

Trabalho 04 - Transformações Geometricas, Interpolação e Registro de Imagens

Lucas Nogueira Roberto ¹ - RA182553

¹Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)

Resumo. Este relatório detalha o desenvolvimento do Trabalho 04 da disciplina de Introdução ao Processamento Digital de Imagens (MC920/MO443), ministrada pelo Professor Dr. Hélio Pedrini durante o primeiro semestre de 2024. O trabalho é dividido em duas partes distintas. A primeira parte explora a aplicação de técnicas de interpolação para realizar transformações geométricas em imagens, como escala e rotação. A segunda parte utiliza ferramentas existentes para detectar pontos de interesse em imagens de uma mesma localização, associando-os para o registro das imagens e, assim, construindo uma imagem panorâmica resultante.

1. Introdução

1.1. Interpolação de Imagens

A interpolação é uma técnica fundamental no processamento digital de imagens, utilizada para estimar valores de pixels desconhecidos em uma imagem a partir de valores conhecidos. Esta técnica é essencial para diversas operações, como redimensionamento, rotação e correção de distorções. Em termos simples, a interpolação permite a criação de novos pixels com base em uma combinação ponderada dos pixels vizinhos, preservando a continuidade visual e a integridade da imagem.

Historicamente, a interpolação tem suas raízes na matemática e na engenharia, com aplicações que datam de séculos atrás. No século XIX, Carl Friedrich Gauss e outros matemáticos desenvolveram métodos de interpolação para resolver problemas de astronomia e física. Com o advento dos computadores na segunda metade do século 20, essas técnicas foram adaptadas e aprimoradas para o processamento digital de imagens. Apartir da década de 1960, com o surgimento dos primeiros algoritmos de processamento de imagens, a interpolação se tornou uma ferramenta indispensável, permitindo a manipulação eficiente de imagens digitais em pesquisas científicas e aplicações industriais.

Atualmente, algoritmos como a interpolação bilinear e bicúbica, são amplamente utilizados devido à sua simplicidade e eficácia, principalmente em software de edição de imagens e jogos. Com o aumento da demanda por imagens de alta resolução e a evolução das técnicas de aprendizado profundo, novos métodos de interpolação estão sendo desenvolvidos para melhorar a qualidade e a precisão dos resultados. Esses avanços têm aplicações significativas em campos como visão computacional, medicina, fotografia digital e muitos outros, demonstrando a importância contínua da interpolação.

1.2. Registro de Imagens

Outra técnica crítica no processamento de imagens é a de registro, em especial com o intuito de alinhar duas ou mais imagens de uma mesma cena capturadas de diferentes

ângulos. Esta técnica é essencial para a construção de imagens panorâmicas, onde várias fotos são combinadas para criar uma única imagem de maior campo de visão. O processo de registro envolve a identificação de pontos de interesse comuns entre as imagens, a correspondência desses pontos e a aplicação de transformações geométricas para alinhar as imagens de forma precisa.

Historicamente, o conceito de registro de imagens tem sido explorado desde o surgimento da fotografia no século XIX. No entanto, a automatização deste processo só se tornou viável com o advento da computação moderna. Nos anos 1970 e 1980, os primeiros algoritmos automáticos de registro começaram a ser desenvolvidos, utilizando técnicas de correlação e transformadas de Fourier. Com o avanço das tecnologias de captura de imagens e o aumento do poder computacional, métodos mais sofisticados, baseados em características locais e técnicas de otimização, foram introduzidos. Softwares populares, como o Adobe Photoshop e o Hugin, automatizam este processo, permitindo aos usuários criar panorâmicas de alta qualidade com facilidade.

2. Projeto

2.1. Funcionalidade

Neste trabalho foi implementado um software simples de edição de imagens capaz de performar as seguintes operações:

- Escalar uma imagem por um dado fator S com precisão de ponto flutuante
- Rotacionar uma imagem por um dado ângulo θ com precisão de ponto flutuante
- Redimensionar uma imagem para uma dimensão (w, h) , com w e h inteiros
- Montar a Panorâmica de duas imagens de entrada

Com os seguintes métodos de interpolação:

- Vizinho mais próximo
- Bilinear
- Bicubica
- Lagrangeana

Na opção de gerar a imagem panorâmica é preciso ser informado um limiar de correspondência das imagens que irá filtrar os pontos de interesse e pode ser esolhido dentre os seguintes algoritmos de descritores:

- SIFT (*Scale Invariant Feature Transform*)
- BRIEF (*Robust Independent Elementary Features*)
- ORB (*Oriented FAST, Rotated BRIEF*)

Assim ambas as partes do projeto são chamadas pelo mesmo ponto de entrada no código, a estrutura dos arquivos será detalhada em seguida

2.2. Estrutura Geral

Em uma visão geral do projeto, a sua implementação em código foi bem direta ao ponto, primeiro implementando as 3 transformações geometricas (escala, rotação e redimensionamento) tomando como base a interpolação mais simples de vizinho mais próximo e em seguida implementando os outros métodos, por fim foi feito a segunda etapa do registro e criação da panorâmica. Foram empregadas as bibliotecas **NumPy**, **OpenCV**, **Scikit-Image**, **Matplotlib** ao longo de 5 arquivos: **app.py**, **my_cv.py**, **interpolations.py**, **matching.py** e **tests.py**

