MC920 - Projeto 4 - Processamento De Imagens

Luiz Eduardo Cartolano - RA: 183012

Junho de 2019

Resumo

A correspondência de imagens é um aspecto fundamental de muitos problemas no âmbito de visão computacional, são operações cuja implementação, em sua maioria, já está feita em diversas bibliotecas de diferentes linguagens de programação, o que torna seu uso bastante simples. Neste trabalho o objetivo da aplicação das técnicas de detecção de pontos em duas imagens diferentes para alinhá-las e juntá-las, formando uma nova imagem panorâmica. Os resultados obtidos, disponíveis nas Figuras 5 e 6, foram bastante positivos, validando a utilização das técnicas para o objetivo proposto.

1 Introdução

O objetivo deste trabalho, como solicitado em 2, é a aplicação de técnicas de detecção de pontos de interesse para registrar um par de imagens e criar uma imagem panorâmica formada pela ligação entre as imagens após sua correspondência.

A correspondência de imagens é um aspecto fundamental de muitos problemas no âmbito de visão computacional, incluindo reconhecimento de objetos ou cenas, resolução de estruturas 3D a partir de várias imagens, correspondência estéreo e rastreamento de movimento. Como será explicado nas próximas seções, os algoritmos usados aplicam a seguinte sequência de técnicas: Detecção extrema do espaço de escala , localização de pontos chaves, atribuição de orientação e descrição dos pontos chaves.

Este relatório encontra-se organizado da seguinte forma: a Seção 2 apresenta a maneira como o trabalho foi implementado a fim de solucionar o problema do enunciado. A Seção 3 apresenta os resultados obtidos pela implementação feita para determinadas imagens de entrada. As discussões a respeito dos resultados encontrados estão dispostas na Seção 4, e as conclusões na Seção 5. As imagens, tabelas e equações apresentadas encontram-se nos Anexos, ao final do relatório.

2 Implementação

2.1 Uso do Código

O arquivo com os códigos implementados está nomeado como *proj4.ipynb*, e é um *notebook* do *Jupyter* [1]. Desse modo, para executá-lo, é preciso possuir o *Jupyter*, além da versão 3.7 do *Python* e as bibliotecas que foram usadas no desenvolvimento, sendo elas: *Opencv*, *Matplotlib* e *Numpy*. Além disso, é preciso possuir uma pasta *inputs*, com os arquivos de entrada, e uma *outputs*, na qual serão salvas as imagens geradas pelo código.

Para executar o código basta executar, célula a célula, o *notebook* fornecido. Vale lembrar que é essencial que as células sejam executas na ordem em que aparecem e em algum momento será solicitado o nome dos arquivos contendo as partes "A"e "B"da imagem de entrada.

2.2 Detalhes da Implementação

Nesta seção serão comentadas cada uma das operações morfológicas que foram aplicadas, explicando cada uma delas com mais detalhes. As funções do *OpenCV* que foram usadas podem ser vistas com maior detalhes em 1. É importante ressaltar, que para o funcionamento adequado das bibliotecas não pode ser

[1] https://jupyter-notebook.readthedocs.io/en/stable/

usada a versão mais nova do *OpenCV*, pois os algoritmos dos descritores não estavam implementados nestas.

2.2.1 Conversão para escala de cinza

Para poder aplicar as transformações necessárias a fim de encontrar as similaridades nas imagens e, por fim, criar uma imagem panorâmica a partir das duas imagens de entrada, foi preciso, em um primeiro momento, transformar ambas as imagens para a escala de cinzas. Para tal, fez-se uso da função *cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)*. Nela, são passados como parâmetros a imagem e a escala de cores para a qual ela será convertida, no caso desse trabalho, de colorida para cinza.

