferentes algoritmos de detecção e descrição de feições, indicando onde estes se aplicam. Em grande parte as classes disponíveis são consideradas experimentais ou possuem patentes atreladas ainda vigentes e, por este motivo, tais algoritmos são isolados na extensão “xfeatures2d” que implica em dependência de códigos de terceiros. Outra dependência comum a algumas partes desta biblioteca é o “CUDA” (acrônimo de *Computer Unified Device Architecture*), que é uma arquitetura de processamento paralelo de propósito geral desenvolvida pela NVIDIA permitindo uso de suas placas de vídeo (GPUs). O uso de processamento paralelo pode reduzir o tempo de resposta de sistemas, porém aumenta significativamente o custo de equipamentos compatíveis. Neste sentido, para o Projeto E-Foto desconsideramos no trabalho o uso de quaisquer recurso CUDA e da maioria dos algoritmos considerados experimentais.

Diante da grande variedade de algoritmos disponíveis, e pela concentração de exemplos num pequeno conjunto destes, que permitiam a execução de teste alterando minimamente os códigos iniciais, optou-se pela redução dos algoritmos adotados. Então quatro opções foram

---

<sup>12</sup> Software com código disponível em <https://github.com/OpenDroneMap/> que depende de OpenSfM hospedado em <https://github.com/mapillary/OpenSfM>

<sup>13</sup> Para mais informações acesse <https://repology.org/project/opencv/versions>

<sup>14</sup> Disponível em <https://anaconda.org/conda-forge/libopencv>

Tabela 2 - Implementações da interface Feature2D

Algoritmo	Detecção	Descrição	Dependência
AffineFeature	✓	✓	-
AgastFeatureDetector	✓	✗	-
Akaze	✓	✓	-
BRISK	✓	✓	-
FastFeatureDetector	✓	✗	-
GFTTDetector	✓	✗	-
KAZE	✓	✓	-
MSER	✓	✗	-
ORB	✓	✓	-
SIFT	✓	✓	-
SimpleBlobDetector	✓	✗	-
AffineFeatures2D	✓	✓	xfeatures2d
BEBLID	✗	✓	xfeatures2d
BoostDesc	✗	✓	xfeatures2d
BriefDescriptorExtractor	✗	✓	xfeatures2d
DAISY	✗	✓	xfeatures2d
FREAK	✗	✓	xfeatures2d
HarrisLaplaceFeatureDetector	✓	✗	xfeatures2d
LATCH	✗	✓	xfeatures2d
LUCID	✗	✓	xfeatures2d
MSDDetector	✓	✗	xfeatures2d
StarDetector	✓	✗	xfeatures2d
SURF	✓	✓	xfeatures2d
VGG	✗	✓	xfeatures2d
FastFeatureDetector_CUDA	✓	✗	cuda
ORB_CUDA	✓	✓	cuda
SURF_CUDA	✓	✓	cuda

Legenda: (-) indica nenhuma dependência externa.

Fonte: ‘O autor’.

selecionadas, todas capazes de operar com feições rotacionadas em diferentes escalas, sendo duas delas baseadas em gradientes de imagens (SIFT e SURF) e outras duas baseadas em descrição binária de feições. Cabe informar que SURF foi adotado, a título de comparativo, como possível sucessor do SIFT, mas que este algoritmo ainda possui patente vigente. Deste modo, todos os usos de SURF neste trabalho foram isolados em uma compilação própria do OpenCV com opções particulares que tipicamente não são usadas ao gerar pacotes para distribuição nos repositórios de sistemas. Então o código deste trabalho desativa o SURF caso seja construído com as bibliotecas disponíveis nos repositórios de binários dos sistemas operacionais típicos.

Fez-se necessário instanciar mais de uma máquina virtual (VM, acrônimo de *Virtual Machine*) para testar as diferentes configurações da OpenCV. Logo, foi criada uma VM com sistema operacional Ubuntu 20.04, para a qual foram alocados aproximadamente 8 GB de RAM e 2 núcleos do processador, além de uma segunda VM com Ubuntu 22.04 com configurações semelhantes. Cabe observar que ambas são versões LTS do Ubuntu<sup>15</sup> sendo esta a mais recente.

Foi usado um *script*, cedido pelo prof. orientador e disponível no Anexo A, para obter e compilar a OpenCV com todos os seus recursos na primeira VM. Tal *script* pode ser configurado para qualquer versão alvo, maior que 2.4, da biblioteca. Tendo em vista que no decorrer do trabalho observou-se, através da documentação da biblioteca, uma total integração do método SIFT, desde 11/10/2021, devido ao término de vigência de sua patente (LOWE, 1999a), tornou-se oportuno realizar testes com a segunda VM, onde a OpenCV foi obtida do repositório padrão.

Além da detecção e descrição de pontos chaves, os algoritmos para correlação e a verificação geométrica das soluções tiveram de ser adotados. Neste escopo, foram testadas as principais opções disponíveis que não estivessem listadas como experimentais. São apresentadas na Tabela 3 todos os algoritmos para correlação disponibilizados pela OpenCV na versão 4.5.4.

Tabela 3 - Implementações da interface DescriptorMatcher

Algoritmo	Dependência
BFMatcher	-
FlannBasedMatcher	-
GMSMatcher	xfeatures2d
LOGOSMatcher	xfeatures2d

Legenda: (-) indica nenhuma dependência externa.

Fonte: ‘O autor’.

Por fim, a verificação geométrica da solução, utilizando-se da transformada homográfica entre pares de imagens está disponível com quatro métodos na OpenCV, sendo o ajustamento pelo método dos mínimos quadrados a opção mais básica oferecida. Tal método necessita uma

---

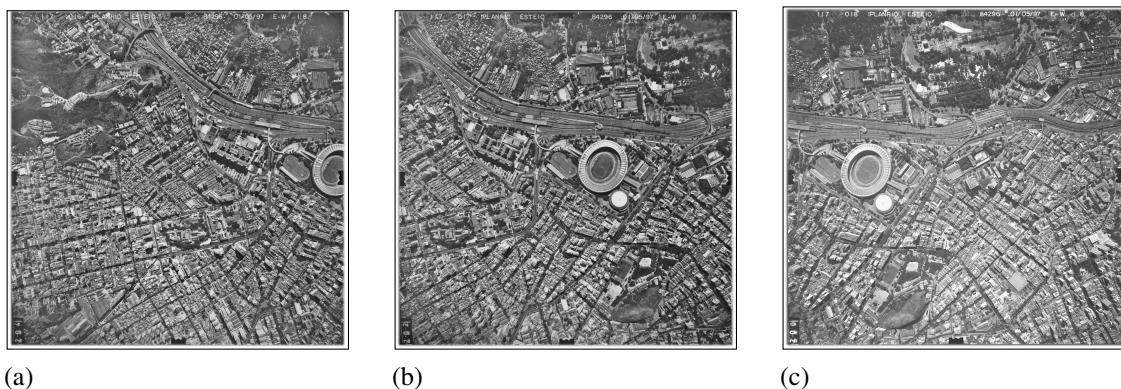
<sup>15</sup> Versões do tipo Long Term Support possuem suporte por 5 anos e são lançadas a cada 2 anos

entrada de pontos bem comportados e com o mínimo possível de falsas correlações, pois por ser um método estatístico clássico acaba tendo seu resultado final gravemente afetado quando há quantidades significativas de erros na informação processada. Tendo em vista o uso da descrição de pontos chaves em termos de poucas informações de seus entornos para montagem do conjunto de feições correlatas, entende-se que estes são conjuntos de dados propensos a erros e que necessitam de soluções de estatística robusta.

Entre as alternativas robustas disponíveis destaca-se o RANSAC (FISCHLER; BOLLES, 1981),acrônimo para *Random Sample Consensus* ou consenso de amostras aleatórias, que tem a capacidade de identificar e descartar grandes quantidades de dados com erros grosseiros. Outro método disponibilizado é o LMedS (MASSART et al., 1986),acrônimo de *Least median of squares* ou mínimos quadrados aparados, para tratamento de dados com erros sendo uma melhor opção para uso geral se comparado com o método dos mínimos quadrados clássico. Por fim, PROSAC (CHUM; MATAS, 2005),acrônimo para *Progressive Sample Consensus* ou consenso de amostra progressiva, inserida em OpenCV por Bazargani, Bilaniuk e Laganiere (2018), produz uma resposta mais adequada aos problemas de tempo real, com foco na correlação de superfícies planas. Neste estudo, adotou-se o RANSAC por sua generalidade e como limitação de escopo.

### 2.1.3 Conjuntos de dados para testes

Figura 6 - Sequência de imagens adotadas em UERJ.IO.epp



Legenda: (a) 16; (b) 17; e (c) 18

Fonte: Arquivo do Projeto E-Foto sobrevoo UERJ de 1997.

Um conjunto de imagens de exemplo do e-foto<sup>16</sup>, conforme observado na Figura 6, foi

<sup>16</sup> O conjunto de imagens pode ser obtido em (<http://www.efoto.eng.uerj.br/download/imagens-e-dados>)

adotado nos testes principais deste trabalho. Por simplicidade, estas imagens de sobrevoo da região da Tijuca, com foco na área da UERJ, realizado em 1997 e cedido ao LFSR pelo IPP, serão denominadas 16, 17 e 18 quando exibidas no Capítulo 3.

## 2.2 Definição dos critérios de análise das opções disponíveis

Com a fase de estudos finalizada, fez-se indispensável a definição de critérios para comparação entre os métodos disponíveis, tanto para detectar e descrever feições, quanto para compará-las e restringir o conjunto com uma análise geométrica. Estes critérios devem permitir o entendimento de suas diferenças, quantitativas e qualitativas, e como eles se comportariam para imagens de relevância fotogramétrica. Adotaram-se então 4 critérios de análise:

- Tempo de execução e memória alocada
- Distribuição e volume das respostas
- Verificação da qualidade
- Viabilidade de filtragem dos melhores pontos

### 2.2.1 Tempo de execução e memória alocada

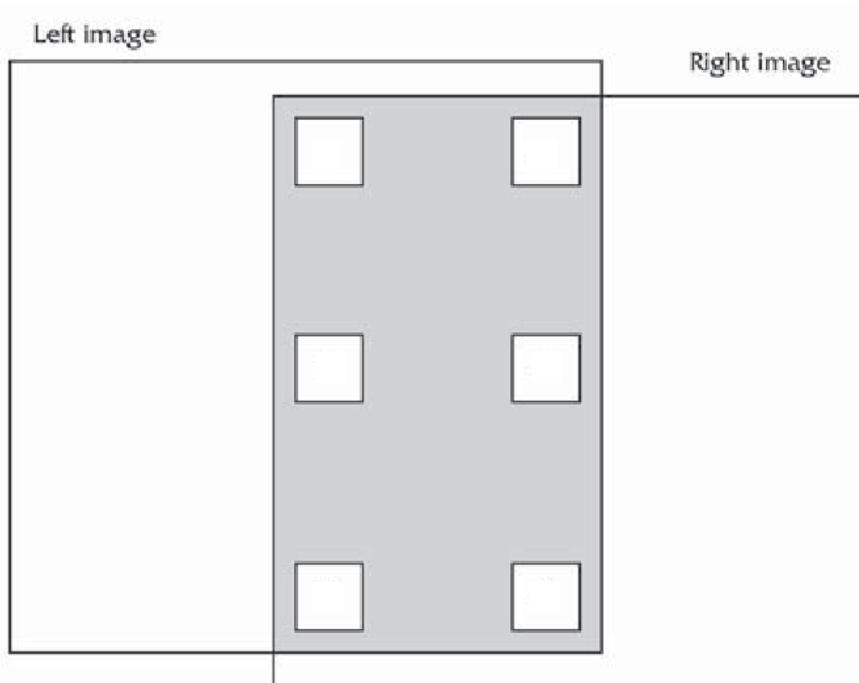
Para análise do tempo de execução foram isolados pontos específicos dos códigos testados e em desenvolvimento com a biblioteca “Chrono”, que é parte das bibliotecas nativas do C++11, e adotado retorno de intervalos em milissegundos (ms). Para a quantificação de memória alocada foram feitas macros de cálculo do espaço reservado para as principais estruturas em uso em cada código analisado. Entende-se que a escolha de pontos específicos do processamento, bem como de algumas estruturas principais, não reflete o total de tempo e memória usados. Contudo, é razoável esperar que seja possível averiguar, pela diferença entre as somas obtidas e um contador externo ao programa (como a função *time -v <program>* do sistema Linux), se as partes selecionadas são realmente as mais significativas de todo o processo.

### 2.2.2 Distribuição e volume das respostas

Este critério implica na percepção visual do preenchimento da região de sobreposição das imagem com pontos correlatos que puderam permanecer até o fim do processo, assim se tornando pontos de costura, bem como do decaimento do número de pontos chaves e de correlações após as etapas de refinamento como a verificação geométrica. Para análise e avaliação da distribuição

de respostas será utilizado o conceito de regiões de gruber (MIKHAIL et al., 2006). Tal conceito afirma que apenas quando temos pontos bem distribuídos e de preferência cobrindo distintas regiões de interesse na área de sobreposição das imagens, conforme a ilustra a Figura 7, podemos ter uma solução robusta e que elimine a paralaxe vertical.

Figura 7 - Regiões de interesse para distribuição de pontos de gruber



Fonte: Imagem adaptada de Mikhail et al. (2006)

### 2.2.3 Verificação da qualidade

Para este critério de análise podem ser considerados tanto a análise visual dos pontos medidos após sua carga no módulo de aerotriangulação ou numa imagem do par, quanto o valor numérico de erro quando usamos uma transformação projetiva (homografia) para projetar pontos de uma imagem sobre a outra na qual há correlatos previamente estimados.

### 2.2.4 Viabilidade de filtragem de melhores pontos

A viabilidade de filtragem dos melhores pontos serve ao princípio de não sobrecarregar o projeto com pontos. Logo, é importante que parâmetros adicionais possam ser adotados na solução final para efetuar controle do total de pontos gerados.

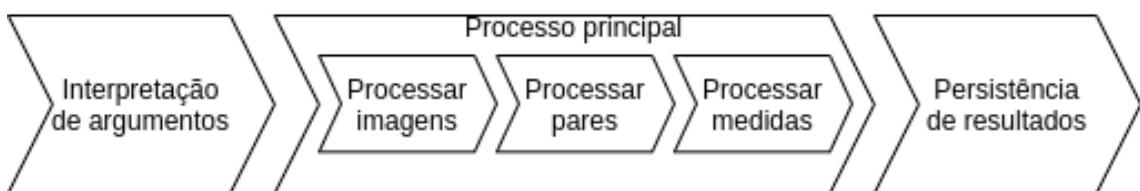
### 2.3 Planejamento de testes comparativos

Para verificar a viabilidade de execução de todos os algoritmos selecionados foram localizadas bases comuns de código, principalmente entre os exemplos do módulo *features2d*<sup>17</sup>. Os exemplos para estudo deste módulo incluem a produção e uso de detectores, bem como a apresentação de diferentes extratores de feições, métodos de correlação e a verificação geométrica, incluindo a aplicação na localização de objetos em imagens e de rastreamento em vídeo. Foram então planejados e efetuados testes de configuração com os diferentes algoritmos em estudo adotando-se um mesmo fluxo de base. Foram incorporadas e testadas diferentes técnicas de tomada de medidas para extrair os valores que atendessem aos critérios definidos. Ao fim deste planejamento, foi possível esboçar um conjunto de variações nas entradas que pudessem ser usadas para testar o produto final.

### 2.4 Codificação da solução

Tendo feito modificações e testes de uso de diversos exemplos de código da biblioteca, iniciou-se a fase de codificação de um produto final (disponível no Apêndice A) integrando aspectos estudados para atender o objetivo deste projeto. Por simplificação foi adotada a interface de linha de comandos, entendendo que esta poderá ser acessada diretamente por outro software, como o e-foto, para obter os pontos de costura dado um conjunto de argumentos variado.

