

Introdução ao Processamento de Imagem Digital

Trabalho 4

RAYSA MASSON BENATTI

RA 176483

I. ESPECIFICAÇÃO DO PROBLEMA

O presente relatório apresenta a descrição das conclusões e estratégias adotadas para a resolução do Trabalho 4 da disciplina Introdução ao Processamento de Imagem Digital (MC906/MO443), ministrada no primeiro semestre de 2019 na Unicamp pelo professor Hélio Pedrini.

O trabalho teve como objetivo aplicar técnicas de detecção de pontos de interesse para registrar um par de imagens e criar uma imagem panorâmica a partir das correspondências encontradas.

No presente trabalho, foram usados e comparados dois detectores de pontos de interesse e descritores de imagens - SIFT e ORB - para computar similaridades entre imagens. Para a conclusão da tarefa, foi aplicada a técnica RANSAC (*Random Sample Consensus*) para estimar a matriz de homografia e, em seguida, alinhar as imagens e criar a resultante panorâmica.

II. ENTRADA DE DADOS

Foram usados dois pares de imagens de entrada, retirados de [1], no formato JPEG (*Joint Photographic Experts Group*).



Figura 1. Primeiro par de imagens de entrada.



Figura 2. Segundo par de imagens de entrada.

Para a leitura das imagens, foi utilizado o método `imread(filename)`, que faz parte da biblioteca **OpenCV**. A imagem é lida e armazenada em uma variável sobre a qual são feitas, em seguida, as manipulações necessárias. A leitura foi feita com ativação do parâmetro `cv2.IMREAD_COLOR`, por se tratar de imagens coloridas. Para aplicação das técnicas, as imagens foram convertidas para suas versões em escala de cinza, o que foi feito com o método `cvtColor(img, cv2.COLOR_BGR2GRAY)`.

Também foi dada ao usuário a opção de selecionar o descritor a ser aplicado (SIFT ou ORB). A depender de sua escolha (feita através da entrada dos caracteres 'A' ou 'B', respectivamente), é chamada a função que implementa a técnica correspondente.

III. SAÍDA DE DADOS

Os resultados das operações são recuperados com o método `imwrite(filename, img)`, também da biblioteca OpenCV. As imagens de saída são gravadas no formato JPG. Os métodos que implementam as técnicas também retornam a matriz de homografia, a qual é impressa pelo programa.

IV. ALGORITMOS E SOLUÇÕES ADOTADAS

Cada par de imagens foi submetido a um processo de busca por correspondências entre si. Para a execução dos passos de tal processo, foram implementados dois métodos: `sift(img1, img2, gray1, gray2, lim)` - cujos parâmetros representam, respectivamente: primeira imagem; segunda imagem; primeira imagem em tons de cinza; segunda imagem em tons de cinza; limite entre 0.01 e 0.5, definido pelo usuário, que define quais pontos serão considerados para marcar correspondências - e `orb(img1, img2, gray1, gray2)`.

Ambos os métodos executam os seguintes passos:

- 1) Busca por pontos de interesse (descritores invariantes) em cada uma das imagens;
- 2) Cômputo de similaridades entre cada descritor das imagens;
- 3) Seleção das melhores correspondências;

- 4) Gravação de nova imagem que exibe as linhas de correspondência entre as imagens;
- 5) Execução da técnica RANSAC através do método `cv2.findHomography`, para encontrar a matriz de homografia;
- 6) Aplicação de uma projeção de perspectiva para alinhar as imagens, através do método `cv2.warpPerspective`;
- 7) União das imagens alinhadas, criando, assim, a imagem panorâmica resultante.

Os itens de 4 a 7 são implementados de maneira praticamente idêntica em cada um dos métodos. Há, contudo, diferenças significativas entre eles nos passos de 1 a 3.

A. SIFT

SIFT é um acrônimo para *Scale-Invariant Feature Transform*. Trata-se de um descritor de imagens invariante a mudanças de escala, algoritmo criado por David Lowe (University of British Columbia) em 2004 [2]. O algoritmo é patenteado, razão pela qual não pode ser usado livremente em aplicações comerciais e exige a instalação de um módulo adicional para implementação na linguagem Python (`opencv-contrib-python`).

A implementação foi adotada conforme descrito em [3]. Inicialmente, cria-se um objeto `sift` com o método `xfeatures2d.SIFT_create()`, o qual é utilizado para computar pontos de interesse / descritores de cada uma das imagens, com `sift.detectAndCompute`.

Em seguida, o algoritmo busca as melhores correspondências entre os pontos de interesse de cada uma das imagens. Para tal, é criado um objeto `BFMatcher` para invocar o método `knnMatch`, que recebe como parâmetros os descritores encontrados anteriormente, e retorna as *features* mais similares. Neste caso, foi passado também um parâmetro $k = 2$, para que a função retorne as duas melhores correspondências para cada descritor.

As correspondências são, então, filtradas para diminuir a probabilidade de falsos *matches*. Isso é feito adicionando-se a um vetor apenas as correspondências cujas distâncias não excedam determinado limite. Tal limite é representado pela variável `lim`, que pode variar de 0.01 a 0.5 e é escolhida pelo usuário. Quanto menor o limite, maior será o refinamento de correspondências encontradas.

B. ORB

ORB é um acrônimo para *Oriented FAST and Rotated BRIEF*. Trata-se de um descritor de imagens não patenteado - ao contrário do SIFT -, tendo sido criado por Ethan Rublee, Vincent Rabaud, Kurt Konolige e Gary R. Bradski (OpenCV Labs) como uma alternativa eficiente a outros algoritmos

[4]. Ele é descrito como uma fusão do detector de pontos de interesse FAST com o descritor BRIEF, com algumas modificações para aumentar a performance [4].