2.2.2 Descritores de Imagem

Descritores de imagem são descrições das características visuais do conteúdo em imagens, vídeos ou algoritmos ou aplicativos que produzem tais descrições. Eles descrevem características elementares, como a forma, a cor, a textura ou o movimento, entre outros. Existem uma série de algoritmos que são constantemente usados para tais operações, neste trabalho fez-se uso do *SIFT - Scale-Invariant Feature Transform* e do *SURF - Speeded-Up Robust Features*, que serão explicados em mais detalhes a seguir.

- *SIFT*
  O algoritmo *SIFT* é usado para detecção e descrição de *features* em técnicas de processamento de imagens, ele foi publicado por David Lowe em 1994, e suas aplicações incluem reconhecimento de objetos, mapeamento robótico e navegação, costura de imagens, modelagem 3D, reconhecimento de gestos, rastreamento de vídeo, identificação individual da vida selvagem e movimentação de correspondência.

  Os pontos chave do *SIFT* são primeiramente extraídos de um conjunto de imagens de referência e armazenados em um banco de dados. Um objeto é reconhecido em uma nova imagem comparando individualmente cada *feature* da nova imagem com esse banco de dados e encontrando as *features* correspondentes à distância euclidiana de seus vetores de *features*. A determinação de *clusters* consistentes é executada rapidamente usando uma implementação de tabela de hash eficiente da transformação de *Hough* generalizada. Cada *cluster* de 3 ou mais *features* que concordam com um objeto e sua pose é então sujeito a uma verificação mais detalhada do modelo e, posteriormente, os *outliers* são descartados. Finalmente, a probabilidade de um determinado conjunto de *features* indicar a presença de um objeto é computada, dada a precisão do ajuste e o número provável de correspondências falsas. Correspondências de objetos que passam por todos esses testes podem ser identificadas como corretas com alta confiança.

  O algoritmo *SIFT* está implementado na biblioteca do *OpenCV*, portanto, sua utilização é feita de maneira bem fácil. O primeiro passo para utilização da técnica é instânciar um objeto *SIFT*, o que pode ser feito a partir da função *sift=cv2.xfeatures2d.SIFT_create()*.

  A partir do objeto criado, é possível encontrar todos os pontos chaves e descritores de uma imagem usando a função *sift.detectAndCompute(image,None)*. Ou seja, a função recebe como parâmetro uma imagem e devolve duas listas, uma com os pontos chaves da imagem e outra com os descritores encontrados. Os pontos chaves, por sua vez, podem ser desenhados na imagem com a função *cv2.drawKeypoints(img_in, keypoints, img_out)*.

- *SURF*
  Na visão computacional, o *SURF* é um detector e descritor de *features*. Ele pode ser usado para tarefas como reconhecimento de objetos, registro de imagens, classificação ou reconstrução 3D. É parcialmente inspirado no *SIFT*. No entanto, sua versão padrão do é várias vezes mais rápida que o *SIFT* e, de acordo com seus autores, mais robusta contra diferentes transformações de imagem.

  O seu funcionamento é baseado nos mesmos princípios e etapas do *SIFT*, que podem ser vistos com detalhes na Seção 2.2.2, mas os detalhes em cada etapa são diferentes. O algoritmo possui três partes principais: detecção de pontos de interesse, descrição e correspondência de vizinhança local.

O *SURF* também está implementado na biblioteca *OpenCV* e seu funcionamento é bem parecido com o visto para outros descritores de imagem. Primeiro, é preciso instânciar um objeto do descritor, o que pode ser feito através da função *surf=cv2.xfeatures2d.SURF_ create()*.

A partir do objeto criado é possível encontrar todos os pontos chaves e descritores da imagem a partir da função *surf.detectAndCompute(img, None)*, que recebe como parâmetro a imagem a ser analisada e retorna, em duas listas separadas, os pontos chaves e descritores que estão sendo buscados. Os pontos chaves, mais uma vez, podem ser desenhados na imagem com a função *cv2.drawKeypoints(img_ in, keypoints, img_ out)*. É importante ressaltar que o parâmetro *keypoints* é uma lista com os pontos chaves.