Figura 8 - Fases da execução da solução de linha de comando proposta



Fonte: O autor

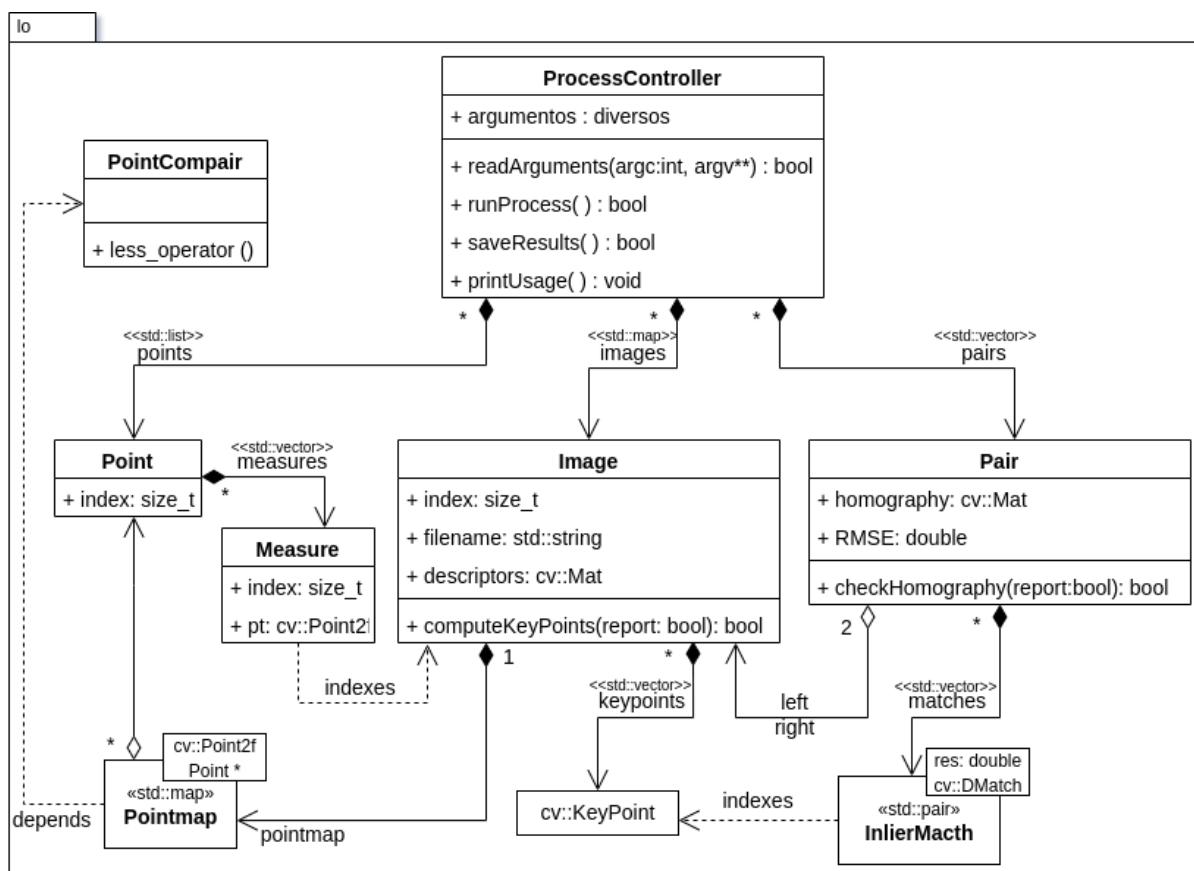
De um ponto de vista geral, as partes do processo podem ser descritas como indica a Figura 8, onde é observado que além do processamento principal é a necessidade de ao menos duas fases dedicadas à interface com o mundo externo. A interpretação dos argumentos lida com informações de entrada que configuram o pedido de processamento. A persistência de resultados lida com a saída das respostas encontradas durante o processo principal. Ao seu tempo, o processo principal pode ser subdividido em 3 outros focados em tratamento das

<sup>17</sup> Materiais disponíveis em <[https://docs.opencv.org/4.5.4/d9/d97/tutorial\\_table\\_of\\_content\\_features2d.html](https://docs.opencv.org/4.5.4/d9/d97/tutorial_table_of_content_features2d.html)>

imagens individualmente, dos pares sugeridos pelo usuário e das medidas identificadas pelo processamento de pares.

Por sua vez, as tentativas de acomodar a solução com programação procedural apenas, como transcorre na maioria dos exemplos de OpenCV, mostrou-se de grande complexidade para a documentação, pois muitas em alguns pontos o código tornava-se de difícil leitura. Por exemplo, assumindo que uma função *knnMatch* pode retornar os  $n$  vizinhos mais próximos de cada um dos  $m$  pontos chaves descritos, e que este processo seria repetido  $p$  vezes para atender aos pares processados, teríamos um vetor com 3 dimensões causando uma complexa administração de índices ao acessar tais informações, ou seja, algo como *knnResult[i][j][k]* seria necessário para indicar o  $k$ -ésimo vizinho do  $j$ -ésimo ponto chave da primeira imagem do  $i$ -ésimo par de imagens.

Figura 9 - Diagrama de classe da solução



Fonte: O autor

Pela adoção de um modelo de programação orientado à objetos podemos ainda expressar a estrutura da solução proposta em termos de linguagem gráfica na notação UML, conforme ilustra a Figura 9. No diagrama de classes um controlador de processo define métodos para as principais atividades, além de um método de impressão dos argumentos para uso da solução. O controlador

de processo acomoda todas as imagens carregadas através do arquivo de entrada, instancia os pares cuidando das ligações necessárias com suas imagens (evitando assim a repetição das informações de imagens em memória) e cria, atualiza ou junta pontos durante o processamento de medidas (haja visto que pontos de costura precisam ser medidos em diferentes imagens). No processamento de pares e imagens, o controlador repete a chamada de métodos apropriados de cada imagem a ser descrita ou par a ser correlacionado.

Objetos de OpenCV que configuram pontos chaves e correlações, bem como matrizes de descrição ou transformações geométricas, são apresentados no diagrama de classe e cabe notar que a OpenCV usa índices numéricos (ligações indiretas) que apontam estes objetos pela posição em que se encontram num *container* de dados, tipicamente vetores da biblioteca padrão de C++, a STL (acrônimo para *Standard Template Library* ou biblioteca de modelos padrão).

A ligação entre as classes imagem e ponto faz-se importante em dois momentos e nos dois sentidos, sendo caracterizada a ligação com o ponto, isto é, partindo da imagem, como prioritária. Isto se justifica pelo algoritmo de integração da medidas dos pontos de costura quando estes ocorrem em pares distintos. Nos casos onde uma imagem participa de diversos pares é necessária a busca na imagem pela medida, a fim de determinar se esta já foi registrada durante o processamento de outros pares com a mesma imagem. Ao seu tempo, a ligação com a imagem, partindo do ponto, faz-se necessária em termos de entrega do produto final, na qual cada medida do ponto em distintas imagens deve ser corretamente associada. A associação das medidas de cada ponto podem então ser atendidas com uma simples cópia do índice da imagem, sem que isso implique em perda de performance ou grande consumo de memória.

#### 2.4.1 Escolha de argumentos

Para possibilitar alguma flexibilidade de uso, a solução final requer que sejam apontados nomes de arquivos de entrada e sugeridos outros dois nomes para os arquivos de saída. Além destes arquivos, é possível adicionar opções de configuração que podem, entre outras coisas, trocar o método de detecção e descrição em uso, alternar entre os métodos de correlação disponíveis e inserir critérios para limitar o conjunto de resultados obtidos.

O primeiro dos arquivos de entrada esperados é uma lista de caminhos para as imagens que serão processadas, precedidas de seus índices no projeto, pois o resultado será apresentado em termos destes índices de imagens. O segundo arquivo deve indicar quais pares de imagens devem ser processados. Neste segundo arquivo, apenas os índices de imagens precisam ser apresentados.

Alternativamente, uma opção *mode* pode ser usada para escolher processar todos os pares possíveis (*ALL*) ou através da sequência (*SEQUENCE*), na qual as imagens foram descritas. Deste modo, se *SEQUENCE* está ativo e as imagens listadas são 16, 17 e 18, por exemplo, os pares serão 16 com 17 e 17 com 18.

Já os nomes dos arquivos de saída devem ser indicados para que sejam gravados, no primeiro arquivo, as medidas nas imagens para cada ponto de costura, e no segundo arquivo, o índice e tipo de cada ponto. Neste segundo arquivo são gravados então o tipo *Photogrammetric*, que indicam para o e-foto que estes podem ser usados na aerotriangulação, e uma sequência com 6 zeros (valores para E, N, H e suas precisões destas 3 medidas, quando disponíveis). A gravação com zeros é feita para que o e-foto não precise de quaisquer alterações evitando assim erros de leitura. Cabe notar que no processo de aerotriangulação as coordenadas de mundo destes pontos serão sobrescritas.

Na versão 472<sup>18</sup> do código do e-foto, observou-se que o método *importPointsFromTxt2* pode ser acessado somente pelo atalho de teclado (“Ctrl+Shift+P”). Este está disponível durante a execução do módulo de projetos e resulta no pedido de dois arquivos a saber: arquivo de coordenadas ENH (nossa segunda saída); e o arquivo de medidas no espaço das imagens.

Argumentos extras para o processamento incluem:

- **detector** para selecionar o algoritmo adotado (AKAZE, ORB, SIFT ou SURF);
- **ccheck** para acionar a verificação cruzada de correlações com BFM (desligando o FLANN);
- **verbose** para ligar a impressão de informações do processamento úteis na confecção dos resultados apresentados no Capítulo 3;
- **number\_f** para definir o número  $f$  de pontos chaves o algoritmo ORB irá detectar;
- **force\_f** para ativar a limitação de detecções nos demais algoritmos baseado em  $f$ ;
- **gruber\_roi** para ativar a limitação da área passível a detecção de pontos chave;
- **help** para exibir os parâmetros e suas descrições;
- **init\_index** para definir o identificador inicial dos pontos de costura;
- **list\_pairs** para registrar os pares que possuem solução com uma taxa de *inliers* desejada;
- **limit\_out** para limitar a saída de pontos de costura por par;
- **mode** para definição do modo em que serão selecionados os pares;
- **n\_measures** para limitar os pontos àqueles contidos em um mínimo de  $n$  imagens;
- **residue** para definir o valor máximo do resíduo aceito na verificação geométrica;
- **scale** para definir o denominador para realizar a redução das imagens;

---

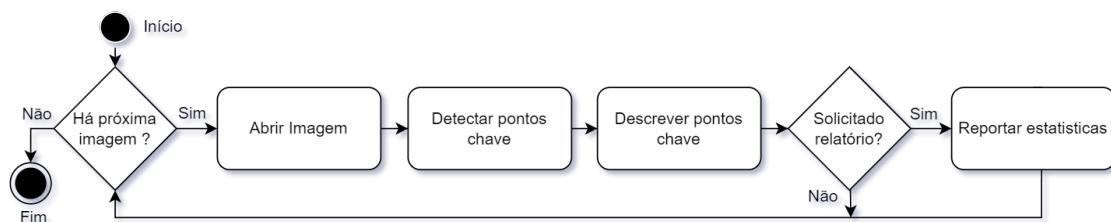
<sup>18</sup> Método disponível em <[https://sourceforge.net/p/e-foto/code/472/tree/trunk/qt/interface/ProjectUserInterface\\_Qt.cpp#l2635](https://sourceforge.net/p/e-foto/code/472/tree/trunk/qt/interface/ProjectUserInterface_Qt.cpp#l2635)>

A impressão de toda a lista de argumentos foi programada para atender a opção *help*, ou no caso de quaisquer falhas de execução do processo que inviabilizem a gravação de resultados. Erros graves que podem suspender o processamento geram mensagens ao usuário, como a inviabilidade de abrir uma das imagens indicadas pelo usuário, enquanto que falhas toleráveis, como a inviabilidade de obter medidas para um par, são contornadas e podem ser reportadas ao final do processo.

#### 2.4.2 Processamento de imagens

No processamento de imagens se cumprem as extrações dos pontos chaves do conjunto de imagens. Há método em OpenCV para fazê-lo para um conjunto de imagens pré-carregadas pelo usuário, contudo entende-se que se o tamanho de todas as imagens for muito grande pode ser inviável usar este recurso. Logo, como indicado na Figura 10, adotou-se uma sequência de atividades onde cada imagem é aberta, tem suas informações extraídas e então fechada, tendo o número de pontos-chave relatado se solicitado.

Figura 10 - Diagrama de atividades para o processamento de imagens



Fonte: O autor

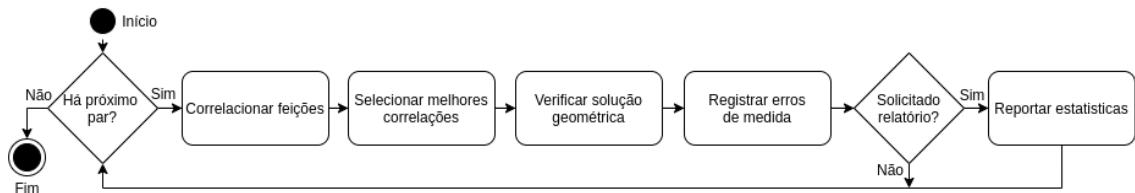
De igual modo, os exemplos disponibilizados pela OpenCV demonstram a detecção e descrição simultaneamente, porém optamos por separar essas atividades para que uma melhor compreensão dos algoritmos possa ser alcançada. Isto ajuda na quantificação de tempos individuais para estas etapas e poderia ser usado para mesclar os métodos ou adotar algoritmos especializados em uma das duas atividades. Medidas de tamanho das estruturas podem demonstrar como o processo escala em memória e ajudam a compreender qual a quantidade de memória que pode vir a ser necessária para grandes conjuntos de imagens com cada método.

#### 2.4.3 Processamento de pares

O processamento de pares faz correlação de pontos chaves descritos previamente para as imagens e, em seguida, verifica se é possível obter uma solução de transformação geométrica

entre o conjunto de pares, considerando um erro máximo de reprojeção aceitável. Estas atividades são implementadas integralmente pela OpenCV, mas é importante realizarmos atividades extras, como as ilustradas na Figura 11, para selecionar melhores correlações e registrar os erros de medida observados. Estes erros são possíveis de computar usando a transformação geométrica, que restringe os pontos da solução final.

Figura 11 - Diagrama de atividades para o processamento de pares



Fonte: O autor

A correlação dos pontos chaves pode ser feita por força bruta (BFMatcher na OpenCV), isto é, cada ponto chave em uma imagem é confrontado com todos os pontos chaves da outra imagem no par. Deste modo, são preservadas as menores distâncias no espaço de descrição observadas para cada ponto chave processado. Embora caro computacionalmente, ficando seu uso restrito aos pequenos conjuntos de dados, este processo permite a verificação cruzada de vizinhos mais próximos. Isto equivale a responder a questão “se um ponto chave  $P$ , na primeira imagem, possui melhor correlação com um ponto chave  $Q$ , na segunda imagem, a análise no sentido oposto também é verdadeira?”, onde o “afirmativo” corresponde a correlação cruzada.

A alternativa viável ao processamento de correlações por força bruta, quando o conjunto de pontos chaves é muito grande, é a busca por vizinhos mais próximos aproximados (Flann-BasedMatcher na OpenCV). Ela usa estruturas especializadas para a busca espacial, como as árvores de k-dimensões (KD-Tree) ou o hash sensível à localidade (LSH), implementadas e amplamente divulgadas pela biblioteca flann<sup>19</sup>. Neste caso, a estrutura de busca espacial deve estar alinhada ao descritor usado, ou seja, descritores binários requerem algoritmos de hash, enquanto descritores modelados com números reais, como SIFT e SURF, usam KD-Tree. Esta diferença é notável na própria forma de computar distâncias, pois a distância aplicável aos descritores binários é a distância de Hamming (RUBLEE et al., 2011).

O ganho em tempo de execução quando utilizados vizinhos mais próximos aproximados é contraposto pela inviabilidade de verificação cruzada. Contudo, um outro critério de eliminação pode ser adotado ao observar a razão entre a distância para o vizinho mais próximo de um ponto chave e o segundo vizinho mais próximo. A lógica embarcada nesta observação é que um ponto

<sup>19</sup> Fast Library for Approximate Nearest Neighbors está disponível em <https://github.com/flann-lib/flann>

chave com apenas um vizinho mais próximo é potencialmente mais relevante do que aqueles que possuem mais de um vizinho em seu entorno (LOWE, 2004).

