Visão computacional

Lista 3 - SIFT e RANSAC

Isabelle Rodrigues Vaz de Melo

28 de agosto de 2023

1 SIFT

Para esta questão, utilizaremos o algoritmo SIFT (Scale invariant Feature transform). É apresentado como uma técnica muito útil para detecção de características em imagens, sendo essencial para procedimentos como montagem de panoramas, feature matching, homografias, detecção, entre outros.

Na última listo foi implementado o detector de Harris para detecção de cantos, entretanto vimos que em muitos casos ele pode não ser tão efetivo, realizando muitas correspondências errôneas e podendo levar ao cálculo inadequado da matriz de correspondência.

O algoritmo possui 5 passos principais:

- (1) Scale-space peak selection: Potenciais localizações para encontrar características
- (2) Localização de keypoints: Localização dos keypoints das características
- (3) Orientação: Atribuição de orientação aos keypoints
- (4) Descritor de keypoints: descrição dos keypoints como vetores de maior dimensão

Após estes passos, pode-se realizar o casamento dos keypoints. Cada etapa será brevemente explicada a seguir.

- (1): O espaço de escala de uma imagem é uma função $L(x, y, \sigma)$ que é produzida a partir da convolução de um kernel gaussiano em diferentes escalas com a imagem de entrada. O espaço de escala é separado em oitavas e o número de oitavas e escala depende do tamanho da imagem original. Assim geramos várias oitavas da imagem original. O tamanho da imagem de cada oitava é a metade da anterior. Dentro de uma oitava, as imagens são progressivamente desfocadas usando o operador Gaussian Blur:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{1}$$

  Onde G é o operador Gaussian Blur e I é uma imagem. x, y são as coordenadas de localização e σ é o parâmetro de escala, controlando a quantidade de desfoque. Quanto maior seu valor, maior o desfoque:

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

  Usamos então essas imagens borradas para gerar outro conjunto de imagens, a Diferença de Gaussianas (DoG). Essas imagens DoG auxliam na descoberta de keypoints interessantes na imagem. As DoGs são obtidas como a diferença de desfoque gaussiano de uma imagem com dois σ diferentes, dados σ e $k\sigma$. Este processo é feito para diferentes oitavas da imagem em forma de pirâmide para a aplicação dos blur Gaussianos. Os resultados com DoG são então usados para calcular aproximações laplacianas de gaussianas que são invariantes à escala. Um pixel em uma imagem é comparado com seus 8 vizinhos, bem como 9 pixels na escala seguinte e 9 pixels nas escalas anteriores. Dessa forma, são feitas um total de 26 verificações. Se for um extremo local, é um ponto-chave potencial. Significa que este ponto seria melhor representado nesta escala em específico.

- (2) Os pontos gerados na etapa anterior são excessivos. Podemos encontrar um excesso de pontos que não sejam muito significativos ou relevantes no final, assim como no detector de Harris. Para atributos de baixo contraste, verificamos suas intensidades. É feita a expansão em série de Taylor do espaço da escala para obter uma localização mais precisa dos extremos, e se a intensidade neste extremo for menor que um valor limite (0.03 pelo artigo do SIFT), este ponto é rejeitado. Já para a DoG, haverá uma resposta mais acentuada para arestas, então as arestas também precisam ser removidas.

- (3) Já sabemos a escala em que o keypoint foi detectado, ou seja, a mesma escala da imagem desfocada. Desta forma, temos invariância de escala. O próximo passo é atribuir uma orientação a cada keypoint para torná-los invariável a rotação. Selecionam-se vizinhos em torno da localização do keypoint dependendo da escala, e a magnitude e a direção do gradiente são calculadas nessa região. É criado um histograma de orientação com 36 bins cobrindo 360 graus para todos os keypoints, o histograma terá um pico em algum ponto. O pico mais alto do histograma é obtido e qualquer pico acima de 80% dele também é considerado para calcular a orientação. São criados keypoints com a mesma localização e escala, mas em direções diferentes. Isso contribui para a estabilidade da correspondência dos pontos.