- **app.py:** Demonstra as funcionalidades do software na linha de comando. Quando executado interage com o usuário por meio de uma interface que primeiro pergunta qual operação quer ser feita e em seguida vai requerindo as informações necessárias para realizar tal operação. Esse arquivo também checa se as entradas estão de acordo com o esperado e informa o usuário caso tenha algum problema.
- **my_cv.py:** Serve como uma camada de abstração para ler, salvar e fazer operações em imagens. Neste arquivo foram implementadas as operações de transformações geometricas do zero utilizando multiplicação de matrizes, técnicas de vetorização com arrays numpy e as funções definidas em **interpolations.py**. Nele também foram criadas abstrações em cima das funções correspondentes presentes nas bibliotecas **OpenCV** e **Scikit-Image** para fins de comparação entre as diferentes implementações.
- **interpolations.py:** Neste módulo são definidas todas as funções de interpolação mencionadas anteriormente, cada uma delas recebe a imagem original e retorna a imagem interpolada.
- **matching.py:** Este arquivo é responsável por definir as funções necessárias para criar a imagem panorâmica das duas imagens de entrada, para isso ele utiliza das funções nativas da biblioteca **OpenCV**.
- **tests.py** Tem o papel de testar as funções implementadas nos módulos **my_cv.py** e **matching.py** com as imagens da pasta **data**, realiza as comparações entre as funções de transformações geometricas das diferentes bibliotecas e mostra os resultados das diferentes escolhas de descritores para a imagem panorâmica lado a lado

2.3. Uso do Software

O software é executado de forma simples, chamando **python app.py** dentro da pasta do projeto. Conforme mencionado anteriormente, ele utiliza uma CLI para interagir com o usuário, oferecendo opções disponíveis e verificando qualquer entrada incorreta. Abaixo, são apresentados exemplos de uso:

```

[1] ~ > ~/python/data-science/Image-Editor> ls
app.py data_interpolations.py matching.py my_cv.py __pycache__ src tests.py
[2] ~ > ~/python/data-science/Image-Editor> python app.py
Welcome! To the Image editor 2049!
What operation do you want to perform ?
(1) Scale
(2) Rotate
(3) Resize
(4) Panoramic Merge
(5) Exit
Choose an operation: 2
Please insert the image path: /home/lucas/p/python/data-science/Image-Editor/data/input_images/baboon.png
libpng warning: iCCP: known incorrect sRGB profile
Please insert the folder where to save the result image: /home/lucas/p/python/data-science/Image-Editor/data/output_images
Choose the rotation angle (counter-clockwise): 45
Please choose which kind of interpolation you would like to use:
(1) Nearest Neighbor
(2) Bilinear
(3) Bicubic
(4) Lagrangean
Choose an interpolation: 4
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in "/home/lucas/.local/lib/python3.11/site-packages/cv2/qt/plugins"
[3]

```

Figura 1. Exemplo de uso

2.4. Transformações Geometricas e Interpolações

Cada uma das operações de transformação de imagem (rotação, escala e redimensionamento) foi implementada como uma função separada: **my_scale_image()**, **my_rotate_image()**, e **my_resize_image()**. Essas funções utilizam a formulação matricial para definir a matriz de transformação necessária e, em seguida, chamam a função **transform_image()** para aplicar tal transformação na matriz de pixels que representa a imagem de entrada. Em seguida será explicado como cada uma dessas rotinas funciona:

A função **my_scale_image()** escala uma imagem por um fator em ponto flutuante especificado, aplicando a interpolação para manter a qualidade visual. Ela constrói uma matriz de escala com base no fator fornecido, aplicando ele de forma igual tanto na largura quanto na altura e então passa essa matriz para a função **transform_image**, que realiza de fato a transformação:

```
def my_scale_image(image, scale, interpolation='bilinear'):
    scale_matrix = np.array([
        [scale, 0, 0],
        [0, scale, 0],
        [0, 0, 1]
    ])

    return transform_image(image, scale_matrix, interpolation)
```

A função **my_rotate_image** é responsável por rotacionar uma imagem em um ângulo especificado, aplicando a interpolação para manter a qualidade visual. Primeiro o ângulo de rotação é invertido para assegurar uma rotação no sentido anti-horário e em seguida, esse ângulo é convertido de graus para radianos e os componentes seno e cosseno necessários são calculados para construir a matriz de rotação que é passada para a função **transform_image**:

```
def my_rotate_image(image, angle, interpolation='bilinear'):
    theta      = np.radians(-angle)
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)

    rotation_matrix = np.array([
        [cos_theta, -sin_theta, 0],
        [sin_theta, cos_theta, 0],
        [0, 0, 1]
    ])

    return transform_image(image, rotation_matrix, interpolation)
```

A função **my_resize_image** redimensiona uma imagem para novas dimensões especificadas pelo usuário. Ela primeiro calcula os fatores de escala para as dimensões de largura e altura com base nas novas dimensões fornecidas. Em seguida, cria uma matriz de redimensionamento utilizando esses fatores de escala e utiliza a função **transform_image** para aplicar a transformação:

```

def my_resize_image(image, dim, interpolation='bilinear'):
    height, width = image.shape[:2]
    new_width, new_height = dim
    x_scale = float(new_width) / float(width)
    y_scale = float(new_height) / float(height)

    resize_matrix = np.array([
        [x_scale, 0, 0],
        [0, y_scale, 0],
        [0, 0, 1]
    ])

    return transform_image(image, resize_matrix, interpolation)

```

A função **transform_image** aplica uma transformação geométrica em uma imagem usando uma matriz de transformação e um método de interpolação. Inicialmente, a função calcula os novos limites da imagem transformada aplicando a matriz de transformação aos cantos da imagem original. Em seguida, ela cria uma nova imagem com as dimensões calculadas e ajusta a matriz de transformação para corrigir a translação necessária. A função então gera uma grade de coordenadas na imagem de saída e calcula as coordenadas correspondentes na imagem original utilizando a inversa da matriz de transformação. Por fim, ela verifica se essas coordenadas estão dentro dos limites da imagem original e aplica a interpolação escolhida para determinar os valores dos pixels na nova imagem transformada.