É importante notar que apenas selecionar as melhores correlações pode não ser suficiente para eliminar erros grosseiros. Logo, uma segunda rodada de análise, considerando as restrições geométricas que podem ser observadas para imagens de uma mesma cena, deve ser aplicada. Nesta atividade, um outro módulo de OpenCV, *calib3d*, apresenta contribuições importantes, das quais adotamos o método *findHomography*. Sua aplicação principal é o cálculo de uma transformação projetiva entre os planos das imagens dadas por um conjunto de pontos correlatos. Aqui aplicam-se as técnicas de ajustamento robustas discutidas anteriormente e, como consequência, podem ser identificados e removidos quaisquer pares de pontos chaves que apresentem erro de reprojeção superior a algum limiar definido pelo usuário.

De posse da transformação geométrica podemos ainda registrar e classificar as correlações restantes segundo o resíduo observado. Entendemos que este pode ser um critério melhor do que a distância de correlação propriamente quando se trata da seleção dos melhores pontos de costura para um bloco de imagens.

#### 2.4.4 Processamento de medidas

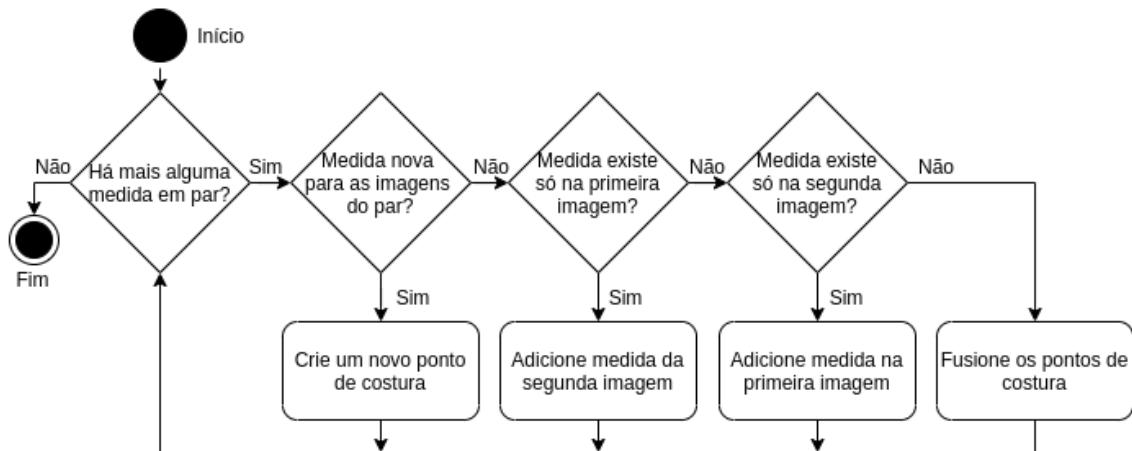
Com os dados obtidos após o processamento de pares, resta a organização das medidas a serem salvas como resultado do processo. Isto é feito pelo próprio controlador de processo, tendo em vista que ele é o detentor de todas as informações e que os pontos de costura individualmente não devem realizar qualquer processamento, além do registro de suas medidas, evitando indexar a mesma imagem mais de uma vez.

Embora seja viável na OpenCV correlacionar feições de uma imagem com diversas outras em apenas uma chamada, algo que se aplica muito bem para aplicações de detecção de objetos, esta abordagem parece inapropriada para a finalidade de costura de blocos fotogramétricos, pois tornar-se-ia complexo distinguir entre padrões repetitivos de bons pontos de costura observáveis em mais de duas imagens. Porém, para o e-foto e para a fotogrametria em geral, uma faixa de aerofotos pode ter inúmeras imagens, todas com sobreposição regular e muitas faixas, onde a sobreposição é menor, mas pode haver boas feições. Logo, é desejável a percepção de pontos em 3 ou mais imagens, mesmo se considerarmos apenas uma faixa com recobrimento lateral de aproximadamente 60%, que é o caso da maioria das faixas de sobrevoo convencionais. Para mais, aerolevantamentos com drones podem ter ainda mais sobreposição para compensar os pequenos formatos de sensores e a falta de estabilidade destas plataformas.

Logo, julgamos que não cabe registrar todos os pares de pontos correlatos diretamente sem antes efetuar qualquer crítica sobre a possibilidade de um mesmo ponto chave ser bem correlacionado, e com pouco resíduo, em diferentes pares de imagens. O que se espera obter é a observação de uma propriedade transitiva que afirma, “Se  $a = b$  e  $b = c$ , então  $a = c$ ”, ou

seja, pela correlação ter sido realizada em imagens par-a-par, um mesmo ponto pode estar em dois ou mais pares e ter algumas de suas coordenadas ligadas a duas outras imagens ou mais. Uma extensão da mesma propriedade pode ser importante para lidar com o encadeamento não sequencial de pares, isto é, “Se  $a = b$ ,  $c = d$  e posteriormente é constatado que  $b = c$ , então  $a = c$ ,  $a = d$  e  $b = d$ ”.

Figura 12 - Diagrama de atividades para o processamento de medidas



Fonte: O autor

Com o entendimento destas possibilidades de combinação de pares podemos formular um diagrama de atividades que incorpore a criação, atualização e fusão de pontos chaves, como o ilustrado na Figura 12. Então para todos os *inliers* de cada par repete-se o processo de registro das medidas observadas. A criação de um novo ponto de costura ocorre sempre que nenhuma das imagens do par registrou previamente a medida observada. Se apenas uma delas registrou o previamente a medida observada, então existe um ponto de costura a ser atualizado pela adição de mais uma medida. Contudo, se ambas as imagens de um par já registraram as medidas observadas, pode haver dois pontos de costura distintos que se aplicam a uma mesma feição do espaço objeto. Neste caso, as medidas devem ser mescladas num único ponto de costura.

Para a busca na imagem por medida pré-existente adotou-se a estrutura de dados *map*<sup>20</sup> da STL, pois esta possui tempo de acesso reduzido, aplicando internamente o algoritmo de busca binária. Outras alternativas seriam adotar uma *KD-tree*, com tempo de acesso equivalente ao *map*, ou um algoritmo de *hash* (como implementado em *unordered\_map*) com tempo de resposta menor, mas possivelmente com elevado custo de armazenamento.

<sup>20</sup> Ajustes necessários para uso desta estrutura com os pontos do OpenCV foram indicados em <https://stackoverflow.com/questions/26483306/stdmap-with-cvpoint-as-key>

#### 2.4.5 Persistência de dados

A persistência de dados será realizada sob os formatos de arquivos adotados pelo método de inserção pré-existente no e-foto. O primeiro arquivo solicitado pelo e-foto identifica o ponto, atribui um tipo e apresenta os valores para E, N e H e as precisões deste no espaço objeto. Assim, para cada ponto de costura gerado teremos uma linha, como pode ser observado na Figura 13, onde o primeiro valor equivale à identificação do ponto seguido pelo tipo de ponto. Os demais valores são zerados, pois não se aplicam aos pontos fotogramétricos.

Figura 13 - Exemplo do arquivo de persistência de pontos.

462	Photogrammetric	0	0	0	0	0
463	Photogrammetric	0	0	0	0	0
464	Photogrammetric	0	0	0	0	0
465	Photogrammetric	0	0	0	0	0
466	Photogrammetric	0	0	0	0	0
467	Photogrammetric	0	0	0	0	0
468	Photogrammetric	0	0	0	0	0
469	Photogrammetric	0	0	0	0	0
470	Photogrammetric	0	0	0	0	0

Legenda: Da esquerda para a direita as colunas representam: identificação de cada ponto; tipo de ponto; E; N; H; e suas respectivas precisões.

Fonte: O autor.

Figura 14 - Exemplo do arquivo de persistência de medidas.

1	462	1930.18	1496.45			
2	462	943.488	1477.44			
1	463	1847.23	1686.53			
2	463	857.088	1669.25			
1	464	1959.55	1689.98			
2	464	971.136	1670.98			
1	465	1983.74	1387.58			
2	465	997.056	1368.58			
1	466	1714.18	1584.58			
2	466	724.032	1569.02			

1	467	2329.34	1147.39			
2	467	1349.91	1121.82			
1	468	2355.26	1145.66			
2	468	1373.76	1119.74			
3	468	328.32	1197.5			
1	469	2500.42	1479.17			
2	469	1520.64	1451.52			
1	470	1639.87	1907.71			
2	470	648	1892.16			

(a)

(b)

Legenda: Da esquerda para a direita as colunas representam: índice na imagem; índice do ponto; coluna; e linha de medição do ponto na imagem.

- (a) Medidas dos pontos de índice 462 à 466; e
- (b) Medidas dos pontos de índice 467 à 470.

Fonte: O autor.

Por sua vez, a Figura 14 ilustra o conteúdo do segundo arquivo solicitado, no mesmo processo de importação de pontos fotogramétricos pelo e-foto. Este arquivo leva as medidas

dos pontos em coordenadas digitais (no espaço imagem) a serem inseridas. Quatro colunas são usadas para: a identificação da imagem na qual o ponto foi medido; indexar do ponto medido; e transmitir as coordenadas em coluna e linha, respectivamente.

Cabe destacar que a carga de pontos fotogramétricos no e-foto poderia ainda ser modificada para eliminar a necessidade do arquivo que identifica o ponto, pois este não agrupa qualquer informação que não possa ser derivada do segundo formato de arquivo solicitado. Por este motivo o fornecimento de nome para arquivo que identifica os pontos é considerado opcional na solução proposta. Restando o nome do arquivo de medidas nas imagens como argumento obrigatório na interface de linha de comando definida. Este mesmo arquivo é preenchido de forma distinta apenas quando a opção *list\_pairs* está ativa, de modo que podem ser gravados os índices de cada par que possuem solução com uma taxa de *inliers* estipulada pelo usuário.

Como a exibição dos pontos importados no e-foto se dá através de listas de pontos ou pela plotagem sobre o par de imagens na execução da fototriangulação pode ser trabalhoso conferir pontos que possuem medidas em 3 ou mais imagens. Por este motivo, recomenda-se a criação futura de um módulo para exibir e manipular o foto-índice do projeto. Tal módulo vindouro poderia adicionar recursos para melhor gerenciamento de grandes volumes de dados permitindo, por exemplo, a ocultação de pontos que atendam a algum critério de filtragem.

### 3 RESULTADOS

Neste capítulo serão apresentados alguns dos resultados obtidos para os diversos testes realizados durante a execução desse projeto. Inicialmente serão apresentados valores numéricos e avaliadas as diferenças entre algoritmos adotados, na sequência serão ilustrados os resultados sobre pares de imagens para a conferência visual das correlações e finalmente serão consideradas as variações que podem ser obtidas pela manipulação dos argumentos admitidos na solução construída.

#### 3.1 Resultados numéricos dos testes

Após codificação do produto final e execução dos testes planejados temos resultados divididos entre subprocessos de imagens, que realiza detecção e descrição de feições; de pares de imagens, que implicam nas correlações e verificações geométricas; e de medidas, que implicam em sumarizar os pontos de costura obtidos.

##### 3.1.1 Custo de detecção e descrição

A apresentação dos resultados foi realizada através da Tabela 4 por imagem e, para maior clareza, foram agrupados para cada algoritmo seus tempos de detecção de feições, número de respostas obtidas, uma razão entre esses valores e o espaço de memória alocado em MB (*Mega Byte*) para a resposta. Também foram integrados dados de tempo da descrição de feições e uma razão entre este e o número de feições, além do espaço necessário para manter a descrição disponível para as etapas seguintes no fluxo da aplicação. A última coluna da mesma tabela apresenta médias dos valores anteriores de cada imagem para auxílio na análise.

Ao realizar a análise da mesma tabela, pode-se verificar que o ORB tem seu número de feições regularizado, o que se deve a uma configuração do próprio método que requer o número de feições a serem retidas. A quantidade de feições predefinida pela OpenCV é de 500 feições, o valor apresentado (100000) foi escolhido de acordo com uma aproximação dos valores observados quando realizados os testes com os demais algoritmos. Uma vez que AKAZE, SIFT e SURF não necessitam de tal definição, pois retornam a quantidade de feições disponíveis, é observada uma flutuação considerável nos número de feições extraídas e consequentemente do espaço alocado.

Ao comparar o tempo gasto nos diversos métodos utilizados para detecção e descrição é possível observar que o tempo médio por ponto durante a detecção e a descrição é estável na maioria dos casos quando analisado dentro do mesmo método. O método SURF foge dessa regra

Tabela 4 - Resultados numéricos de detecção e descrição dos métodos analisados.

Método	Etapa	Medida	Imagens			Média
			16	17	18	
AKaze	Detecção	Número de pontos	81652	78797	73851	78100
		Tempo total (ms)	1208,4	974,291	899,603	1027,431
		Tempo / Ponto	0,015	0,012	0,012	0,013
	Descrição	Tamanho alocado(MB)	2,170	2,100	1,970	2,080
		Tempo total (ms)	1138,68	1029,09	967,404	1045,058
		Tempo / Ponto	0,014	0,013	0,013	0,013
ORB	Detecção	Tamanho alocado(MB)	19,000	18,330	17,180	18,170
		Número de pontos	100000	100000	100000	100000
		Tempo total (ms)	302,914	261,936	267,354	277,401
	Descrição	Tempo / Ponto	0,003	0,003	0,003	0,003
		Tamanho alocado(MB)	2,660	2,660	2,660	2,660
		Tempo total (ms)	192,903	203,595	190,662	195,72
SIFT	Detecção	Tempo / Ponto	0,0019	0,0020	0,0019	0,002
		Tamanho alocado(MB)	12,2	12,2	12,2	12,2
		Número de pontos	114684	106728	103486	108299
	Descrição	Tempo total (ms)	2252,18	1823,36	1589,33	1888,29
		Tempo / Ponto	0,02	0,017	0,015	0,017
		Tamanho alocado(MB)	3,06	2,84	2,76	2,887
SURF	Detecção	Tempo total (ms)	2496,85	2382,41	2244,64	2374,633
		Tempo / Ponto	0,022	0,022	0,022	0,022
		Tamanho alocado(MB)	55,99	52,11	50,53	52,877
	Descrição	Número de pontos	91048	90826	104511	95462
		Tempo total (ms)	1283,61	1197,88	1245,89	1242,46
		Tempo / Ponto	0,014	0,013	0,012	0,013
SURF	Detecção	Tamanho alocado(MB)	2,43	2,42	2,790	2,547
		Tempo total (ms)	4292,260	4257,53	4421,740	4323,843
		Tempo / Ponto	0,047	0,047	0,042	0,045
	Descrição	Tamanho alocado(MB)	22,22	22,17	25,510	23,3

Fonte: ‘O autor’.

possivelmente por um de seus parâmetros específicos desligado por padrão, o *upright*<sup>21</sup>, que ocasiona levar em conta a orientação de cada ponto aumentando assim o tempo de descrição total. Ressalta-se que a orientação dos descritores é desnecessária para blocos de imagens bem comportadas e previamente alinhadas como geralmente são as imagens aéreas utilizadas no e-foto, sendo sobretudo mais adequadas às imagens obtidas em plataformas leves e com pouca estabilização como ocorrem em imageamento com drones.

Cabe observar ainda que uma maior capacidade de descrição parece estar relacionada a uma maior possibilidade de obter melhores pontos de costura, o que pode ser visualizado ao se comparar o tamanho alocado para os descritores de cada método com seu respectivo RMSE conforme ilustrado na próxima seção (3.1.2).

### 3.1.2 Custo de correlação e Verificação geométrica

A apresentação dos resultados do processamento de pares está disponível através da Tabela 5 que, para maior clareza, exibe para cada tipo de descritor as observações feitas na fase de correlação. Adotou-se correlação baseada em estruturas de dados espaciais (usando FLANN) e verificação geométrica com consenso randômico (RANSAC). Para cada um desses há tempos de execução, quantidade de correlatos reportados e o tamanho das respostas. Vale ressaltar que a entrada de dados para a correlação é a soma da quantidade de pontos de cada imagem a ser analisada, e os dados de entrada da verificação geométrica são a quantidade de bons pares que a correlação retorna.