- (4) Agora cada keypoint possui uma localização, escala e orientação. Nesta etapa se calcula um descritor para a região local da imagem sobre cada keypoint que seja altamente distinto e invariável quanto possível a variações como mudanças no ponto de vista e iluminação.

Para a questão, utilizei a função **SIFT()** do OpenCV em Python, e não a **sift.m** do Matlab. Todo o meu desenvolvimento foi feito em Python.

Obtive os pontos caracterísiticos e plotei em todas as imagens com suas respectivas orientações e escalas, como se pode ver na figura 1.

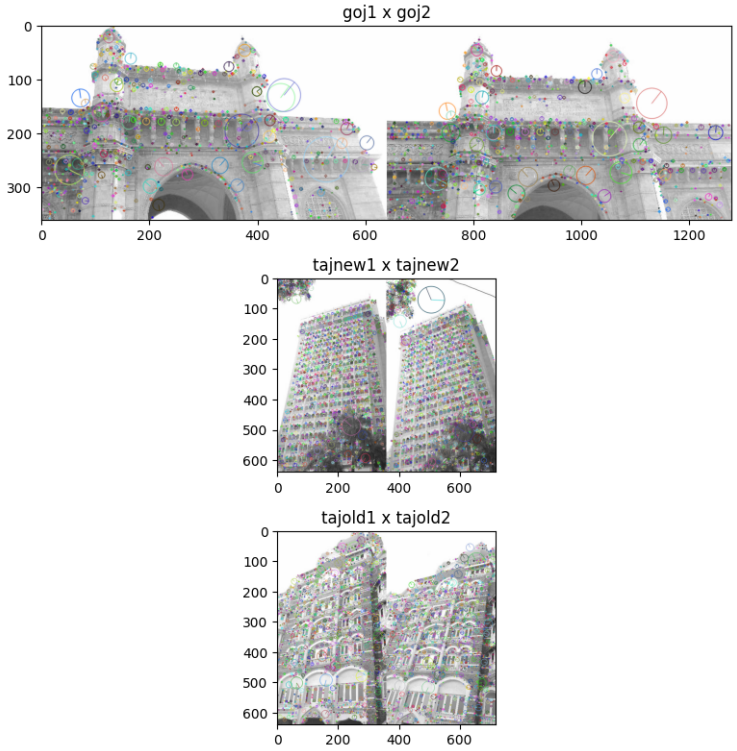

Figura 1: Imagens com keypoints encontrados pelo SIFT

Agora, queremos encontrar pares de características correspondentes, e para isso, utilizamos a menor distância Euclidiana entre os pontos da primeira imagem, comparados aos da segunda imagem. As figuras 2,3 e 4 mostram os resultados obtidos.

Como podemos ver, os resultados no geral são muito bons pois existem muitas correspondências verdadeiras. Entretanto, algumas delas são errôneas. Veja por exemplo na figura 2, as folhas da

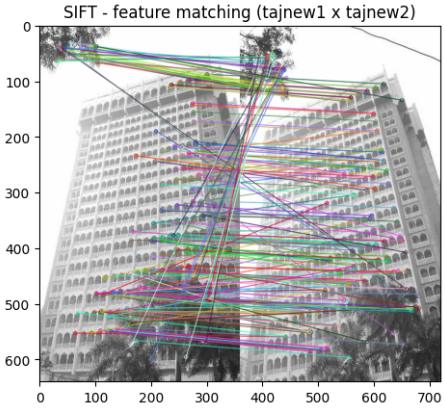

Figura 2: Correspondências SIFT (tajnew1 x tajnew2)

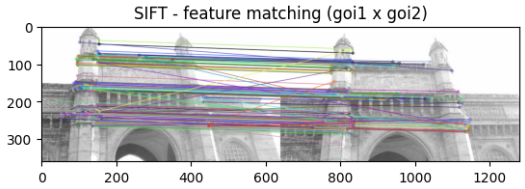

Figura 3: Correspondências SIFT (goi1 x goi2)

árvore do canto superior esquerdo possuem uma correspondência com as folhas das palmeiras do canto inferior direito, sendo estes elementos diferentes na imagem.