Acerca das interpolações experimentadas temos a seguir uma breve explicação de cada uma junto de algumas observações feitas durante testes:

- **Interpolação do vizinho mais próximo:** Simplesmente arredonda as coordenadas (x, y) para os valores de pixel mais próximos na imagem, resultando em uma abordagem direta, porém propensa a artefatos visíveis, especialmente em transições abruptas de valores de pixels.
- **Interpolação Bilinear:** Calcula o valor de um pixel através de uma média ponderada dos quatro pixels mais próximos, determinados pela parte inteira das coordenadas (x, y). Esta técnica utiliza pesos de interpolação baseados na distância do ponto desejado em relação aos pixels vizinhos, proporcionando uma transição suave e eficaz entre os valores dos pixels adjacentes na imagem.
- **Interpolação Bicúbica:** Utiliza polinômios cúbicos para calcular valores entre os pixels vizinhos, preservando detalhes finos e oferecendo uma transição ainda mais suave entre cores e detalhes na imagem. Este método parece mais útil em situações onde a qualidade visual é crucial.
- **Interpolação Lagrangeana:** Ajusta uma curva que passa exatamente pelos pontos de dados conhecidos na imagem. Isso proporciona uma interpolação precisa, especialmente em áreas com grandes variações nos valores dos pixels. É interessante que essa interpolação não está presente nas bibliotecas de visão computacional famosas, o que mostra a flexibilidade ganhada ao implementar tais interpolações do zero.

Após completar o código desta seção, explorei a semelhança dos resultados com as imagens obtidas utilizando bibliotecas conhecidas como OpenCV e Scikit-Image. Para

facilitar a comparação, abstrai as operações em funções adicionais no módulo **my_cv.py**, projetadas para receber os mesmos argumentos das funções que implementei do zero. Em seguida, criei o arquivo **tests.py** para realizar experimentos, onde comparei as imagens simplesmente mostrando elas na tela e onde também calculei algumas métricas de similaridade do Scikit-Image para cada par de imagens, como a distância de Hausdorff, a informação mútua normalizada (NMI) e o erro quadrático médio normalizado (NRMSE), alguns resultados podem ser vistos abaixo:

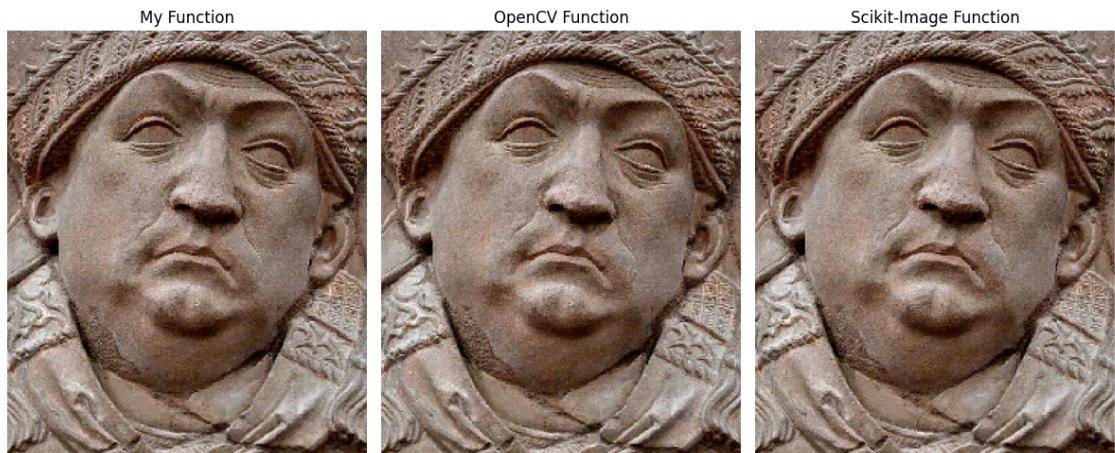


Figura 2. Transformação de Escala de 0.54x e interpolação vizinho mais próximo

	Hausdorff	NMI	NRMSE
My vs OpenCV	1.414	1.271	0.148
My vs Scikit-Image	1.414	1.11	0.191
OpenCV vs Scikit-Image	1.414	1.094	0.211