Ao comparar a quantidade de pontos que a detecção gerou com a quantidade de bons pares é possível perceber uma redução de pontos. Isso ocorre por diversos fatores como a ocorrência de pontos chaves sobre padrões repetitivos nas imagens, a sobreposição parcial das imagens e a oclusão de pontos devido ao relevo em alguma das imagens do par. Ao realizar a verificação geométrica essa quantidade é reduzida ainda mais e essa redução é incentivada pelo rigor das avaliações de dados que buscam minimizar a possibilidade de persistirem erros grosseiros na solução.

A raiz quadrada do erro médio (RMSE), exibida pela solução, deve-se ao ajustamento da solução geométrica de cada par. Isto é medido no espaço das imagens, ou seja, é dado em *pixels* devido aos resíduos de transformação dos pontos (*inliers*) que foram considerados aptos pela solução. A solução nestes resultados foi parametrizada para o erro máximo abaixo de 2.0 *pixels*, internamente os *inliers* nos pares estão ordenados no sentido crescente pelo resíduo observado e o corte do conjunto pode resultar em um RMSE final ainda menor. Essa arrumação de valores foi

---

<sup>21</sup> Maiores informações sobre este e demais parâmetros podem ser encontrados em <[https://docs.opencv.org/4.5.4/d5/df7/classcv\\_1\\_1xfeatures2d\\_1\\_1SURF.html](https://docs.opencv.org/4.5.4/d5/df7/classcv_1_1xfeatures2d_1_1SURF.html)>

Tabela 5 - Resultados numéricos da correlação e verificação geométrica.

Método	Etapa	Medida	Pares		Média
			16 - 17	17 - 18	
AKaze	Correlação	Bons pares	7226	6523	6874,5
		Tempo total (ms)	3103,35	2203,19	2653,27
		Tamanho alocado(MB)	2,48	2,39	2,435
	Verificação geométrica	<i>Inliers</i> nos pares	1867	1352	1609,5
		Tempo total (ms)	23,7245	38,4665	31,0955
		Tamanho alocado(KB)	43,75	31,68	37,715
ORB	Correlação	RMSE	1,12	1,11	1,115
		Bons pares	6110	5362	5736
		Tempo total (ms)	2262,75	2157,68	2210,215
	Verificação geométrica	Tamanho alocado(MB)	3,05	3,05	3,05
		<i>Inliers</i> nos pares	737	260	498,5
		Tempo total (ms)	36,666	35,2232	35,9446
SIFT	Correlação	Tamanho alocado(KB)	17,27	6,09	11,68
		RMSE	1,21778	1,19922	1,2085
		Bons pares	8058	6161	7109,50
	Verificação geométrica	Tempo total (ms)	3125,04	2863,43	2994,24
		Tamanho alocado(MB)	3,49	3,25	3,37
		<i>Inliers</i> nos pares	1984	1607	1795,5
SURF	Correlação	Tempo total (ms)	29,3262	22,4838	25,91
		Tamanho alocado(KB)	31	25,1	28,05
		RMSE	0,948789	0,93715	0,94
	Verificação geométrica	Bons pares	7454	6432	6943
		Tempo total (ms)	2207,38	2295,27	2251,33
		Tamanho alocado(MB)	2,77	2,77	2,77
	<i>Inliers</i> nos pares	<i>Inliers</i> nos pares	1963	1460	1711,5
		Tempo total (ms)	23,3946	39,1725	31,28
		Tamanho alocado(KB)	30,67	22,81	26,74
		RMSE	1,01174	1,04918	1,03

Legenda: Bons pares: são as melhores correlações que o método FLANN, detalhado no capítulo 2, encontrou para os pontos chaves descritos; *Inliers* nos pares: são as correlações de pontos com resíduos aceitáveis e serão consideradas ao registrar as medidas de pontos de costura.

Fonte: ‘O autor’.

escolhida pela importância da obtenção dos melhores pontos de costura, ou seja, pontos muito bem ajustados para evitar pertubações no processo fotogramétrico considerando que este já pode ter de lidar com algum erro inserido por uma escolha manual dos pontos de controle.

### 3.1.3 Taxa de obtenção dos mesmos pontos em diferentes pares

Na Tabela 6 é exibido, para todos os algoritmos comparados, os *inliers* nos pares analisados e sua soma, a quantidade de pontos de costura que a solução retorna quando não é aplicado nenhum argumento de redução para o número de medidas por par a serem gravadas e a quantidade de pontos de costura oriundos de diferentes pares, ou seja, que situam-se sobre 3 imagens. Outra forma de limitar a resposta com argumentos, útil para blocos com muita sobreposição, é a possibilidade de filtrar pontos que não atravessem um número mínimo de imagens, logo, é de interesse comum entender qual é a taxa de obtenção destes pontos quando comparados com os volumes totais obtidos.

Tabela 6 - Comparação entre o número de *inliers* nos pares e o total de pontos de costura

Método	<i>Inliers</i> no par		Soma	Pontos de costura	Diferença	Pontos nas 3 imagens	Pontos mesclados	Taxa
	16x17	17x18						
Akaze	1867	1352	3219	3100	119	119	0	3.8%
ORB	737	260	997	960	37	37	0	3.9%
SIFT	1984	1607	3591	3092	499	236	263	7.6%
SURF	1963	1460	3423	3088	335	333	2	10.7%

Legenda: Soma: totaliza *inliers* nos pares;

Diferença: entre pontos de costura e a soma de *inliers* nos pares;

Taxa: é a razão entre pontos nas 3 imagens e pontos de costura.

Fonte: O autor

Na tabela, a taxa de obtenção dos mesmos pontos em diferentes pares é consideravelmente menor nos métodos descritores binários, como o Akaze e o ORB, quando comparada aos descritores baseados em histograma de gradientes, como o SIFT e o SURF. Contudo, estas taxas parecem muito baixas em todos os casos e é necessário que próximos estudos determinem se configurações específicas podem aumentar estas taxas sem perda da qualidade. Destaca-se ainda que a diferença entre a soma dos *inliers* nos pares e os pontos de costura normalmente se deve à quantidade de pontos que foram medidos em 3 ou mais imagens, porém, como pode ser observado com SIFT e SURF, há ainda a possibilidade de discrepância na diferença. Isto se deve à ativação do processo de mesclagem, mas vale observar que o uso deste processo era previsto em casos de análise de mais de 2 pares de imagens. A aparição neste ensaio revela um desdobramento não previsto a ser tratado em atualizações futuras.

Com o entendimento atual, determinou-se a hipótese de que essas mesclas acontecem quando o método encontra boas feições no mesmo *pixel* porém em escalas diferentes. Estas feições foram correlacionadas com descritores bastante distintos, pois são máximos ou mínimos locais no espaço de escala, e correlacionaram entre si nas distintas escalas, assim se tornando pontos duplicados, quando analisados nas circunstâncias de espaço da imagem, e por isto foram mesclados. Vale ressaltar que tais pontos não introduzem maior erro à amostra, o que pode ser confirmado ao se perceber que o método com maior quantidade de pontos mesclados não apresenta degradação no RMSE, sendo ainda assim o menor RMSE registrado.

### 3.2 Análise visual dos resultados pela carga no e-foto

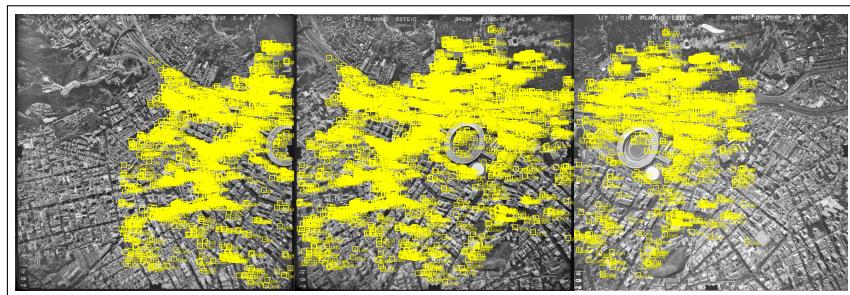
Para fim de análise da dispersão espacial dos pontos de costura e de sua carga no e-foto foi elaborada a Figura 15, com imagens retiradas do software e-foto após carga dos dados resultantes da solução apresentada por esse trabalho. A produção destes conjuntos de dados utilizou os parâmetros bases estabelecidos por esse texto, a única alteração necessária para sua composição sua foi a escolha dos diferentes métodos abordados para viabilizar a comparação. Deste modo, o volume observado equivale aos quantitativos numéricos de pontos de costura exibidos anteriormente neste capítulo.

A Figura 15a exibe os resultados do método AKAZE e a distribuição de 3100 pontos de costura dos quais 119 atravessam as 3 imagens, já a Figura 15b exibe os resultados do método ORB, com volume consideravelmente inferior, sendo 960 pontos de costura dos quais 37 estão nas 3 imagens, mas mantendo a distribuição similar quando comparada aos demais métodos. Ao seu tempo a Figura 15c exibe os pontos resultantes do método SIFT que são, 3092 pontos de costura dos quais 236 existem nas 3 imagens. Outros 263 pontos obtidos com este método que estavam duplicados foram mesclados e não encontram-se apresentados. A Figura 15d exibe os resultados do método SURF, na qual existem 3088 pontos de costura, dos quais 2 dois foram suprimidos pela mesclagem, e 333 são correlatos entre as 3 imagens.

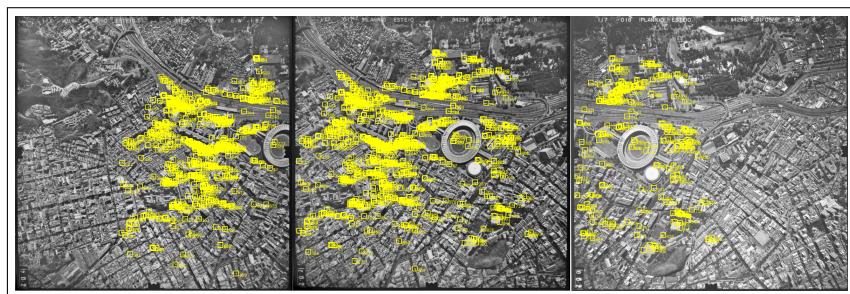
Como podemos analisar nas figuras os dados são bem distribuídos apesar de termos maior quantidade dos pontos de costura nos locais mais centrais das imagens o que possivelmente ocorre por conta da limitação imposta pelo resíduo máximo escolhido. Cabe ressaltar que o cálculo de verificação geométrica é modelado como uma transformação entre planos. Áreas com grande variação de altitude acabam sendo julgadas como pontos com maior resíduo. Por conseguinte é possível observar que a área que mais recebeu massa de pontos foi a superior ao maracanã que engloba a linha de trem por terem baixa variação de altitude.

Para viabilizar a verificação da qualidade dos pontos de costura foram feitos recortes de tamanho regular na vizinhança das medidas nas imagens onde foram computados pela solução. Nos recortes o ponto medido corresponde ao centro e sua marcação foi omitida para não interferir na identificação dos pares. Ao todo, 20 pontos de costura em cada par foram isolados e

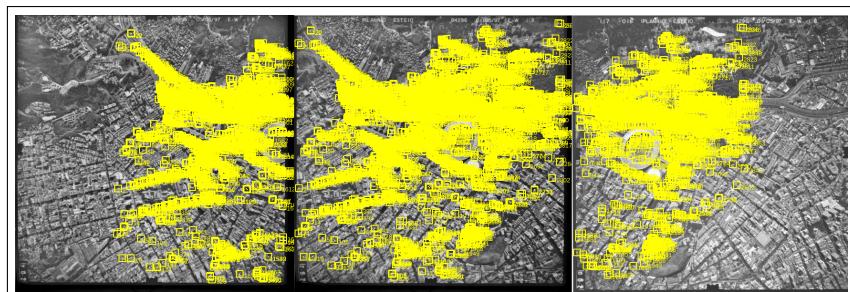
Figura 15 - Volume e distribuição dos resultados obtidos



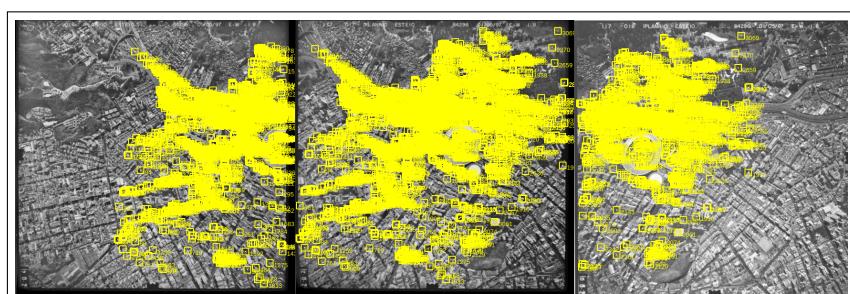
(a)



(b)



(c)



(d)

Legenda: Resultados exibidos após inserção dos pontos no arquivo

UERJ\_IO.epp, disponibilizado pelo Projeto E–Foto, que  
utiliza as imagens 16, 17 e 18 com: (a) Resultado obtido com  
o método AKAZE; (b) Resultado obtido com o método ORB;  
(c) Resultado obtido com o método SIFT; e (d) Resultado  
obtido com o método SURF.

Fonte: O autor.

encontram-se apresentados na Figura 16. Em nenhum dos pares foi observado erro grosseiro, mas é possível notar que há variações como iluminação, deslocamento devido ao relevo ou de objetos moveis e desfoque. A rotação entre os recortes não é notada, pois há muito pouca rotação entre as imagens usadas para os testes.

Figura 16 - Seleção de recortes das imagens para verificação da qualidade

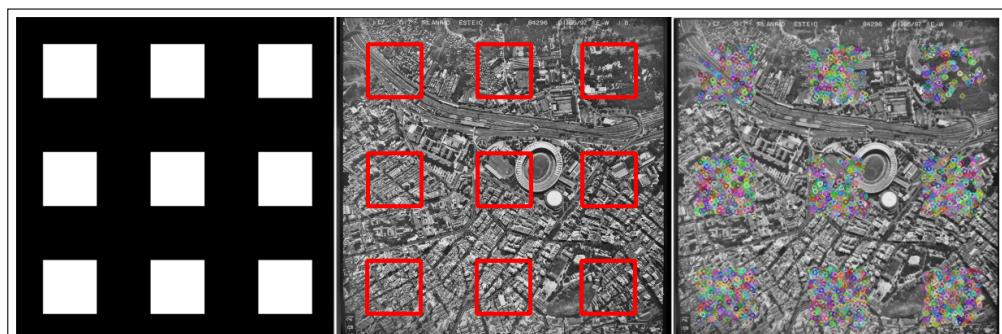


Legenda: Visualização de 20 pares de pontos selecionados nas imagens: (a) 16 e 17; e (b) 17 e 18.

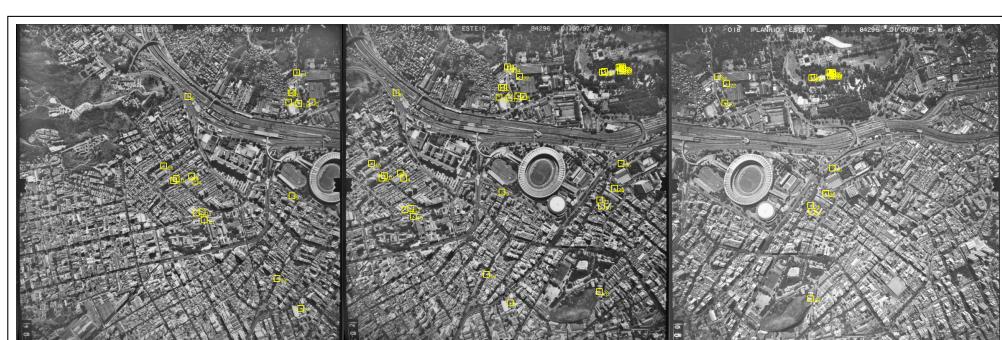
Fonte: O autor.