A implementação foi adotada conforme descrito em [5]. Inicialmente, cria-se um objeto `orb` com o método `ORB_create`. Os pontos de interesse / descritores de cada imagem são encontrados com `orb.detectAndCompute`, similarmente ao que é feito no algoritmo anterior.

As correspondências, por sua vez, são encontradas com o método `DescriptorMatcher_create` (`cv2.DESCRIPTOR_MATCHER_BRUTEFORCE_HAMMING`), e filtradas com a simples seleção dos 15% melhores *matches* após a aplicação de uma função de ordenação.

V. RESULTADOS E DISCUSSÕES

A. SIFT

O algoritmo SIFT foi aplicado com diferentes valores de `lim` para ambas as imagens.

Em todos os testes, valores maiores de `lim` resultaram em uma maior quantidade de correspondências entre as imagens, o que é esperado. Como exemplo, comparemos as correspondências criadas para as imagens das montanhas, para valores `lim = 0.5` e `lim = 0.05`:



Figura 3. Correspondências SIFT para as imagens de montanhas com `lim = 0.5`.



Figura 4. Correspondências SIFT para as imagens de montanhas com `lim = 0.05`.

O tempo total de execução nos dois casos foi similar (2.5 e 2.4 segundos, respectivamente).

Para as imagens da praça, valores de `lim` menores que 0.1 geraram o erro `IndexError: too many indices for array` - isto é, tais valores não permitiram encontrar correspondências

o suficiente (o algoritmo somente calcula a matriz de homografia e executa os passos seguintes se houver, no mínimo, 4 *matches*). Com $\text{lim} = 0.15$, o resultado foi:

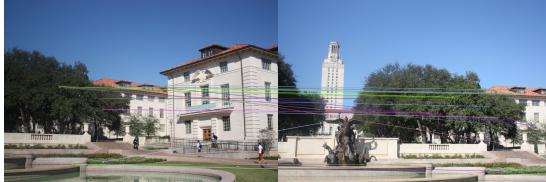


Figura 5. Correspondências SIFT para as imagens da praça com $\text{lim} = 0.15$.

Nesse caso, o tempo total de execução foi de 1.9 segundos. Para $\text{lim} = 0.5$, o tempo foi de 2.1 segundos com o seguinte resultado:



Figura 6. Correspondências SIFT para as imagens da praça com $\text{lim} = 0.5$.

Em todos os casos, as imagens panorâmicas finais produzidas - que se encontram no anexo deste relatório - são praticamente idênticas.

Assim, pode-se advogar pela adoção de valores pequenos tanto quanto possível para o limiar da distância considerada entre os pontos de interesse de diferentes imagens - desde que se encontre um valor mínimo para o qual o algoritmo funcione. No nosso caso, isso trouxe pequenos ganhos de eficiência - sugeridos por menores tempos de execução - sem comprometer o resultado.

Em todos os exemplos testados, notou-se, ainda, que o SIFT encontra correspondências inconsistentes (notáveis visualmente pelas linhas diagonais exibidas nas imagens acima), mesmo após sua seleção fina. Embora isso não tenha comprometido os resultados no caso em tela, trata-se de uma característica que pode prejudicar a eficiência da solução.

B. ORB

Para as imagens de montanhas, o ORB encontrou as seguintes correspondências, com tempo total de execução de 0.18 segundos:



Figura 7. Correspondências ORB para as imagens de montanhas.

Para as imagens da praça, o tempo de execução foi também de 0.18 segundos, encontrando tais correspondências:

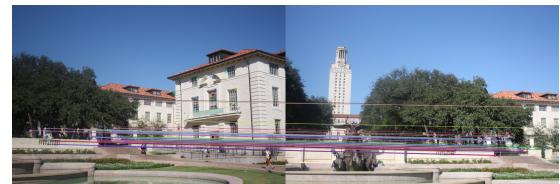


Figura 8. Correspondências ORB para as imagens da praça.

Nota-se que, nas condições do presente trabalho, o algoritmo ORB se mostrou consideravelmente mais eficiente que o SIFT, com menor tempo de execução, menos correspondências selecionadas e praticamente sem correspondências inconsistentes (diagonais). As imagens panorâmicas resultantes da aplicação do ORB também se encontram no anexo deste relatório e mostram que, para a imagem das montanhas, o resultado final foi tão bom quanto o obtido pelo SIFT. Para a imagem da praça, houve uma pequena diminuição do alinhamento na junção das imagens em relação àquela obtida pelo SIFT - porém, com um resultado ainda aceitável.

REFERÊNCIAS

- [1] Hélio Pedrini. Banco de imagens disponível em http://www.ic.unicamp.br/~helio/imagens_registro/.
- [2] OpenCV. Introduction to SIFT (Scale-Invariant Feature Transform). Disponível em <https://bit.ly/2RBhLti>.
- [3] Vagdevi Kommineni. Image Stitching Using OpenCV. Disponível em <https://bit.ly/2R19FGw>.
- [4] OpenCV. ORB (Oriented FAST and Rotated BRIEF). Disponível em <https://bit.ly/2D310kx>.
- [5] Satya Mallick. Image Alignment (Feature Based) using OpenCV (C++/Python). Disponível em <https://bit.ly/2zrgyZI>.

Anexo: imagens resultantes



Figura 1: Imagem final de montanhas com SIFT, lim = 0.5.



Figura 2: Imagem final de montanhas com SIFT, lim = 0.05.



Figura 3: Imagem final da praça com SIFT, lim = 0.5.



Figura 4: Imagem final da praça com SIFT, $\text{lim} = 0.15$.



Figura 5: Imagem final de montanhas com ORB.



Figura 6: Imagem final da praça com ORB.