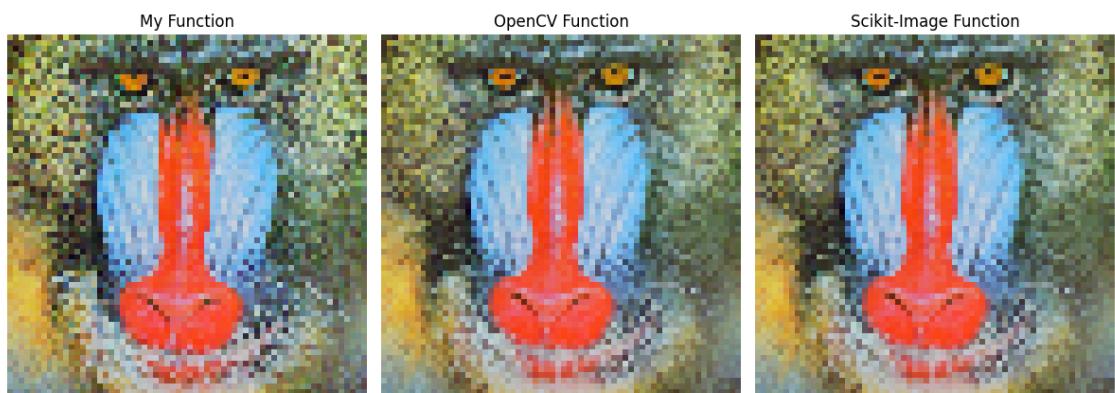


Figura 3. Transformação de Redimensionamento para (64, 64) e interpolação bilinear

	Hausdorff	NMI	NRMSE
My vs OpenCV	0.0	1.102	0.268
My vs Scikit-Image	0.0	1.102	0.268
OpenCV vs Scikit-Image	0.0	1.854	0.002

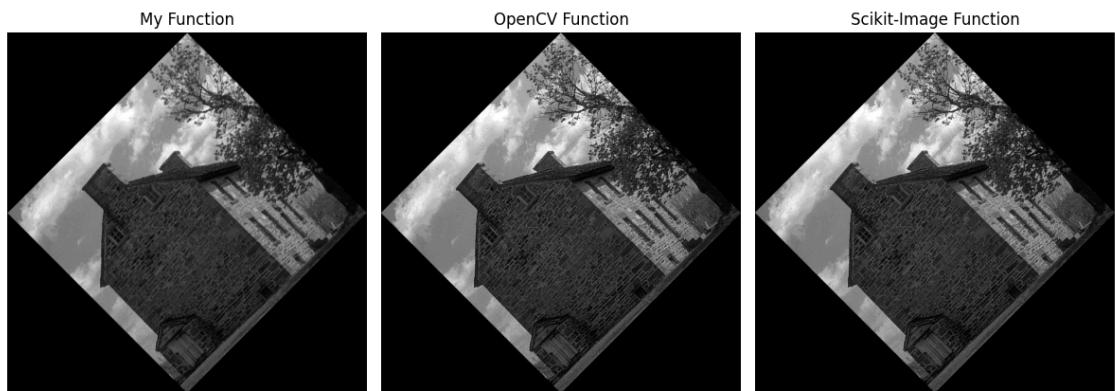


Figura 4. Transformação de Rotação de 45 graus e interpolação bicúbica

	Hausdorff	NMI	NRMSE
My vs OpenCV	1.414	1.325	0.140
My vs Scikit-Image	2.236	1.304	0.168
OpenCV vs Scikit-Image	2.0	1.284	0.152

Com essa análise, foi possível observar que mesmo entre as duas principais bibliotecas de processamento de imagens, as implementações das técnicas de transformações geométricas podem apresentar diferenças, como evidenciado pelas variações nas métricas e alguns artefatos perceptíveis em imagens de menor escala. No entanto, essas diferenças são mínimas e tanto o funcionamento das técnicas quanto das interpolações implementadas do zero parecem estar corretos. Isso sugere que, apesar das variações nas implementações, os resultados obtidos são consistentes e satisfatórios para aplicações práticas. Interessante notar que as imagens geradas pelo OpenCV e o Scikit-Image foram mais próximas quando usado a interpolação bilinear.

Como a Interpolação Lagrangeana não está presente nem no OpenCV nem no Scikit-Image, não foi possível compará-la diretamente com esses métodos. No entanto, uma inspeção visual das imagens resultantes indica que a Interpolação Lagrangeana funciona consideravelmente bem:

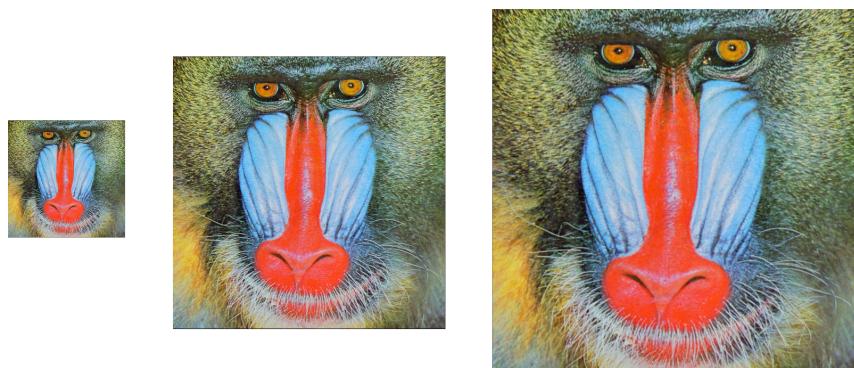


Figura 5. Transformação de Escala no baboon (0.45x, 1.00x, 1.45x) usando a interpolação lagrangeana