Queremos encontrar uma forma de nos livrar dessas possíveis correspondências erradas e filtrá-las para tentarmos conseguir encontrar a melhor homografia possível. Para limitar o número de correspondências (falsos negativos) em uma correspondência de recursos usando o algoritmo ORB com o OpenCV, usei um critério baseado na distância entre os descritores correspondentes. A ideia é que correspondências com distâncias menores seriam mais confiáveis do que aquelas com distâncias maiores. Portanto, defini um limite máximo de distância e considerar apenas as correspondências que têm uma distância menor do que esse limite. Para isso, organizei os matches da menor para a maior distância e então limitei o número de matches para um threshold que o usuário pode alterar. Por exemplo, no meu caso, fixei em 30 para todas as imagens. As figuras 4,5 e 6 mostram os resultados.

Agora de fato, apesar de poucas correspondências, quase todas as encontradas são verdadeiras. Se eu fosse utilizar todas para fazer a homografia, poderíamos dizer que teríamos um ruído muito alto no cálculo, podendo levar ao resultado de uma matriz de homografia que não faz uma correspondência adequada entre duas imagens.

2 RANSAC

O RANSAC (Random Sample Consensus), é um algoritmo amplamente utilizado em visão computacional e processamento de imagens para estimar parâmetros de modelos matemáticos a partir de um conjunto de dados contaminado por outliers.

Podemos pensar que correspondências erradas podem ser outliers no modelo.

Ele iterativamente seleciona subconjuntos aleatórios dos dados, ajusta um modelo aos pontos selecionados e, em seguida, avaliar quantos dos pontos restantes se encaixam bem com o modelo. Os pontos que se encaixam bem são chamados de "inliers,"enquanto os que não se encaixam bem são considerados "outliers."

O processo de seleção aleatória e avaliação é repetido várias vezes, e o modelo que possui o maior número de inliers é geralmente considerado como a melhor estimativa dos parâmetros do modelo. Essa abordagem é particularmente útil quando os dados estão sujeitos a ruído ou contaminação por valores fora do padrão, pois o RANSAC é robusto o suficiente para ignorar outliers e estimar o modelo corretamente.

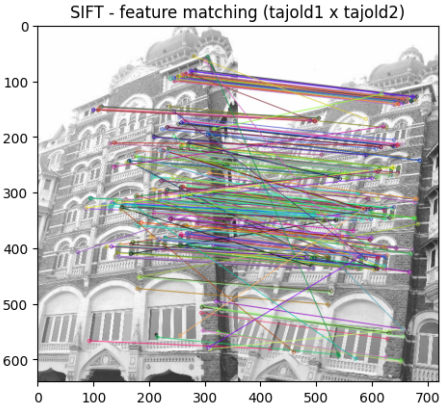

Figura 4: Correspondências SIFT (tajold1 x tajold2)

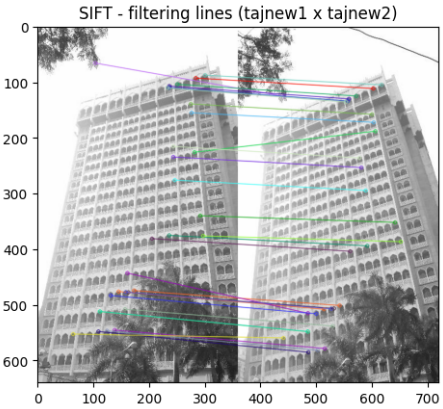

Figura 5: Correspondências SIFT limitadas (tajnew1 x tajnew2)