Com intuito de atender aos critérios de restrição do número de respostas ou as regiões com respostas em diferentes fases do processamento foram estabelecidos parâmetros adicionais como o de filtragem baseada em regiões de interesse para detecção de pontos de gruber, outro para limitar o total de pontos chave usados para correlação e mais um para limitar *inliers* usados para a solução final. Exemplos de padrões aplicáveis são oferecidos no Apêndice B. O padrão 3-3-3, por exemplo, produz resultados como o apresentado na Figura 17a. Este padrão limita a área de procura em todas as imagens utilizadas, portanto deve ser criada com zelo, pois caso não exista regiões sobrepostas nos pares estes não trarão resultado. A limitação do volume dos pontos de costura gerados pode ser visto na imagem 17b. Tal limite leva em consideração os melhores pontos, com menor RMSE, para gerar melhores resultados quando considerado a volumetria, por vezes excessiva, que os métodos poderiam retornar.

Figura 17 - Imagem ilustrativa da resposta ao utilizar máscara.



(a)



(b)

Legenda: Imagens ilustrativas da resposta ao utilizar o padrão 3-3-3, com alternativas no apêndice B, no método ORB com filtragem de pontos de costura.

- (a) Imagens do padrão 3-3-3, onde este se aplicaria na imagem 17 e dos pontos chave detectados para esta imagem; e
- (b) Imagens 16, 17 e 18 com pontos de costura restritos às regiões de interesse e nas quantidades parametrizadas.

Fonte: O autor.

Cabe destacar que a limitação do volume de pontos chave detectados faz-se importante principalmente nas imagens de grandes resoluções, para evitar que o tempo de correlação cresça de forma descontrolada. Outra alternativa para o processamento destas imagens seria a redução das mesmas, por parâmetro da solução criado para este fim, pois entende-se que isto está relacionado a uma diminuição considerável no número máximo de pontos que podem ser detectados e que a solução proposta está pronta para lidar com a apresentação dos pontos de costura na escala da imagem original.

### 3.3 Impacto do uso de diferentes parâmetros

Nesta seção são exibidas as principais variações que são introduzidas no resultado ao alterar os valores dos diferentes parâmetros, a fim de ilustrar como os valores de referência podem ser manipulados para atender a outras aplicações.

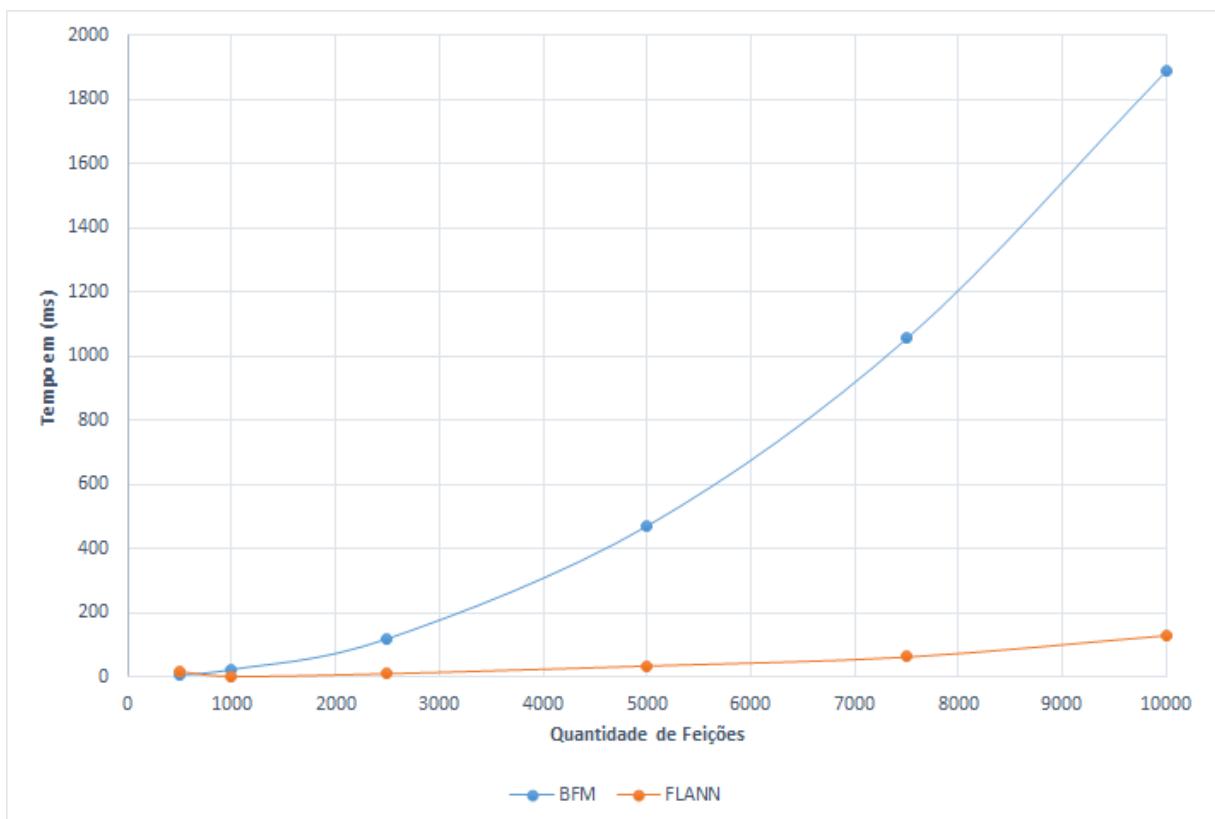
#### 3.3.1 Variação dos métodos de correlação

Para discutir a diferença entre os métodos de correlação, que em nosso caso são o BFM e o FLANN, e o motivo que levou à escolha do algoritmo FLANN como parâmetro base a Figura 18 exibe um gráfico que foi montado a partir dos resultados de tempo obtidos para ambos os métodos disponíveis. Adotou-se detecção e descrição pelo ORB, pois este sempre atende a quantidade de pontos chaves a serem extraídos, assim possibilitando a demonstração de escalabilidade do método de correlação. Foram adotadas 500, 1000, 2500, 5000, 7500, 10000 extrações, e plotadas as curvas para cada método de correlação representando o aumento do tempo de correlação em função do volume de pontos chave na entrada.

Pode-se captar no gráfico que o uso de força bruta não tem boa escalabilidade, pois o aumento de tempo para seu funcionamento se aproxima de um aumento exponencial, enquanto o aumento do método baseado em vizinho mais próximos aproximados é quase linear neste intervalo. Mesmo extrapolando o número de pontos extraídos, para 250 mil por exemplo, FLANN parece manter o comportamento quase linear, contudo neste intervalo seria impossível observar o ponto em que as curvas se cruzam. Deve ser ressaltado também que, quando a quantidade de pontos é muito baixa, inferior a 1000 no gráfico, este método tem tempo de execução pior que seu concorrente.

Apesar do desempenho, quando analisado o RMSE das mesmas soluções, mantido o total de pontos chave abaixo de 10000 pontos, o BFM retorna melhores correlatos devido principalmente à adoção de correlação cruzada, ou seja, atinge resíduos menores ao eliminar correlações que não são observáveis nos dois sentidos de análise (quando se compara uma imagem A com B e em seguida a imagem B com A).

Figura 18 - Gráfico da diferença entre métodos de correlação BFM e FLANN.



Legenda: Eixo vertical demonstra o tempo em milissegundos, e o eixo horizontal a quantidade de pontos chaves nas imagens de cada par.

Fonte: O autor.

### 3.3.2 Variação dos resíduos

Nesta seção demonstra-se as alterações observáveis do resultado quando variado o resíduo máximo admitido nas soluções geométricas computadas para os pares processados. O tempo de processamento total no ajustamento das soluções geométricas dos pares, o RMSE resultante por par e o número dos pontos de costura na saída são apresentados na Tabela 7, onde foi utilizado exclusivamente o ORB como método de detecção e descrição.

A fim de alcançar melhores resultados foram escolhidos valores máximos de resíduo para análise dentro das recomendações da OpenCV<sup>22</sup>, exceto por um valor abaixo dos limites sugeridos para observar a viabilidade de manter resultados com restrições mais acentuadas. Pode-se notar que mesmo com o resíduo máximo aumentado o RMSE pode não acompanhar esse aumento diretamente. Isso se dá por dois motivos: estamos apenas acrescentando poucos novos resíduos altos, que não passam a ser maioria no cálculo; e porque a solução continua sendo refinada (usando *inliers* apenas se forem usados métodos robustos) com o método de Levenberg-Marquardt. O custo de tempo diminui conforme o resíduo aumenta, isso se dá pois o processo de encontrar a matriz homográfica é um processo iterativo baseado em confiança. Portanto, quando este encontra muitas iterações com resíduo abaixo do solicitado, ele reduz o número de iterações restantes e acelera a entrega do resultado.

Tabela 7 - Resposta aos diferentes valores de resíduo máximo aplicáveis

Resíduo	RMSE		Média	Tempo total	Pontos de costura
	16 x 17	17 x 18			
0,5	0,334696	0,33891	0,336803	68,0047	169
1	0,658826	0,648102	0,653464	68,5509	537
2	1,19721	1,23801	1,21761	67,815	1467
3	1,83996	1,69305	1,766505	45,6131	2295
4	2,18077	2,13489	2,15783	23,2233	2697
5	2,39894	2,46765	2,433295	15,3314	3025
6	2,79574	2,88174	2,83874	10,8983	3374
7	2,9575	3,14462	3,05106	9,0656	3690
8	3,09234	3,33832	3,21533	8,54693	3856
9	3,22821	3,51434	3,371275	9,17461	3972
10	3,24963	3,68748	3,468555	7,52193	3771

Legenda: Resíduo: é valor escolhido ao executar o processo *findHomography* na solução; RMSE: é a raiz quadrada do erro médio em cada par de imagens; Média: dos RMSE observados; Tempo total: da verificação geométrica; e Pontos de costura: gerados pela solução.

<sup>22</sup> Mais informações podem ser encontradas na documentação do processo *findHomography* em <[https://docs.opencv.org/4.x/d9/d0c/group\\_\\_calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780](https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780)>

## CONCLUSÃO

Após diversos testes realizados com os métodos estudados para a solução a que este trabalho se aplica, se tratando de recursos de código aberto e passíveis de utilização no Projeto E-Foto, foi construído um programa de interface de linha de comando que pode ser integrado em versões futuras. O método SURF, apesar de oferecer boas respostas, não deve ser aproveitado com facilidade por conta de sua patente ainda vigente. Entende-se que deva permanecer válida até meados de 2034. Por tal motivo o código fonte resultante está preparado para desconsiderar esta opção quando compilado em distribuições mais comuns da OpenCV.

Dentre os demais métodos que foram estudados, o AKaze e o SIFT demonstraram maior tempo de execução quando comparados com o método ORB. Na comparação entre esses métodos quanto ao tamanho alocado, o ORB se mostrou o método mais eficiente, apesar de não haver muita discrepância entre a quantidade de memória requerida para os resultados entre este e o AKaze. Em comparação com o tamanho utilizado pelo método SIFT, este necessitou de maior alocação de memória para a descrição das feições, tendo um acréscimo de aproximadamente 300% no tamanho alocado por ponto.

A correlação e a verificação geométrica de pares depende principalmente da quantidade de dados passados a elas. Conforme foi discorrido no capítulo 3, a escolha do método de correlação afeta em grande parte a velocidade de processamento da solução, portanto, indicamos o uso da correlação por FLANN já que este apresenta melhor desempenho quando realiza a análise de grande quantidades de feições, o que é bastante comum no tipo de análise que este trabalho está realizando. Vale enfatizar que o uso de correlação cruzada por força bruta pode gerar soluções com menor erro geométrico, mas requer controle para limitar o número de pontos chaves detectados, haja visto o seu desempenho, e maiores estudos dos impactos que podem estar relacionados às restrições aplicáveis.

A verificação geométrica por sua vez não apresentou grandes discrepâncias nos métodos estudados quando analisados os tempos e tamanhos alocados. Ainda assim recomenda-se a execução de mais estudos que abordem o número de respostas e a própria solução geométrica para classificar os pares analisados. Na fotogrametria é comum, por exemplo, que haja uma expectativa de sobreposição das fotos de uma mesma faixa de voo e uma sobreposição entre faixas e o atendimento destas expectativas não são atualmente verificadas.

Todos os critérios propostos puderam ser estudados com os resultados numéricos e visuais distintos que foram apresentados. Contudo, seria inviável endereçar todo o conjunto de testes que foi realizado no capítulo de resultados. Ressalta-se que foram feitos testes unitários para cada variação esperada dos argumentos de entrada para a solução. Dados externos ao projeto, além do conjunto de dados apresentado na seção 2.1.3, fizeram-se necessários para a análise de escalabilidade e testes de hipóteses levantadas durante o projeto. Isto não invalida trabalhos futuros com mais detalhes da manutenção da escalabilidade para grandes blocos de imagens.

Conclui-se, portanto, que dentre os métodos expostos dois se destacam nos quesitos estudados: o ORB, por sua velocidade de processamento e distribuição espacial dos pontos de costura; e o SIFT, que apesar de ser um método mais custoso computacionalmente apresenta o melhor resultado quando comparamos o volume de pontos de costura e menor erro. Sugere-se ainda que os resultados aqui apresentados sirvam de incentivo para trabalhos futuros como: uma extensão para visualização blocos de imagens fotogramétricas, em 2D como num foto-índice ou em 3D para a exibição simultânea de fotos e pontos triangulados pela solução fotogramétrica; a criação ou adaptação de uma ou mais interfaces gráficas que incorporem o código derivado deste trabalho aos módulos do e-foto; e a realização de estudos para uso de OpenCV na extração automatizada de linhas e polígonos que sirvam como linhas de quebra no e-foto.

## REFERÊNCIAS

- ACM. *Association for Computing Machinery*. 2022. Disponível em: <<https://selects.acm.org/selections/getting-started-with-computer-vision>>. Acesso em: 23 mar. 2022.
- ALCANTARILLA, P.; NUOVO, J.; BARTOLI, A. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell.*, v. 34, n. 7, p. 1281–1298, 2011.
- BAY, H.; TUYTELAARS, T.; GOOL, L. V. Surf: Speeded up robust features. In: SPRINGER. *European conference on computer vision*. [S.l.], 2006. p. 404–417.
- BAZARGANI, H.; BILANIUK, O.; LAGANIERE, R. A fast and robust homography scheme for real-time planar target detection. *Journal of Real-Time Image Processing*, Springer, v. 15, n. 4, p. 739–758, 2018.
- CHUM, O.; MATAS, J. Matching with prosac - progressive sample consensus. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). [S.l.: s.n.], 2005. v. 1, p. 220–226 vol. 1. ISSN 1063-6919.
- CIARAMBINO, M. et al. Development, validation and test of optical based algorithms for autonomous planetary landing. In: . [S.l.: s.n.], 2016.
- COELHO, L.; BRITO, J. *Fotogrametria digital*. EDUERJ, 2007. ISBN 9788575111147. Disponível em: <<https://books.google.com.br/books?id=jVYENQAACAAJ>>.
- CORREA, N. C. da S.; LARANJA, R. M. *Mapeamento Fotogramétrico Digital: Um Estudo Comparativo para a Bacia Hidrográfica do Rio Piabanha nos Ambientes E-Foto e LPS*. Dissertação (Mestrado) — Departamento de Engenharia Cartográfica - UERJ, Rio de Janeiro, 2011.
- E-FOTO. *Uma Estação fotogramétrica livre*. 2022. Disponível em: <<http://www.efoto.eng.uerj.br>>. Acesso em: 24 março 2021.
- FANG, W. et al. Computer vision applications in construction safety assurance. *Automation in Construction*, v. 110, p. 103013, 2020. ISSN 0926-5805. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0926580519301487>>.
- FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, ACM New York, NY, USA, v. 24, n. 6, p. 381–395, 1981.
- FRICKER, P.; ROHRBACH, A. Pushbroom scanners provide highest resolution earth imaging information in multispectral bands. In: *Proceedings of IPSRS Hannover Workshop: High Resolution Earth Imaging for Geospatial Information*. [S.l.: s.n.], 2005.
- GAO, J. et al. Computer vision in healthcare applications. *Journal of Healthcare Engineering*, Hindawi, v. 2018, p. 4, mar. 2018.
- GOMES, J. F. S.; LETA, F. R. Applications of computer vision techniques in the agriculture and food industry: a review. *European Food Research and Technology*, Springer, v. 235, n. 6, p. 989–1000, 2012.