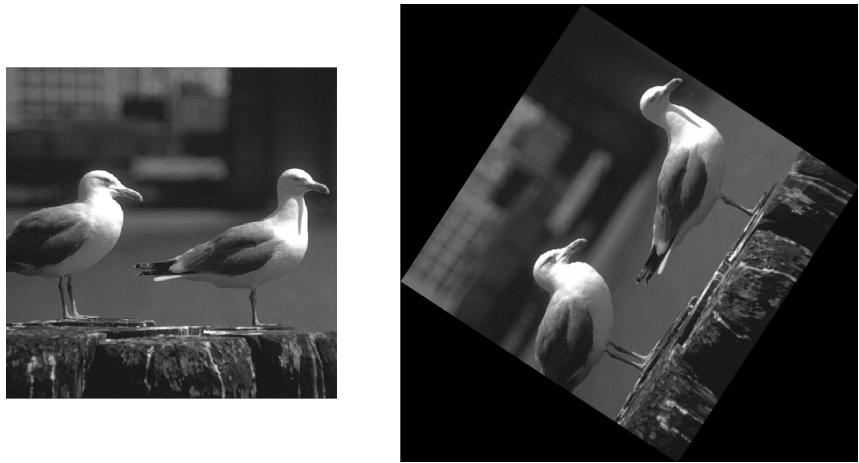


Figura 6. Transformação de Rotação na gaivota (0.00° e 56.8°) usando a interpolação lagrangeana

Importante lembrar que ao mostrar essas imagens nesse relatório elas ainda podem estar sujeitas a interpolações do próprio visualizador de pdf

2.5. Registro e Panorâmica

Ao selecionar a opção "Panoramic Merge", o usuário é solicitado a fornecer o caminho para duas imagens. Diferentemente das outras opções disponíveis, este processo invoca as funções no módulo **matching.py** invés de apenas o **my_cv.py** para realizar a fusão panorâmica das imagens fornecidas. Esse processo depende das rotinas **get_keypoints_and_descriptors()** e **match_images()**, sendo a última o ponto de entrada.

Inicialmente, as imagens são redimensionadas para terem a mesma altura, mantendo a proporção de aspecto. Isso é feito para garantir que ambas as imagens tenham a mesma escala ao longo do eixo vertical. O primeiro passo é converter as imagens coloridas de entrada em imagens em tons de cinza (gray1 e gray2) usando cv2.cvtColor. Essa conversão é realizada pois os métodos de correspondência de imagem do OpenCV operam de forma mais eficiente em imagens em escala de cinza.

Em sequência são encontrados pontos de interesse (keypoints1 e keypoints2) e descritores locais (descriptors1 e descriptors2) para o par de imagens fornecidos utilizando a função **get_keypoints_and_descriptors**. Esta função recebe como parâmetro o tipo de descritor, que determina qual algoritmo será utilizado: SIFT, BRIEF ou ORB. Como pode ser visto abaixo

```
def get_keypoints_and_descriptors(image, descriptor_type='SIFT'):
    if descriptor_type == 'SIFT':
        descriptor = cv2.SIFT_create(contrastThreshold=0.01)
        keypoints, descriptors = descriptor.detectAndCompute(image,
                                                               None)
    elif descriptor_type == 'BRIEF':
        fast = cv2.FastFeatureDetector_create()
        keypoints = fast.detect(image, None)
        brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()
        keypoints, descriptors = brief.compute(image, keypoints)
    elif descriptor_type == 'ORB':
        descriptor = cv2.ORB_create()
```

```

keypoints, descriptors = descriptor.detectAndCompute(image,
                                                    None)
else:
    raise ValueError("Unsupported descriptor type. Choose 'SIFT', "
                     "'BRIEF', or 'ORB'.")

return keypoints, descriptors

```

Em seguida, os descritores das duas imagens são comparados para calcular as distâncias (ou similaridades) entre cada par de descritores, utilizando o ‘cv2.BFMatcher’ com distância Euclidiana (‘cv2.NORM_L2’) e ‘crossCheck=True’ para obter os pares de correspondências mais próximos. As correspondências são ordenadas por distância e as melhores correspondências (‘good_matches’) são selecionadas com base em um limiar especificado pelo usuário (‘threshold’). Este limiar representa a porcentagem das melhores correspondências a serem utilizadas.

Então é aplicado a técnica RANSAC (cv2.findHomography) para estimar a matriz de homografia (H) entre os pontos de interesse das duas imagens. A matriz de homografia é essencial para mapear pontos correspondentes entre as duas imagens, permitindo a criação de uma imagem panorâmica correta.

As dimensões da imagem panorâmica final são determinadas, definindo os cantos da imagem de entrada e mapeando-os utilizando a transformação perspectiva da matriz de homografia (H). Em seguida, é aplicada a transformação perspectiva (‘cv2.warpPerspective’) na primeira imagem (img1) para alinhá-la com a segunda imagem (img2), utilizando a matriz de homografia ajustada por uma matriz de translação. Isso resulta na imagem da primeira câmera na perspectiva da segunda câmera. As duas imagens são então mescladas (panorama) na imagem panorâmica final, assegurando que a segunda imagem (img2) esteja alinhada corretamente com a primeira. Assim, é criada uma única imagem que representa uma visão panorâmica combinada das duas imagens originais.