2.2.3 Similaridades entre Descritores

Após calcular os descritores das duas imagens é preciso comparar ambos para analisar a similaridade entre eles. Neste trabalho usou-se a técnica *Brute-Force Matcher*, disponibilizada na biblioteca do *OpenCV*. Ele pega o descritor de uma *feature* no primeiro conjunto e é combinado com todos as outras *features* no segundo conjunto usando algum cálculo de distância. E o mais próximo é retornado.

Seu uso é feito de modo bem simples. Primeiro, é preciso instânciar um objeto do *matcher*, o que pode ser feito através da função *bf=cv2.BFMatcher()*. E então, basta realizar uma chamada a *bf.knnMatch(des_ a,des_ b, k=2)*, esta, recebe como parâmetro a lista com os descritores de ambas as imagens e retorna uma lista de objetos *DMatch*, cada objeto contém quatro atributos: distância (é a distância entre descritores - quanto menor melhor), um *trainIdx* (o index do descritor na lista de descritores de treino), um *queryIdx* (que é o index do descritor na *query* de descritores) e um *imgIdx* (que é o index da imagem de treino).

A partir da lista de objetos retornada pela função, é preciso encontrar os melhores *matches* existentes entre as duas imagens, para isso, precisamos encontrar as menores distâncias existentes. Portanto, para encontrar as melhores correspondências, percorreu-se ambas as listas e escolheu-se como boas correspondências todas aquelas na qual a distância em um dos descritores fosse menor que um limiar (pré-definido) vezes o valor da distância correspondente na outra lista.

2.2.4 Técnica RANSAC

Após encontrar todas as "boas correspondências", fez-se uso das técnica *RANSAC* para, a partir dos pontos e da distância das correspondências para encontrar a matriz de homografia para o conjunto de imagens analisadas.

O *RANSAC* é um método iterativo para estimar parâmetros de um modelo matemático a partir de um conjunto de dados observados que contém *outliers*, quando *outliers* não devem ter influência sobre os valores das estimativas. Portanto, ele também pode ser interpretado como um método de detecção atípica. Basicamente, ele é uma técnica de aprendizado para estimar parâmetros de um modelo por amostragem aleatória de dados observados.

No campo da visão computacional, quaisquer duas imagens da mesma superfície planar no espaço são relacionadas por uma matriz de homografia. Isso tem muitas aplicações práticas, como retificação de imagem, registro de imagem ou computação do movimento da câmera - rotação e translação - entre duas imagens.

Para aplicar a técnica neste trabalho, usou-se a função implementada na biblioteca do *OpenCV*, basta usar a função *cv2.findHomography(lst_ pts, dst_ pts, cv2.RANSAC, 5.0)*. O retorno da chamada em questão é a matriz de homografia e uma máscara com os *matches*.

2.2.5 Técnica de Perspectiva

Um imagem panorâmica é qualquer visão de ângulo amplo ou representação de um espaço físico, seja em pintura, desenho, fotografia, filme, imagens sísmicas ou um modelo tridimensional.

Neste trabalho, para aplicar a técnica de perspectiva, usou-se a função *cv2.warpPerspective(img, matriz_ homografia, shape_ imagem)*. Para obter o efeito apresentado no enunciado do trabalho (vide 2), passou-se a imagem "A", a matriz de homografia obtida e, como shape, a soma das larguras da imagens "A"e "B"e a altura da imagem "B".

A equação de transformação usada pode ser vista na Equação 3. Após aplicar a perspectiva para o lado direito da imagem foi preciso juntá-la aos pontos faltantes localizados no lado esquerdo, obtendo, por fim, uma imagem panorâmica.