- HARRIS, C.; STEPHENS, M. et al. A combined corner and edge detector. In: CITESEER. *Alvey vision conference*. [S.I.], 1988. v. 15, n. 50, p. 10–5244.
- HUBEL, D. H.; WIESEL, T. N. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, v. 148, n. 3, p. 574–591, out. 1959.
- IBGE. *Instituto Brasileiro de Geografia e Estatística*. 2022. Disponível em: <<https://www.ibge.gov.br/pt/inicio.html>>. Acesso em: 06 abril 2022.
- IBM. *International Business Machines Corporation*. 2022. Disponível em: <<https://www.ibm.com/topics/computer-vision>>. Acesso em: 23 fev. 2022.
- ISPRS. *International Society for Photogrammetry and Remoto Sensing*. 2021. Disponível em: <<https://www.isprs.org/society/>>. Acesso em: 24 ago. 2021.
- LOWE, D. G. *Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image*. Google Patents, 1999. US Patent 6711293B1. Disponível em: <<https://patents.google.com/patent/US6711293B1/en>>.
- LOWE, D. G. Object recognition from local scale-invariant features. In: IEEE. *Proceedings of the seventh IEEE international conference on computer vision*. [S.I.], 1999. v. 2, p. 1150–1157.
- LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, Springer, v. 60, n. 2, p. 91–110, 2004.
- MARR, D. *Vision: A computational investigation into the human representation and processing of visual information*. [S.I.]: MIT press, 2010.
- MASSART, D. L. et al. Least median of squares: a robust method for outlier and model error detection in regression and calibration. *Analytica Chimica Acta*, v. 187, p. 171–179, 1986. ISSN 0003-2670. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0003267000829104>>.
- MIKHAIL, E. M. et al. Manual of photogrammetry, 5th edition. *Photogrammetric Engineering and Remote Sensing*, v. 72, p. 312, 2006.
- OPENCV. *About - OpenCV*. 2022. Disponível em: <<https://opencv.org/about/>>. Acesso em: 23 fev. 2022.
- PUPIM, P. A. B. *Fototriangulação por Feixes Perspectivos e sua aplicação na Versão Integrada da Estação Fotogramétrica Digital Educacional Livre (E-Foto)*. Monografia — Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2012.
- REGAL, N. A. de C. *Medições topográficas para avaliação da qualidade de medições fotogramétricas no ambiente e-foto*. Monografia — Departamento de Engenharia Cartográfica - UERJ, Rio de Janeiro, 2013.
- RIBEIRO, J. *Geração de Relatório de Projeto Fotogramétrico da Estação Fotogramétrica Digital Educacional Livre (E-Foto)*. Monografia — Departamento de Engenharia de Sistemas e Computação - UERJ, Rio de Janeiro, 2012.
- RUBLEE, E. et al. Orb: An efficient alternative to sift or surf. In: IEEE. *2011 International conference on computer vision*. [S.I.], 2011. p. 2564–2571.

RUIZ, L. A.; LERMA, J. L.; GIMENO, J. Application of computer vision techniques to support in the restoration of historical buildings. *INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY REMOTE SENSING AND SPATIAL INFORMATION SCIENCES*, NATURAL RESOURCES CANADA, v. 34, n. 3/B, p. 227–230, 2002.

VERMEER, M.; AYEHU, G. T. *Digital Aerial Mapping – a Hands-On Cour*. Helsinki, Finlândia: Aalto University's Department of the Built Environment, 2018. Disponível em: [⟨http://users.aalto.fi/~mvermeer/book.pdf⟩](http://users.aalto.fi/~mvermeer/book.pdf).

## APÊNDICE A – Código fonte

### A.1 Arquivo de configuração do projeto

#### A.1.1 src/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project( lo_final )
find_package( OpenCV REQUIRED )

if (EXISTS ${OpenCV_INCLUDE_DIRS}/opencv2/xfeatures2d/nonfree.hpp)
    add_definitions (-DNONFREEAVAILABLE)
endif()

include_directories( ${OpenCV_INCLUDE_DIRS} )
FILE(GLOB Sources *.cpp)
add_executable( lo_main ${Sources})
target_link_libraries( lo_main ${OpenCV_LIBS} )
```

### A.2 Arquivos de cabeçalho

#### A.2.1 src/macros.hpp

```
/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * Read the GNU GPL for more details. To obtain a copy of this license
 * see <http://www.gnu.org/licenses/>.
 */

#define GET_P_TIME(p) ({\
    auto t0 = std::chrono::high_resolution_clock::now(); \
    p; \
```

```

auto t1 = std::chrono::high_resolution_clock::now(); \
t1 - t0; })

#define GET_F_TIME(r, f) ({\
auto t0 = std::chrono::high_resolution_clock::now(); \
r = f; \
auto t1 = std::chrono::high_resolution_clock::now(); \
t1 - t0; })

#define GET_VECTOR_USAGE(vec) ({\
vec.size() * sizeof(vec.front()); })

#define GET_2DVECTOR_USAGE(vec) ({\
size_t sum = 0; \
for (auto it = vec.begin(); it < vec.end(); it++)\
    sum += (*it).size() * sizeof( (*it).front() ); \
sum; })

#define GET_CVMAT_USAGE(mat) ({\
(mat.cols*mat.rows) * sizeof(mat.type()); })

#define HUMAN_READABLE(m_usage) ({\
std::string suffix[] = {"B", "KB", "MB", "GB", "TB"}; \
char length = sizeof(suffix) / sizeof(suffix[0]); \
size_t bytes = m_usage; \
double dblBytes = bytes; \
unsigned char i = 0; \
if (bytes > 1024) \
    for (; bytes / 1024 > 0 && i < length - 1; i++, bytes /= 1024) \
        dblBytes = bytes / 1024.0; \
std::string result = std::to_string(dblBytes); \
result.substr(0, result.size()-4) + " " + suffix[i]; })

```

### A.2.2 src/point.hpp

```

/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but

```

```

* WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
* Read the GNU GPL for more details. To obtain a copy of this license
* see <http://www.gnu.org/licenses/>.
*/



#ifndef LO_POINT_H
#define LO_POINT_H


// STL dependencies
#include <map>
#include <list>
#include <vector>
#include <string>
#include <chrono>
#include <utility>
#include <fstream>
#include <iostream>

// OpenCV 4.5.4 dependencies
#include <opencv2/highgui.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/core/types.hpp>
#include <opencv2/features2d.hpp>
#ifdef NONFREEAVAILABLE
#include <opencv2/xfeatures2d.hpp>
#endif

// Point classes definition
namespace lo {

    class Measure {
        public:
            // Attributes
            size_t index;
            cv::Point2f pt;

            // Constructor
            Measure(size_t index, cv::Point2f pt);
    };

    class Point {
        public:
            // Attributes
            size_t index;
            std::vector< Measure > measures;
}

```

```

    // Constructor
    Point(size_t index = 0);

    // Methods
    bool add(const Measure &measure);
};

}

#endif

```

### A.2.3 src/image.hpp

```

/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * Read the GNU GPL for more details. To obtain a copy of this license
 * see <http://www.gnu.org/licenses/>.
 */

#ifndef LO_IMAGE_H
#define LO_IMAGE_H

// STL dependencies
#include <map>
#include <list>
#include <vector>
#include <string>
#include <chrono>
#include <utility>
#include <fstream>
#include <iostream>

// OpenCV 4.5.4 dependencies
#include <opencv2/highgui.hpp>
#include <opencv2/calib3d.hpp>

```

```

#include <opencv2/imgproc.hpp>
#include <opencv2/core/types.hpp>
#include <opencv2/features2d.hpp>
#ifdef NONFREEAVAILABLE
#include <opencv2/xfeatures2d.hpp>
#endif

// Image classes definition
namespace lo {

class Point;

class PointCompair {
public:
    // Methods
    bool operator ()(cv::Point2f const& a, cv::Point2f const& b);
};

class PointMap: public std::map<cv::Point2f, Point*, PointCompair> {
};

class Image {
public:
    // Attributes
    size_t index;
    std::string filename;
    cv::Mat descriptors;
    std::vector< cv::KeyPoint > keypoints;
    PointMap pointmap;
    std::chrono::duration<double, std::milli> t_read, t_detect, t_descript;
    size_t m_read, m_detect, m_descript;

    // Constructor
    Image(size_t idx = 0,
          std::string path = "");
}

// Methods
bool computeAndDetect(const cv::Ptr<cv::Feature2D> &detector,
                      std::string roi = "",
                      size_t scale = 1,
                      bool limitkpts = false,
                      size_t nfeatures = 10000,
                      bool verbose = false);
};

}

```

```
#endif
```

#### A.2.4 src/pair.hpp

```
/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * Read the GNU GPL for more details. To obtain a copy of this license
 * see <http://www.gnu.org/licenses/>.
 */

#ifndef LO_PAIR_H
#define LO_PAIR_H

// STL dependencies
#include <map>
#include <list>
#include <vector>
#include <string>
#include <chrono>
#include <utility>
#include <fstream>
#include <iostream>

// OpenCV 4.5.4 dependencies
#include <opencv2/highgui.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/core/types.hpp>
#include <opencv2/features2d.hpp>
#ifndef NONFREEAVAILABLE
#include <opencv2/xfeatures2d.hpp>
#endif

// Pair class definition
namespace lo {
```

```

class Image;

class Pair {
public:
    // Attributes
    Image *left, *right;
    bool discarded;
    cv::Mat homography;
    double RMSE;
    std::vector< double > errors;
    std::vector< std::pair< double, cv::DMatch > > matches;
    std::chrono::duration<double, std::milli> t_match, t_correct;
    size_t m_match, m_correct;

    // Constructor
    Pair(Image *left = nullptr, Image *right = nullptr);

    // Methods
    bool checkHomography(const cv::Ptr<cv::DescriptorMatcher> &matcher,
                         double maximumError = 2.0,
                         size_t limit = 0,
                         bool crosscheck = false,
                         double rate = 0.0,
                         bool verbose = false);
};

}

#endif

```

### A.2.5 src/control.hpp

```

/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * Read the GNU GPL for more details. To obtain a copy of this license

```

```

* see <http://www.gnu.org/licenses/>.
*/

#ifndef LO_CONTROL_H
#define LO_CONTROL_H

// STL dependencies
#include <map>
#include <list>
#include <vector>
#include <string>
#include <chrono>
#include <utility>
#include <fstream>
#include <iostream>

// OpenCV 4.5.4 dependencies
#include <opencv2/highgui.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/core/types.hpp>
#include <opencv2/features2d.hpp>
#ifndef NONFREEAVAILABLE
#include <opencv2/xfeatures2d.hpp>
#endif

// Join all project classes
#include "image.hpp"
#include "pair.hpp"
#include "point.hpp"

// ProcessControl class definition
namespace lo {

class ProcessController {
    // Command Line arguments
    bool verbose;
    bool crosscheck;
    bool limitkpts;
    bool scapePointList;
    double inlierRate;
    double residue;
    size_t nfeatures;
    size_t imageScale;
    size_t startPointIndex;
    size_t limitMatches;
    size_t nMeasures;
    std::string detectorType;
}

```

```

    std::string roiFile;
    std::string mode;
    std::string imagelist, pairslist, resultname, pointsname;

    // Internal method
    void makePointList(bool verbose = false);

public:
    // Attributes
    std::list< Point > points;
    std::map< size_t, Image > images;
    std::vector< Pair > pairs;
    cv::Ptr<cv::Feature2D> detector;
    cv::Ptr<cv::DescriptorMatcher> matcher;
    cv::Ptr<cv::CommandLineParser> parser;
    std::chrono::duration<double, std::milli> t_stich;
    size_t m_stich, stich_creation, stich_update, stich_merge;

    // Public methods
    bool readArguments(int argc, char **argv);
    bool runProcesses();
    bool saveResults();
    void printUsage();
};

}

#endif

```

### A.3 Arquivos de implementação

#### A.3.1 src/point.cpp

```

/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

```

```

* Read the GNU GPL for more details. To obtain a copy of this license
* see <http://www.gnu.org/licenses/>.
*/
#include "point.hpp"

namespace lo {

    Measure::Measure(size_t index, cv::Point2f pt) {
        this->index = index;
        this->pt = pt;
    }

    Point::Point(size_t index) {
        this->index = index;
    }

    bool Point::add(const Measure &measure) {
        // Prevents different measurements of the same point for an image
        for (size_t i = 0; i < measures.size(); i++)
            if (measures[i].index == measure.index)
                return false;
        // And add the measurement if is relevant
        measures.push_back(measure);
        return true;
    }

}

```

### A.3.2 src/image.cpp

```

/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * Read the GNU GPL for more details. To obtain a copy of this license
 * see <http://www.gnu.org/licenses/>.

```

```

*/
#include <algorithm>
#include <random>

#include "image.hpp"
#include "point.hpp"
#include "macros.hpp"

namespace lo {

Image::Image(size_t idx, std::string path) : index(idx), filename(path) {}

bool PointCompair::operator ()(cv::Point2f const& a, cv::Point2f const& b)
{
    return (a.x < b.x) || (a.x == b.x && a.y < b.y);
}

bool Image::computeAndDetect(const cv::Ptr<cv::Feature2D> &detector,
                             std::string roi,
                             size_t scale,
                             bool limitkpts,
                             size_t nfeatures,
                             bool verbose) {
    // Try to read the image
    cv::Mat img;
    t_read = GET_F_TIME(img, cv::imread(filename, cv::IMREAD_GRAYSCALE));
    // Check if image is loaded
    if (img.data != NULL) {
        // Apply scale, if is needed
        if (scale > 1) {
            cv::Mat shrink;
            cv::resize(img, shrink, cv::Size(), 1.0/scale, 1.0/scale, cv::INTER_AREA);
            img = shrink.clone();
        }
        // Define gruber region of interest to get keypoints
        cv::Mat mask;
        if (!roi.empty()) {
            cv::Mat originMask = cv::imread(roi, cv::IMREAD_GRAYSCALE);
            if (originMask.data == NULL) {
                std::cerr << "Failed_to_open_file:" << roi << std::endl;
                return false;
            }
            cv::resize(originMask, mask, img.size(), 0, 0, cv::INTER_LINEAR
);
        }
    }
}

```

```

    }

    // Detect keypoints
    t_detect = GET_P_TIME(detector->detect(img, keypoints, mask));
    // Apply restrictions, if is necessary
    if (limitkpts && nfeatures < keypoints.size()) {
        // This implies on points shuffle to ensure normal distribution
        auto rng = std::default_random_engine {};
        std::shuffle(keypoints.begin(), keypoints.end(), rng);
        // And vector clipping
        keypoints.erase(keypoints.begin() + nfeatures, keypoints.end());
    }
    // Compute descriptors to keypoints
    t_descript = GET_P_TIME(detector->compute(img, keypoints,
                                                descriptors));
    // And compute memory usage to main image objects
    m_read = GET_CVMAT_USAGE( img );
    m_detect = GET_VECTOR_USAGE( keypoints );
    m_descript = GET_CVMAT_USAGE( descriptors );
}

// Abort the process if there are any exceptions
else {
    std::cerr << "Could not read the file " << filename << std::endl;
    return false;
}

// Reports the image's keypoint count when prompted
if (verbose) {
    std::cout << "Image " << filename << " processing...\n";
    std::cout << "Time_Opening_Image:" << t_read.count() << "\n";
    std::cout << "Memory_usage_on_reading_image:" << HUMAN_READABLE(
        m_read) << "\n";
    std::cout << "Number_of_keypoints:" << keypoints.size() << "\n";
    std::cout << "Time_on_keypoints_detection:" << t_detect.count() <<
        "\n";
    std::cout << "Memory_usage_on_keypoints_detection:" <<
        HUMAN_READABLE(m_detect) << "\n";
    std::cout << "Time_on_keypoints_description:" << t_descript.count()
        () << "\n";
    std::cout << "Memory_usage_on_keypoints_description:" <<
        HUMAN_READABLE(m_descript) << "\n" << "\n";
}