Para testar o funcionamento do código e comparar os resultados de cada uma das técnicas, plotei tanto as correspondências entre os keypoints quanto a imagem panorâmica final. Essas comparações foram exibidas lado a lado para cada técnica utilizada, várias imagens de teste e valores de threshold foram usados também como é possível observar abaixo:

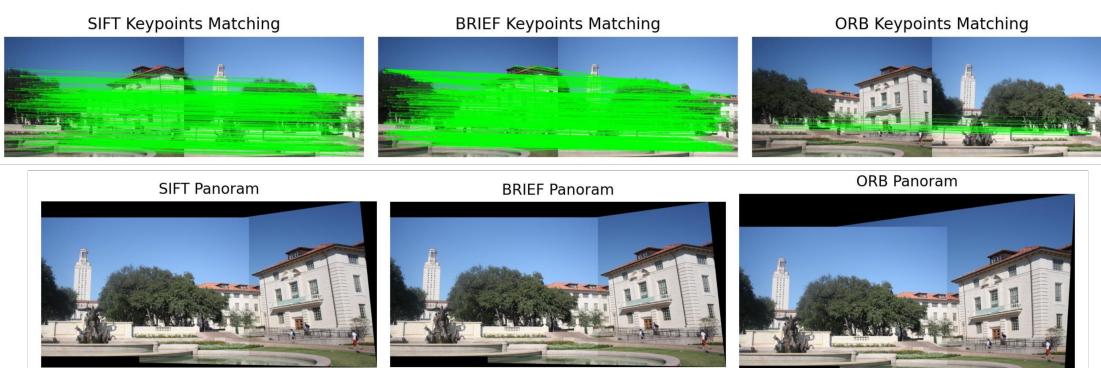


Figura 7. Foto1AB - Threshold = 0.20

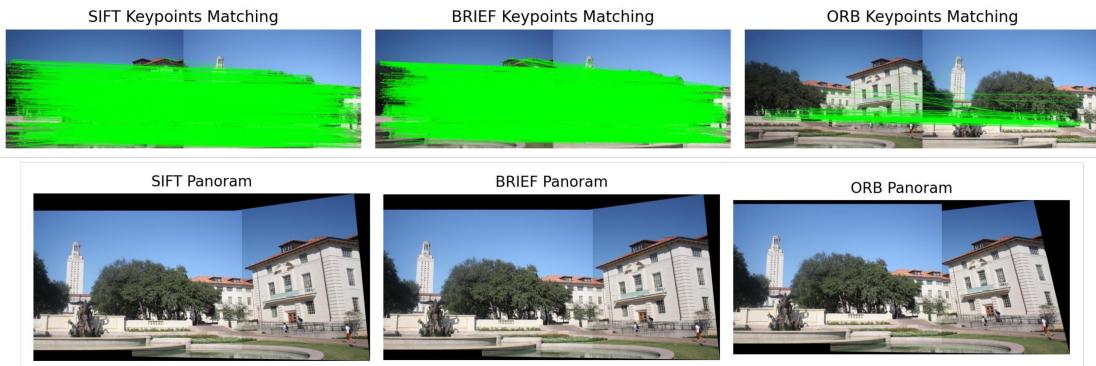


Figura 8. Foto1AB - Threshold = 0.60

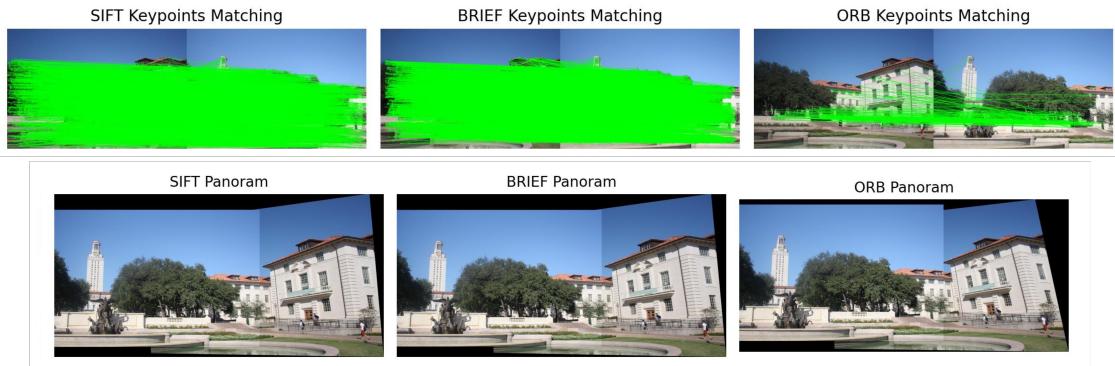


Figura 9. Foto1AB - Threshold = 1.00

Variando o threshold é possível notar, principalmente na técnica ORB, como a quantidade de correspondências reduz e como isso pode afetar o matching das imagens com pequenos desalinhamentos. Mesmo variando bastante esse threshold em todas as imagens de teste problemas mais graves só ocorreram quando ele foi menor que 0.10, o que mostra a robustez desses métodos para imagens bem comportadas.