Seguindo os passos da lista, inicializamos N iterações, sendo que para cada uma escolho aleatoriamente 4 correspondências (pois o modelo para cálculo de Homografia precisa de no mínimo 4 pontos para que tenha solução matricial). Obtemos a homografia H_i entre as imagens com base nas correspondências selecionadas, e aí entra o critério na determinação do seu desempenho, para decidirmos o que será inlier ou não. O desempenho da homografia pode ser calculado em função do erro de projeção:

$$E_k = d(X_k, H_i X'_k) + d(X'_k, H_i^{-1} X_k) \tag{3}$$

Para $d(.,.)$ sendo a distância Euclidiana entre dois pontos. Feito isto, o critério que utilizei foi que se o erro for menor que um threshold (também definido artesanalmente pelo usuário), então eu adiciono o match como um inlier. Preciso checar isso subsequentemente para ir verificando se a lista de inliers corresponde à lista dos melhores inliers. Fazendo isso após cada iteração, calculamos a homografia para a lista completa dos inliers encontrados.

Perceba que é necessário escolher um threshold que seja um "meio-termo", levando em consideração os valores de erro.

Por exemplo: se a maioria dos valores de erro começam em 50, mas tenho alguns erros com 4,6, 10, se eu botar o threshold em 50, todos os valores abaixo disto serão escolhidos. Eles terão um erro baixo, mas o modelo será mais específico para os poucos dados fornecidos, e assim teremos variância mais alta. Caso eu bote o threshold em 1000, estarei sendo mais permissiva, ou seja, todos os valores com erro abaixo de 1000 entrariam como inliners, e isso poderia ocasionar em correspondências erradas (erro alto) entrando nos cálculos de homografia.

Por isso, este trabalho é mais individualizado para cada par de imagens, onde vamos testando os thresholds e os números de iterações, até achar - visualmente - uma imagem transformada que corresponda aproximadamente à imagem destino.

Os resultados do algoritmo podem ser vistos nas imagens 8, 9, e 10.

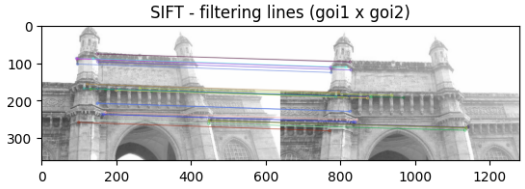

Figura 6: Correspondências SIFT limitadas (goi1 x goi2)

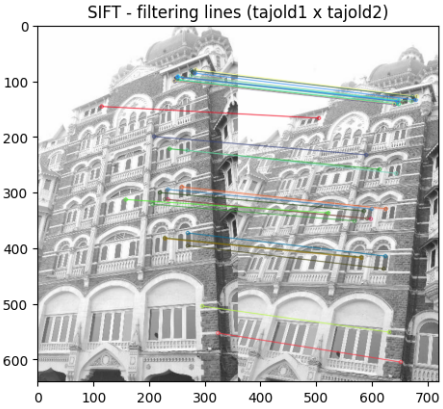

Figura 7: Correspondências SIFT limitadas (tajold1 x tajold2)

3 Informações adicionais

O código para a questão 1 pode ser encontrado em **siftexperiments.ipynb**, já para a questão 2, o código-fonte da função RANSAC é o **RANSAC.py** e os experimentos são **RANSACgoi.ipynb**, **RANSACtajnew.ipynb**, **RANSACtajold.ipynb**.