// Abort if the number of keypoints is insufficient for the other
// processes in the flow
if (keypoints.size() < 4) {
    std::cerr << "The process failed to define keypoints on the set of "
        "images!\n";
    return false;
}

```

```

    return true;
}

}

```

### A.3.3 src/pair.cpp

```

/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * Read the GNU GPL for more details. To obtain a copy of this license
 * see <http://www.gnu.org/licenses/>.
 */

#include <algorithm>

#include "pair.hpp"
#include "image.hpp"
#include "macros.hpp"

namespace lo {

Pair::Pair(Image *left, Image *right) {
    this->left = left;
    this->right = right;
    discarded = true;
    m_match = m_correct = 0;
}

bool compareMatches(const std::pair<double, cv::DMatch> &i,
                    const std::pair<double, cv::DMatch> &j) {
    return i.first < j.first;
}

bool Pair::checkHomography(const cv::Ptr<cv::DescriptorMatcher> &matcher,
                           double me, size_t limit, bool crosscheck, double rate, bool verbose) {

```

```

// Running the images matching
std::vector< std::vector< cv::DMatch > > allMatches;
std::vector< cv::DMatch > goodMatches;
if (crosscheck) {
    t_match = GET_P_TIME(matcher->knnMatch(left->descriptors, right->
        descriptors, allMatches,1));

    // Discard matches based on OpenCV cross-correlation, when
    // requested by the user
    for(size_t i = 0; i < allMatches.size(); i++) {
        if (allMatches[i].size()==1){
            goodMatches.push_back(allMatches[i][0]);
        }
    }
}
else {
    t_match = GET_P_TIME(matcher->knnMatch(left->descriptors, right->
        descriptors, allMatches, 2));

    // Eliminate matches based on the proportion of nearest neighbor
    // distance as an alternative to cross-correlation as described in:
    // www.uio.no/studier/emner/matnat/its/TEK5030/v19/lect/
    // lecture_4_2_feature_matching.pdf
    for(size_t i = 0; i < allMatches.size(); i++) {
        if (allMatches[i].size() == 2) {
            cv::DMatch first = allMatches[i][0];
            float dist1 = allMatches[i][0].distance;
            float dist2 = allMatches[i][1].distance;
            if(dist1 < 0.8 * dist2)
                goodMatches.push_back(first);
        }
    }
}

// Find the homography
std::vector<cv::Point2f> f_pts, s_pts;
cv::Mat inliers;
for( size_t i = 0; i < goodMatches.size(); i++ ) {
    auto match = goodMatches[i];
    auto pair = std::make_pair(match.queryIdx, match.trainIdx);
    f_pts.push_back( left->keypoints[ pair.first ].pt );
    s_pts.push_back( right->keypoints[ pair.second ].pt );
}
t_correct = GET_F_TIME(homography, cv::findHomography(f_pts, s_pts, cv
    ::RANSAC, me, inliers));

// Discard the pair if there is no geometric solution

```

```

size_t N = cv::sum(inliers)[0];
if (homography.empty() || N < 4) {
    // Report the pair discard when prompted
    if (verbose) {
        std::cout << "No correlation could be found between the pair "
        << left->index << "x" << right->index << std::endl;
    }
    return false;
}
discarded = false;

// RMSE computing and inlier matches register
RMSE = 0.0;
for( size_t i = 0; i < goodMatches.size(); i++ ) {
    if (inliers.at<bool>(i)) {
        // Project the point on the first image onto the second image
        cv::Mat f_point = cv::Mat::ones(3, 1, CV_64F);
        f_point.at<double>(0) = f_pts[i].x;
        f_point.at<double>(1) = f_pts[i].y;
        cv::Mat f_point_projected = homography * f_point;
        f_point_projected /= f_point_projected.at<double>(2);
        // Check the distance between the expected point and the
        projected one
        cv::Mat s_point = cv::Mat::ones(3, 1, CV_64F);
        s_point.at<double>(0) = s_pts[i].x;
        s_point.at<double>(1) = s_pts[i].y;
        cv::Mat diff = s_point - f_point_projected;
        // Accumulate the residual error values
        double err = pow(diff.at<double>(0), 2) + pow(diff.at<double
            >(1), 2);
        RMSE += err;
        // Save the inlier match
        matches.push_back( std::make_pair(err, goodMatches[i]) );
    }
}
// Obtain the square root of residuals by the number of solution points
RMSE = sqrt(RMSE/N);
// Sort matches by error
std::sort(matches.begin(), matches.end(), compareMatches);
// And crop it if is needed
if (limit > 0 && matches.size() > limit)
    matches.erase(matches.begin()+limit, matches.end());

// Compute memory usage to main pair objects
m_match = GET_2DVECTOR_USAGE( allMatches );
m_correct = GET_VECTOR_USAGE( matches );

```

```

// Reports the pair's matches count when prompted
if (verbose) {
    std::cout << "Pair_" << left->index << "x" << right->index << "_"
    processing... \n";
    if (crosscheck)
        std::cout << "Total_distance_matches_available:_" << left->
            descriptors.rows * right->descriptors.rows << "\n";
    else
        std::cout << "Total_distance_matches_available:_" << 2 *
            allMatches.size() << "\n";
    std::cout << "Good_matches:_" << goodMatches.size() << "\n";
    std::cout << "Solution_(inlier)_matches:_" << N << "\n";
    std::cout << "The_homography_matrix_is:\n" << homography << "\n";
    std::cout << "RMSE:_" << RMSE << "\n";
    std::cout << "Time_on_matching_keypoints:_" << t_match.count() << "
        \n";
    std::cout << "Time_on_geometry_verification:_" << t_correct.count()
        << "\n";
    std::cout << "Memory_usage_on_correlate_descriptors:_" <<
        HUMAN_READABLE(m_match) << "\n";
    std::cout << "Memory_usage_on_remaining_matches:_" <<
        HUMAN_READABLE(m_correct) << "\n" << "\n";
}

// Test pair discard based on an inlier rate, when given rate in range
// (0,1]
if (rate > 0.0 && rate <= 1.0) {
    if (matches.size() < goodMatches.size() * rate) {
        std::cout << "Discard_pair_" << left->index << "x" << right->
            index <<
                ",_as_the_geometric_solution_has_acceptance_of_"
                <<
                100.0 * matches.size() / (double) goodMatches.size
                () <<
                "%_of_the_good_matches." << std::endl << std::endl
                ;
        discarded = true;
    }
}

return true;
}
}

```

### A.3.4 src/control.cpp

```

/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * Read the GNU GPL for more details. To obtain a copy of this license
 * see <http://www.gnu.org/licenses/>.
 */

#include "control.hpp"
#include "macros.hpp"

namespace lo {

static inline std::string &ltrim(std::string &s) {
    // Left trim string to avoid some errors on read files
    // from https://stackoverflow.com/questions/216823/how-to-trim-a-stdstring
    s.erase(s.begin(), std::find_if(s.begin(), s.end(),
                                    std::not1(std::ptr_fun<int, int>(std::
                                         isspace))));

    return s;
}

bool ProcessController::readArguments(int argc, char **argv) {

    const std::string keys =
        "@imagelist|<none>|list_with_index_and_path_for_images_to_
         process"
        "@pairslist|<none>|list_with_index_of_images_in_pairs_to_
         process"
        "@resultname|<none>|filename_to_save_the_resulting_image_
         measurements_process"
        "@pointsname|<none>|filename_to_save_point's_indexes_and_
         types_as_needed_by_e-foto"
        "crosscheck_c|<none>|use_crosscheck_with_brute_force_matcher_
         default_is_use_flann_with_Lowe's_ratio_test}"

```

```

" {detector_d|ORB|select_detector_type_between_AKAZE,_ORB,_
 SIFT_or_SURF }"
" {number_f|f|10000|suggest_a_number_(f)_of_features_to_
 retain_(adopted_by_ORB) }"
" {force_f|F|forces_the_upper_bound_(f)_of_features_to_
 _retain_on_detection_phase }"
" {gruber_roi_g|sets_a_path_to_gruber's_region_of_
 interest_image_file }"
" {init_index_i|l|start_index_to_new_stich_points }"
" {help_h|show_help_message }"
" {mode_m|FILE|select_mode_of_pair_aquisition_between_
 SEQUENCE,_ALL_or_FILE_guided }"
" {n_measures_n|0|filter_points_by_a_minimum_(n)_of_image_
 measurements }"
" {limit_out_o|0|limit_size_of_measurements_per_pair_on_
 the_output }"
" {list_pairs_l|0.0|write_a_list_of_pairs,_instead_of_list_of_
 _measures,_keeping_those_that_can_have_a_geometric_solution_
 with_an_inlier_rate_(l) }"
" {residue_r|2.0|define_maximum_residue_for_geometric_
 solution }"
" {scale_s|1|a_scale_denominator_to_reduce_the_images_
 size }"
" {verbose_v|show_all_internal_process_messages }";
parser = cv::makePtr<cv::CommandLineParser>(argc, argv, keys);

if ( parser->has("help") )
    return false;

verbose = parser->has("verbose");
crosscheck = parser->has("crosscheck");
limitkpts = parser->has("force_f");

mode = parser->get<std::string>("mode");
imagelist = parser->get<std::string>("@imagelist");
pairslist = parser->get<std::string>("@pairslist");
resultname = parser->get<std::string>("@resultname");
pointsname = parser->get<std::string>("@pointsname");
detectorType = parser->get<std::string>("detector");
roiFile = parser->get<std::string>("gruber_roi");
startPointIndex = parser->get<size_t>("init_index");
nMeasures = parser->get<size_t>("n_measures");
limitMatches = parser->get<size_t>("limit_out");
imageScale = parser->get<size_t>("scale");
nfeatures = parser->get<int>("number_f");
residue = parser->get<double>("residue");
inlierRate = parser->get<double>("list_pairs");

```

```

    scapePointList = (inlierRate > 0.0) && (inlierRate <= 1.0);
if (!parser->check())
{
    parser->printErrors();
    return false;
}

// Avoiding use SURF algorithm when nonfree definition is not available

#ifndef NONFREEAVAILABLE
if (detectorType == "SURF") {
    std::cerr << "This_detector_requires_availability_of_nonfree_opencv
    !\n";
    return false;
}
#endif

// Configure detector
if (detectorType == "AKAZE") {
    detector = cv::AKAZE::create();
}
else if (detectorType == "ORB") {
    detector = cv::ORB::create(nfeatures);
}
else if (detectorType == "SIFT") {
    detector = cv::SIFT::create();
}
else if (detectorType == "SURF") {
    detector = cv::xfeatures2d::SURF::create();
}
else {
    std::cerr << "Unexpected_detector_type!\\n";
    return false;
}

// Configure matcher
if (crosscheck) {
    if (detectorType == "AKAZE" || detectorType == "ORB")
        matcher = cv::makePtr<cv::BFMatcher>(cv::NORM_HAMMING,
                                              crosscheck);
    else
        matcher = cv::makePtr<cv::BFMatcher>(cv::NORM_L2, crosscheck);
}
else {
    // @TODO: Find better guidance on how lsh index parameters can be
    varied as data volume increases
    if (detectorType == "AKAZE" || detectorType == "ORB")

```

```

        matcher = cv::makePtr<cv::FlannBasedMatcher>( cv::makePtr<cv:::
            flann::LshIndexParams>(1,20,2) );
    else
        matcher = cv::makePtr<cv::FlannBasedMatcher>( );
    }
    detector->descriptorType();

// Set the initial state on counters
stich_creation = stich_update = stich_merge = 0;
return true;
}

bool ProcessController::runProcesses() {
    std::chrono::duration<double, std::milli> zero;
    zero = std::chrono::duration<double, std::milli>::zero();

// Try to read address list from images and instantiate related objects
std::ifstream inputImages(imagelist);
if (inputImages.is_open()) {
    size_t index, last_index;
    std::string filename;
    bool sequence = mode == "SEQUENCE";
    while (inputImages >> index) {
        if (images.find(index) == images.end()) {
            std::getline(inputImages, filename);
            images[index] = Image(index, ltrim(filename));
            // Make pairs with image's list sequence when defined by user
            if (sequence) {
                if (images.size() > 1) {
                    pairs.push_back( Pair(&images[last_index], &images[
                        index]) );
                }
                last_index = index;
            }
        }
        // Avoid index colision
    else {
        std::cerr << "There are repeated image indexes in the list
        !\n";
        inputImages.close();
        return false;
    }
}
else {
    std::cerr << "Failed to open file: " << imagelist << std::endl;
}
}

```

```

    return false;
}

// Select pairs as defined by user
if (mode == "FILE") {
    // FILE mode read image pair's list, create pairs and link to
    // images
    std::ifstream inputPairs(pairslist);
    if (inputPairs.is_open()) {
        size_t first, second;
        while (inputPairs >> first >> second) {
            auto left = images.find(first), right = images.find(second)
            ;
            if (left != images.end() && right != images.end())
                pairs.push_back( Pair(&left->second, &right->second) );
            // Avoid loose index
            else {
                std::cerr << "There are loosed image index on the list
                !\n";
                inputImages.close();
                return false;
            }
        }
    }
    else {
        std::cerr << "Failed to open file: " << pairslist << std::endl;
        return false;
    }
}
else if (mode == "ALL") {
    // ALL mode define all pairs
    for (auto &pivot: images)
        for (auto &elem: images)
            if (elem.first > pivot.first)
                pairs.push_back( Pair(&images[pivot.first], &images[
                    elem.first]) );
}
else if (mode != "SEQUENCE") {
    // SEQUENCE mode is handled while reading the image list
    // so it remains to check if an unexpected mode has been passed
    std::cerr << "Unexpected mode has passed!\n";
    return false;
}

// Image processing may abort if any image cannot be opened
for (auto &indexed_image: images) {
    if (!indexed_image.second.computeAndDetect( this->detector,

```

```

        roiFile,
        imageScale,
        limitKpts,
        nfeatures,
        verbose ))
    return false;
}

// Pair processing does not abort execution, but pairs can be discarded
for (size_t i = 0; i < pairs.size(); i++)
    pairs[i].checkHomography( this->matcher,
                               residue,
                               limitMatches,
                               crosscheck,
                               inlierRate,
                               verbose );

// Make the measurement selection process
if (scapePointList) {
    t_stich = zero;
    m_stich = 0;
}
else
    t_stich = GET_P_TIME(makePointList(verbose));

// Report main parts time consumption when prompted
if (verbose) {
    // Make zero allocated durations
    std::chrono::duration<double, std::milli> read, detect, descript,
    match, correct;
    read = detect = descript = match = zero;
    size_t m_read = 0, m_detect = 0, m_descript = 0, m_match = 0,
    m_correct = 0;
    // Sum elapsed time
    for (auto &indexed_image: images) {
        auto image = indexed_image.second;
        read += image.t_read;
        detect += image.t_detect;
        descript += image.t_descript;
        m_read += image.m_read;
        m_detect += image.m_detect;
        m_descript += image.m_descript;
    }
    for (size_t i = 0; i < pairs.size(); i++) {
        match += pairs[i].t_match;
        correct += pairs[i].t_correct;
        m_match += pairs[i].m_match;
    }
}
```