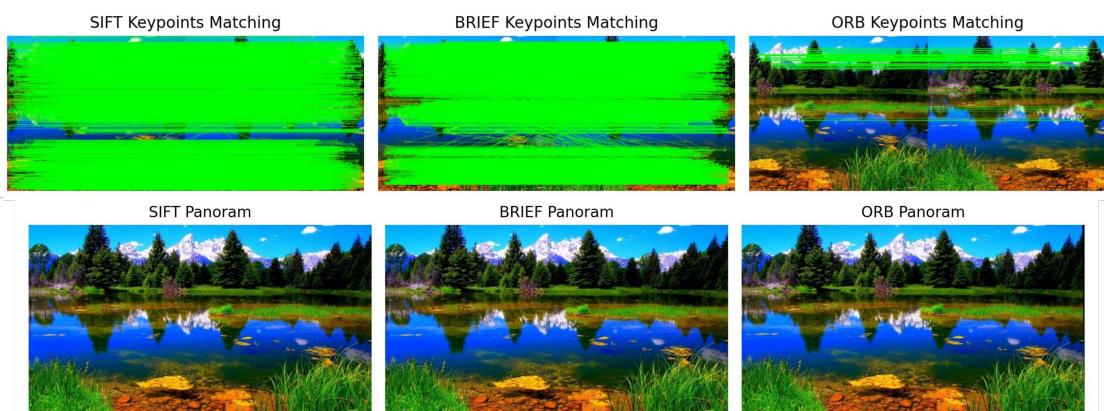


Figura 10. Foto2AB - Threshold = 0.60

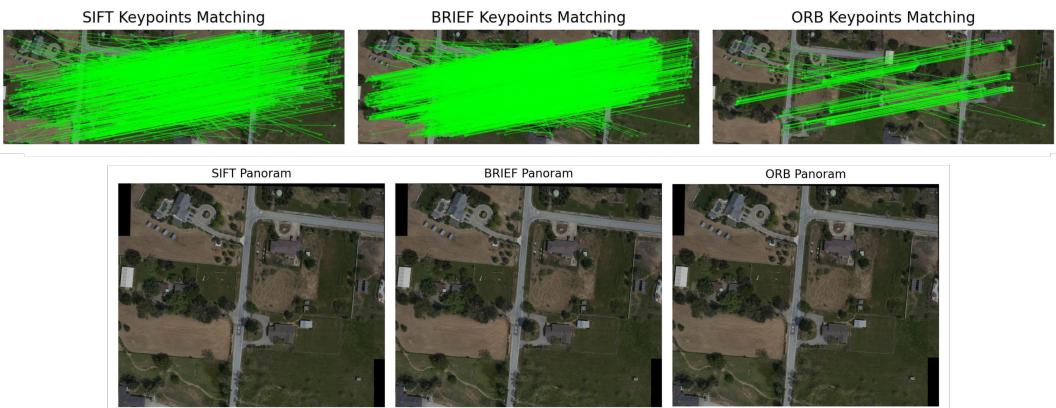


Figura 11. Foto3AB - Threshold = 0.60

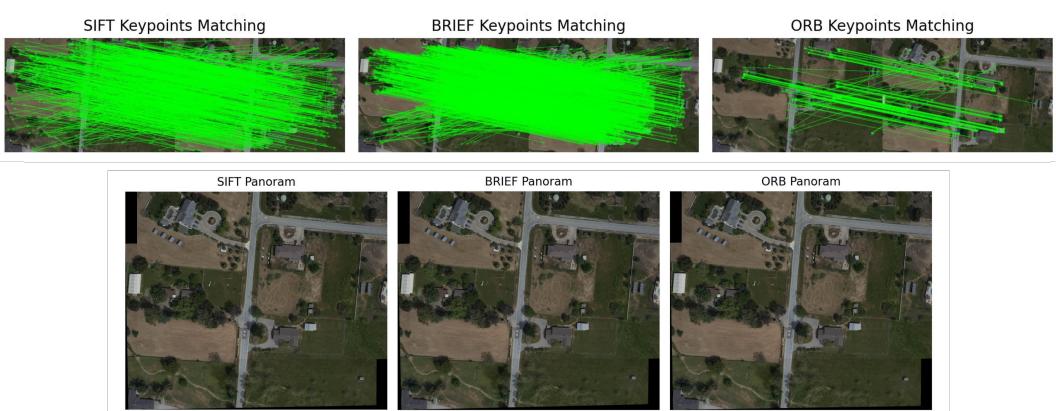


Figura 12. Foto3BA - Threshold = 0.60

Nas figuras 11 e 12, testei passar as imagens 3A e 3B em ordens diferentes para a função de criar panoramas. Com isso, pude confirmar que a rotina não é sensível à ordem das imagens. Apesar de leves diferenças no panorama resultante, o resultado é quase idêntico.