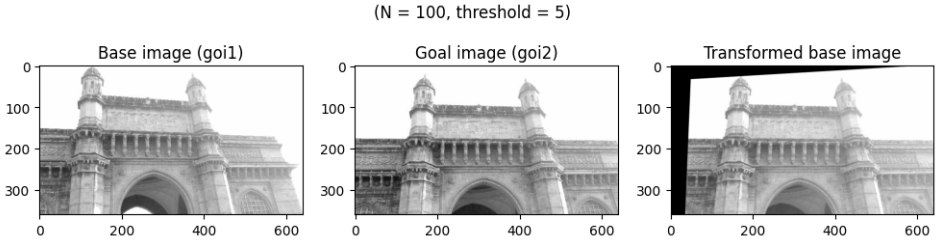

Figura 8: RANSAC (goi1 x goi2)

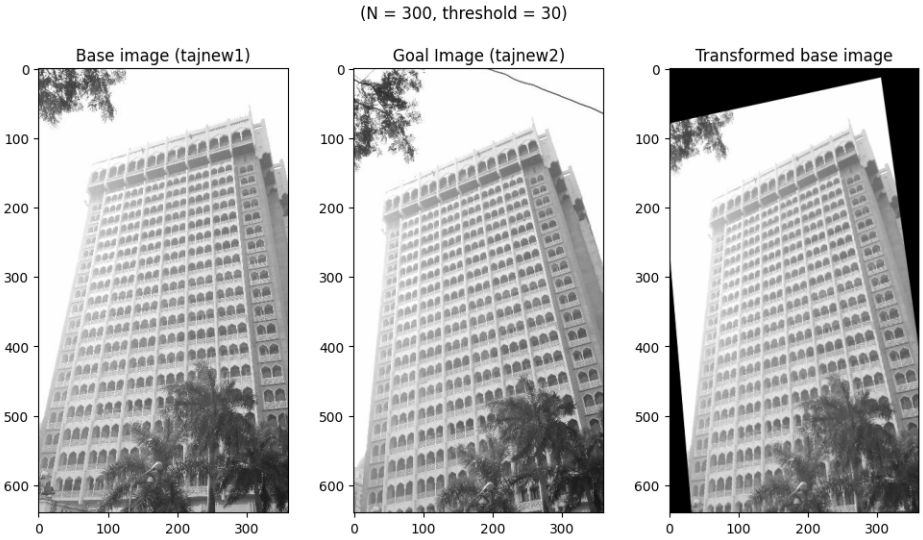

Figura 9: RANSAC (tajnew1 x tajnew2)

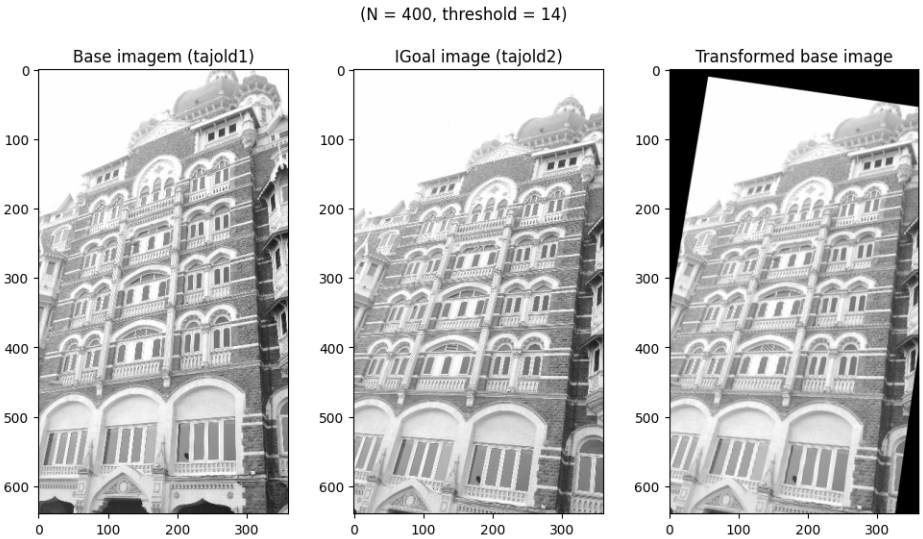

Figura 10: RANSAC (tajold1 x tajold2)