```

        m_correct += pairs[i].m_correct;
    }

    // And report it
    std::cout << "Stiches_created:" << stich_creation << "\n";
    std::cout << "Stiches_updated:" << stich_update << "\n";
    std::cout << "Stiches_merged:" << stich_merge << "\n" << "\n";

    std::cout << "Total_time_reading_images:" << read.count() << "\n";
    std::cout << "Total_time_on_keypoints_detection:" << detect.count
        () << "\n";
    std::cout << "Total_time_on_keypoints_description:" << descript.
        count() << "\n";
    std::cout << "Total_time_on_matching_keypoints:" << match.count()
        << "\n";
    std::cout << "Total_time_on_geometry_verification:" << correct.
        count() << "\n";
    std::cout << "Total_time_on_stich_point_registration:" << t_stich.
        count() << "\n";

    std::cout << "Total_memory_usage_on_reading_images:" <<
        HUMAN_READABLE(m_read) << "\n";
    std::cout << "Total_memory_usage_on_keypoints_detection:" <<
        HUMAN_READABLE(m_detect) << "\n";
    std::cout << "Total_memory_usage_on_keypoints_description:" <<
        HUMAN_READABLE(m_descript) << "\n";
    std::cout << "Total_memory_usage_on_correlate_descriptors:" <<
        HUMAN_READABLE(m_match) << "\n";
    std::cout << "Total_memory_usage_on_remaining_matches:" <<
        HUMAN_READABLE(m_correct) << "\n";
    std::cout << "Total_memory_usage_on_stich_point_registration:" <<
        HUMAN_READABLE(m_stich) << "\n";
}

return true;
}

bool ProcessController::saveResults() {
    // Write a list of pairs and scape when requested by the user
    if (scapePointList) {
        std::ofstream pairsList(resultname);
        if (pairsList.is_open()) {
            bool flag = false;
            for (size_t i = 0; i < pairs.size(); i++) {
                if (!pairs[i].discarded) {
                    if (flag)
                        pairsList << std::endl;
                    else
                        flag = true;
                }
            }
        }
    }
}
```

```

        pairsList << pairs[i].left->index << "\t" << pairs[i].
                    right->index;
    }
}

pairsList.close();
return true;
}

else {
    std::cerr << "Failed_to_create_file:" << resultname << std::
                endl;
    return false;
}
}

// Save the main results, all digital images measurements
std::ofstream imageMeasures(resultname);
if (imageMeasures.is_open()) {
    bool flag = false;
    for (auto point = points.begin(); point != points.end(); point++) {
        if (point->index != 0) {
            if (flag)
                imageMeasures << std::endl;
            else
                flag = true;
            size_t n = point->measures.size();
            for (size_t i = 0; i < n; i++)
                imageMeasures << point->measures[i].index << "\t"
                            << point->index << "\t"
                            << point->measures[i].pt.x * imageScale
                            << "\t"
                            << point->measures[i].pt.y * imageScale
                            << ((i == n-1)? "" : "\n");
        }
    }
    imageMeasures.close();
}

else {
    std::cerr << "Failed_to_create_file:" << resultname << std::endl;
    return false;
}

// And e-foto's ENH points, if is needed
if (!pointsname.empty()) {
    std::ofstream pointENH(pointsname);
    if (pointENH.is_open()) {
        bool flag = false;
        for (auto point = points.begin(); point != points.end(); point
              ++) {

```

```

    if (point->index != 0) {
        if (flag)
            pointENH << std::endl;
        else
            flag = true;
        pointENH << point->index << "\t" << "Photogrammetric"
            << "\t"
            << "0" << "\t" << "0" << "\t" << "0" << "\t"
            << "0" << "\t" << "0" << "\t" << "0";
    }
}
pointENH.close();
}

else {
    std::cerr << "Failed_to_create_file:" << pointsname << std::
        endl;
    return false;
}

return true;
}

void ProcessController::printUsage() {
    parser->printMessage();
}

void ProcessController::makePointList(bool verbose) {
    // For each remaining pair
    for (size_t i = 0; i < pairs.size(); i++) {
        if (pairs[i].discarded)
            continue;
        // Get image pointers to the pair in the process
        auto f_image = pairs[i].left;
        auto s_image = pairs[i].right;
        // For each remaining (inlier) match
        for (size_t j = 0; j < pairs[i].matches.size(); j++) {
            // Get the match indexes
            size_t f_match = pairs[i].matches[j].second.queryIdx;
            size_t s_match = pairs[i].matches[j].second.trainIdx;
            // Look on image's keypoints to get matched points
            cv::Point2f f_point = f_image->keypoints[f_match].pt;
            cv::Point2f s_point = s_image->keypoints[s_match].pt;
            // Access image's pointmaps to check for existing measurements
            auto f_pt_id = pairs[i].left->pointmap.find(f_point);
            auto s_pt_id = pairs[i].right->pointmap.find(s_point);
            bool f_isnew = ( f_pt_id == f_image->pointmap.end() );

```

```

bool s_isnew = ( s_pt_id == s_image->pointmap.end() );
// Chooses between generating, updating or merging stich points
// based on the state of existence of measurements in the
// images
if (f_isnew && s_isnew) {
    // Generate and store point
    stich_creation++;
    Point stich;
    points.push_back(stich);
    // Add measurements
    Measure f_measure(f_image->index, f_point);
    Measure s_measure(s_image->index, s_point);
    points.back().add(f_measure);
    points.back().add(s_measure);
    // Store point and update images
    f_image->pointmap[f_point] = &points.back();
    s_image->pointmap[s_point] = &points.back();
}
else if (f_isnew) {
    // Update point to add the first image measurement
    stich_update++;
    Point* stich = s_pt_id->second;
    Measure measure(f_image->index, f_point);
    stich->add(measure);
    // Update the first image
    f_image->pointmap[f_point] = stich;
}
else if (s_isnew) {
    // Update point to add the second image measurement
    stich_update++;
    Point* stich = f_pt_id->second;
    Measure measure(s_image->index, s_point);
    stich->add(measure);
    // Update the second image
    s_image->pointmap[s_point] = stich;
}
else {
    // Copy all measurements from the second point to the first
    // and update the images that pointed to the second point
    stich_merge++;
    Point* f_stich = f_pt_id->second;
    Point* s_stich = s_pt_id->second;
    for (size_t k = 0; k < s_stich->measures.size(); k++)
    {
        auto measure = s_stich->measures[k];
        f_stich->add( measure );
        s_image->pointmap[ measure.pt ] = f_stich;
    }
}

```

```

        }
        // Avoid the search cost on list to clear the second point
        s_stich->measures.clear();
    }
}

// Apply indexes to points
size_t stichesCount = m_stich = 0;
for (auto point = points.begin(); point != points.end(); point++) {
    // Avoiding points without measurements
    // or below the number of measurements required, if any
    if ( (nMeasures == 0 && point->measures.size() > 0) ||
        (nMeasures > 0 && point->measures.size() >= nMeasures) ) {
        point->index = startPointIndex + stichesCount++;
        // And computing memory usage to main stich point objects
        m_stich += sizeof(point->index) + point->measures.size() *
            sizeof(Measure);
    }
}
// Reports the stich point's count when prompted
if (verbose) {
    std::cout << "Total_stich_points:" << stichesCount << "\n";
}
}
}
}

```

### A.3.5 src/main.cpp

```

/* Copyright 2022 Luiz Otavio Soares de Oliveira by FEN/UERJ
 * This file is part of the final version of the TCC by Luiz Otavio, as
 * a requirement for obtaining a degree at this public university under
 * guidance of Irving Badolato professor.
 * The resulting software is free: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License (GNU GPL) as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 * Our code is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * Read the GNU GPL for more details. To obtain a copy of this license
 * see <http://www.gnu.org/licenses/>.
 */

```

```
#include "control.hpp"

// Main routine
int main (int argc, char** argv) {
    lo::ProcessController controller;

    if ( controller.readArguments(argc, argv) && controller.runProcesses()
        )
        controller.saveResults();
    else
        controller.printUsage();

    return 0;
}

/* Notes on the libraries adoption:
 * - We uses STL containers in this project: vectors mainly, but this
 * project requires at least one list, as this structure guarantees that
 * the iterators (or even simple pointers) will not be invalidated when
 * a new element is added or even removed. Maps were also used to ensure
 * a quick search for measurements taken on images and images by index
 * and to get images by index on process controller.
 * - We uses openCV algorithms: some experimental or non-free opencv
 * classes used for academic tests can be removed from compilation by
 * default if opencv from the standard linux repository is in use.
 */
```

## **APÊNDICE B – Padrões para definir regiões de interesse de gruber**

## B.1 Implementação dos padrões em formato ASCII Portable GrayMap

### B.1.1 Padrão 2-2-2 implementado no arquivo data/pattern0.pgm

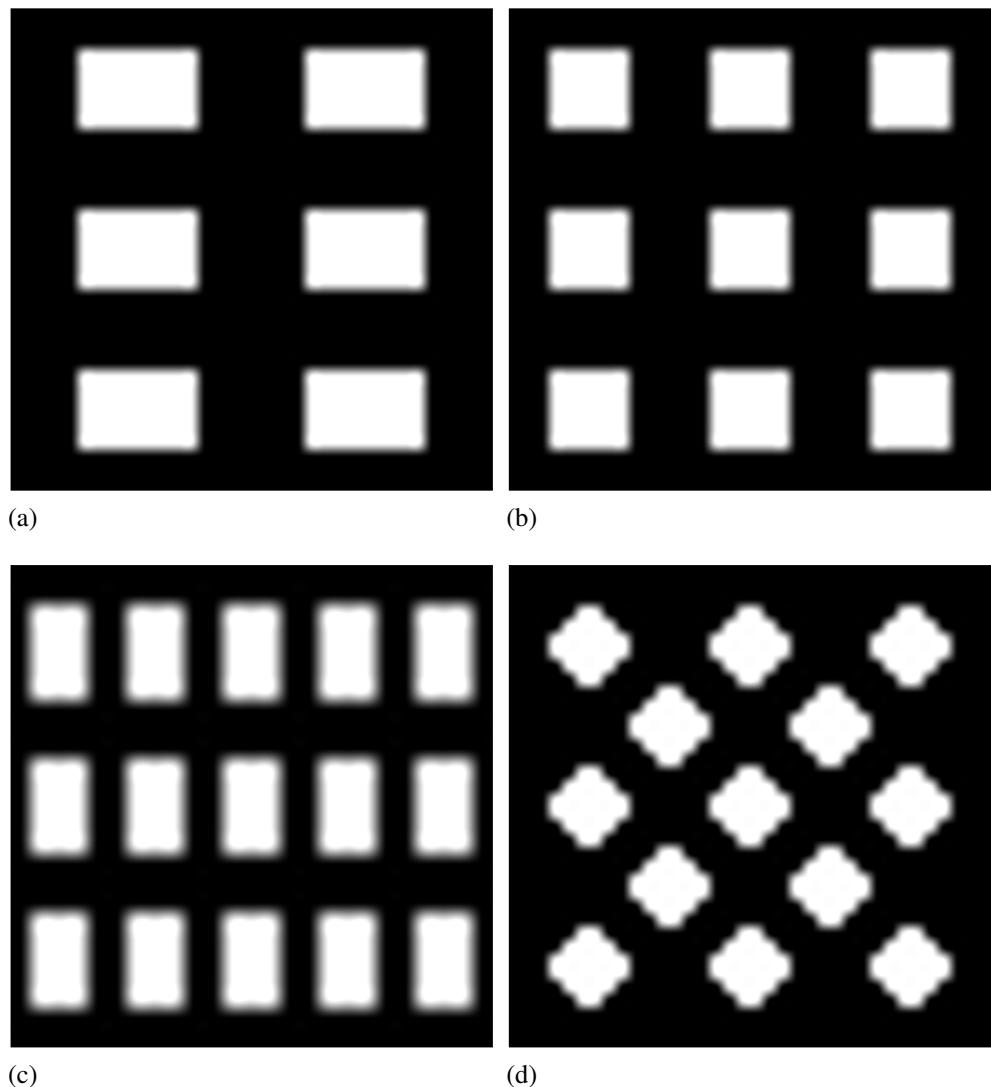
### B.1.2 Padrão 3-3-3 implementado no arquivo data/pattern1.pgm

### B.1.3 Padrão 5-5-5 implementado no arquivo data/pattern2.pgm

#### B.1.4 Padrão 3-2-3-2-3 implementado no arquivo data/pattern3.pgm

### B.1.5 Visualização dos padrões implementados

Figura 19 - Padrões interpolados para imagens de 360x360 pixels



Legenda: Visualização dos padrões de gruber implementados a saber:

19a 2-2-2; 19b 3-3-3; 19c 5-5-5; 19d 3-2-3-2-3.

Fonte: O autor.

## ANEXO A – Script de instalação do OpenCV partindo de seus códigos fonte

### A.1 Arquivo data/zxc-cv

```

#!/bin/bash

echo "OpenCV installation by Badolato (based on learnopencv.com)"
echo "This script has been tested on ubuntu 20.04 and may require"
echo "modification to newer versions."

# Check if has arguments supplied
if [[ $# -eq 0 ]]; then
    echo 'Usage:'
    echo '\tzxc-cv some-version-number'
    echo 'Sample:'
    echo '\tzxc-cv 3.4.8'
    exit 1
fi

# Define version
cvVersion=$1

# Save current working directory
cwd=$(pwd)

#echo "Update Packages"
apt -y update

echo "Install OS Libraries"
apt -y remove x264 libx264-dev
apt -y install build-essential checkinstall cmake pkg-config yasm
apt -y install git gfortran
apt -y install libjpeg8-dev libpng-dev
apt -y install software-properties-common
apt -y install jasper libxine2-dev libv4l-dev
cd /usr/include/linux
ln -s -f ../../libv4l1-videodev.h videodev.h
cd "$cwd"
apt -y install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
apt -y install libgtk2.0-dev libtbb-dev qt5-default
apt -y install libatlas-base-dev
apt -y install libfaac-dev libmp3lame-dev libtheora-dev
apt -y install libvorbis-dev libxvidcore-dev
apt -y install libopencore-amrnb-dev libopencore-amrwb-dev
apt -y install libavresample-dev

```

```

apt -y install x264 v4l-utils
apt -y install libprotobuf-dev protobuf-compiler
apt -y install libgoogle-glog-dev libgflags-dev
apt -y install libgphoto2-dev libeigen3-dev libhdf5-dev doxygen

echo "Download_and_clean_build_directories_to_installation"
git clone https://github.com/opencv/opencv.git ~/opencv
cd ~/opencv
git checkout $cvVersion
cd ..
git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib
git checkout $cvVersion
cd "$cwd"
rm -rf ~/opencv/build
rm -rf ~/opencv_contrib/build
mkdir /usr/local/opencv-"$cvVersion"

echo "Install_Python_Libraries"
apt-get -y install python3-dev python3-pip python3-venv python3-
    testresources
cd /usr/local/
python3 -m venv opencv-"$cvVersion"-py3
echo "#_Virtual_Environment_Wrapper" >> ~/.bashrc
echo "alias_workoncv-$cvVersion=\"source /usr/local/opencv-$cvVersion-py3/
    bin/activate\"" >> ~/.bashrc
source /usr/local/opencv-"$cvVersion"-py3/bin/activate
pip install wheel numpy scipy matplotlib scikit-image scikit-learn ipython
deactivate
cd "$cwd"

echo "Compile_and_install_OpenCV_with_contrib_modules"
cd ~/opencv
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local/opencv-"$cvVersion" \
    -D WITH_TBB=ON \
    -D WITH_V4L=ON \
    -D WITH_QT=ON \
    -D WITH_OPENGL=ON \
    -D WITH_CUDA=ON \
    -D ENABLE_FAST_MATH=1 \
    -D CUDA_FAST_MATH=1 \
    -D WITH_CUBLAS=1 \
    -D BUILD_EXAMPLES=ON \
    -D INSTALL_C_EXAMPLES=ON \

```

```
-D INSTALL_PYTHON_EXAMPLES=ON \
-D BUILD_opencv_python2=OFF \
-D BUILD_opencv_python3=ON \
-D OPENCV_PYTHON3_INSTALL_PATH=~/opencv-"$cvVersion"-py3/lib/
    python3.6/site-packages \
-D PYTHON3_EXECUTABLE=~/opencv-"$cvVersion"-py3/bin/python \
-D PYTHON3_LIBRARY=/usr/lib/python3.6/config-3.6m-x86_64-linux-gnu/
    libpython3.6.so \
-D PYTHON3_PACKAGES_PATH=~/opencv-"$cvVersion"-py3/lib/python3.6/
    site-packages \
-D PYTHON3_NUMPY_INCLUDE_DIRS=~/opencv-"$cvVersion"-py3/lib/python3
    .6/site-packages/numpy/core/include/ \
-D OPENCV_ENABLE_NONFREE=ON \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules ..
```

make -j6  
make install