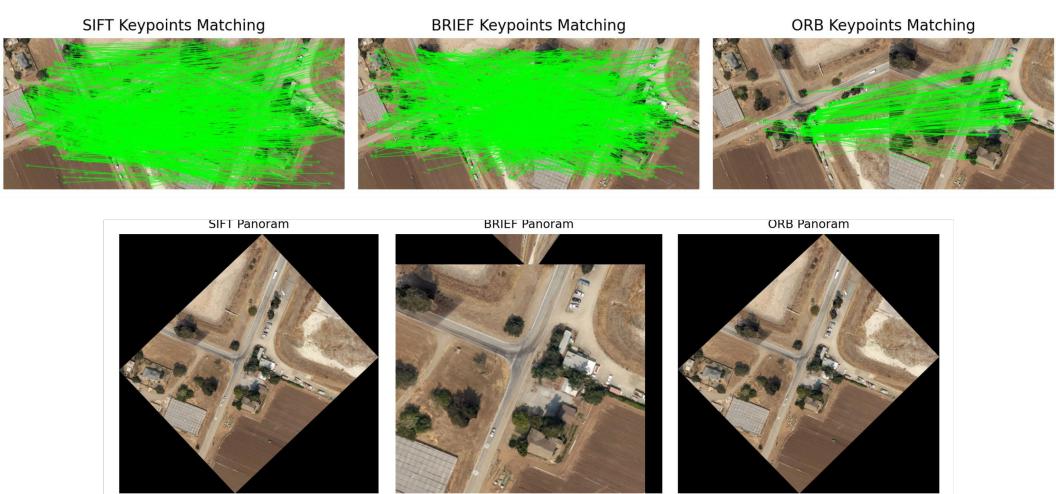


Figura 13. Foto4AB - Threshold = 0.60

A Foto 4, diferentemente das outras, possui duas imagens que se complementam ao serem rotacionadas. A técnica BRIEF não lida muito bem com esse tipo de correspondência, o resultado da aplicação dela nessa imagem dá errado, o que demonstra a importância de conhecer os detalhes de cada técnica para escolher a mais adequada para a situação prática. Inclusive, o ORB (Oriented FAST and Rotated BRIEF) foi desenvolvido para lidar com esse tipo de caso, aplicando o BRIEF junto com outras transformações para manejar melhor imagens rotacionadas.

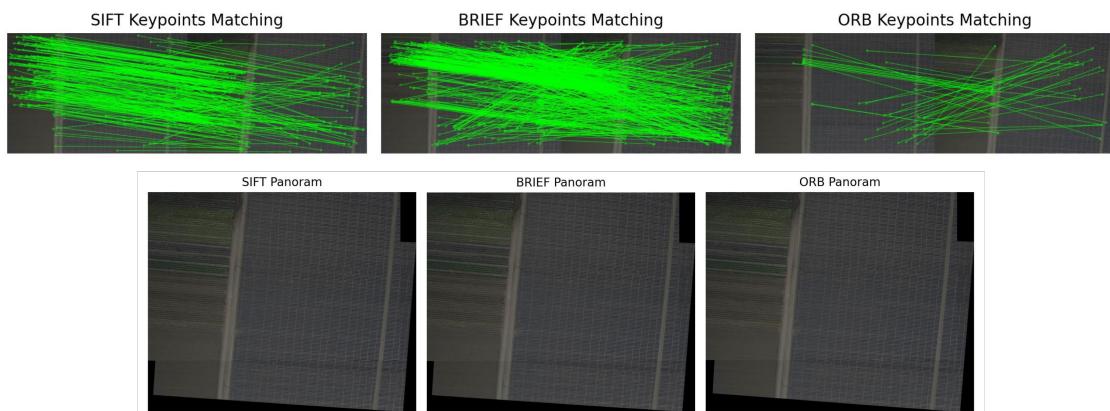


Figura 14. Foto5AB - Threshold = 0.60

Por fim, a Foto 5 apresenta padrões bem repetitivos, o que inicialmente dificultou para o SIFT encontrar correspondências que permitissem construir a imagem panorâmica corretamente. No entanto, ao alterar um de seus parâmetros, ajustando o ‘contrastThreshold’ para 0.01, o problema foi resolvido. O SIFT é uma técnica com muitos parâmetros, e dependendo da tarefa, é interessante ajustá-los. Ao rodar todos os outros testes com esse parâmetro alterado, não houve impacto nos resultados, o que demonstra mais uma vez o quanto poderosos e robustos esses algoritmos de registro podem ser.

3. Conclusão

Em conclusão, esse trabalho tratou da aplicação de técnicas de interpolação para transformações geométricas em imagens e da construção de imagens panorâmicas utilizando pontos de interesse.

Na primeira parte, foi explorado diversas técnicas de interpolação, incluindo interpolação por vizinho mais próximo, bilinear, bicúbica e Lagrangeana. Cada método foi detalhado, abordando suas vantagens, desvantagens e implementações específicas. A interpolação por vizinho mais próximo é simples e rápida, mas pode resultar em imagens de baixa qualidade devido à falta de suavização. A interpolação bilinear oferece uma melhoria significativa na suavidade das imagens interpoladas, enquanto a interpolação bicúbica proporciona uma suavização ainda maior, ideal para transformações que exigem alta precisão. A interpolação Lagrangeana, embora complexa, oferece uma abordagem polinomial que pode ser ajustada para obter resultados muito precisos, tal interpolação apresentou bons resultados apesar de não ser presente nas bibliotecas de processamento de imagens usuais ela poderia muito bem ser usada na prática.

Na segunda parte, implementamos a técnica de registro de imagens para a criação de panorâmicas. Utilizamos descritores locais, como SIFT, BRIEF e ORB, para detectar e descrever pontos de interesse nas imagens. Em seguida, empregamos a técnica RANSAC para calcular a matriz de homografia e alinhar as imagens. A função `match_images` foi detalhada, explicando cada passo desde a detecção de keypoints até a fusão final das imagens em uma panorâmica contínua. Para testar o funcionamento do código e comparar os resultados de cada técnica, plotei tanto as correspondências entre os keypoints quanto a imagem panorâmica final, exibindo essas comparações lado a lado para cada técnica.

Ao final, verificamos que cada técnica de interpolação e descritor de pontos de interesse tem suas próprias vantagens e desvantagens, dependendo do contexto e dos requisitos específicos da aplicação. Através deste trabalho, foi possível entender melhor os fundamentos teóricos e práticos do processamento digital de imagens, bem como a importância da escolha adequada das técnicas para atingir resultados ótimos em diferentes cenários.

Referências

- H. Pedrini, W. S. (2007). *Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações*. Thomson Learning.