2.2.6 Retas entre os pares correspondentes

Para exemplificar todas os pontos de correspondência entre as duas imagens, para isso usou-se, novamente, uma função disponibilizada pela biblioteca do *OpenCV*. Para tal, primeiro é preciso definir os parâmetros das linhas que serão desenhadas, o que pode ser feito através da seguinte linha de código: *draw_params = dict(matchColor=(0,255,0), singlePointColor = None, matchesMask=mask.ravel().tolist(), flags=2)*. Uma vez definidos os parâmetros basta usar a função *cv2.drawMatches(img_a, kp_a, img_b, kp_img, selec_pts, None, **draw_params)*.

2.3 Restrições Da Implementação

Os resultados obtidos para as implementações adotadas foram bem satisfatórios, como será visto nas próximas seções. Contudo, a solução possui algumas restrições.

Uma importante restrição está no modo como o código foi arquitetado. A maneira como são passados os *inputs* é bem limitada e dificulta a execução do código para diferentes arquivos. Outro limitante é o modo como é definido o *threshold* usado para calcular os "bons *matches*"entre os pontos chaves das duas imagens, já que, além de pré-definido ele possui o mesmo valor para qualquer técnica aplicada.

Além disso, outra restrição encontrada está na escolha do valor de *threshold* usado na Seção 2.2.3, ele foi escolhido sem muito embasamento teórico e, portanto, pode não trazer bons resultados para qualquer caso genérico. Por exemplo, quando testado com a Figura 1b, o método *SIFT* não foi capaz de encontrar *matches* bons o suficiente para que fosse possível calcular a matriz de homografia e, por conseguinte, aplicar a técnica de perspectiva.

3 Resultados

As imagens resultantes após cada um dos passos executados no projeto podem ser encontradas em sua totalidade nos Anexos. As imagens originais podem ser vistas na Figura 1a.

Após converter as imagens para a escala de cinza foram obtidas as Figura 2. Após aplicar as técnicas para computar os descritores e pontos chaves foram obtidas, para a técnica *SIFT* a Figura 3 e a Figura 4 para a técnica *SURF*. Para o primeiro método encontrou-se 1280 pontos chaves para a imagem "A"e 1457 para a imagem "B". Já para a segunda técnica, encontrou-se, respectivamente, 1551 e 1605, pontos chaves. Após aplicar as técnicas para computar similaridades e usando um *threshold* de 0.5, encontrou-se 306 "bons *matches*"para o método *SIFT* e 73 para o *SURF*.

Após aplicar a técnica *RANSAC*, foi possível obter as matrizes de homografia para ambos os métodos. No método *SIFT* obteve-se a Matriz 1. Já para o método *SURF*, a homografia é dada pela Matriz 2.

Por fim, as imagens panorâmicas obtidas para ambas as técnicas estão disponíveis na Figura 6. E as imagens dispostas lado a lado com as linhas indicando as correspondências existentes estão apresentadas na Figura 5.

4 Discussão

A imagem original precisa ser convertida para escala de cinzas, isto é, os elementos da matriz de entrada precisam ser agrupados em níveis de cinza, pois os algoritmos de detecção de *features* usados funcionam apenas com imagens descritas em níveis de cinza. Como podemos observar, a única característica que se modifica nas imagens (vide Figuras 1a e 2) é a coloração delas, nenhuma outra característica é alterada ou perdida, logo a detecção não será afetada.

Dos resultados obtidos, os que fornecem mais *insights* para discussão são as imagens nas quais foram demarcados os pontos chaves das imagens para as técnicas de *SIFT* e *SURF*, vide Figuras 3 e 4. É visível analisando as imagens que o método *SURF* encontrou mais pontos chaves (e descritores) que o método *SIFT*, o que é um resultado esperado na literatura, já que aquele é mais robusto com relação a

possíveis variações ou transformações na imagem. Contudo, um resultado não esperado foi o tempo de processamento gasto para ambos os métodos, já que o *SIFT* foi cerca de duas vezes mais rápido, quando na verdade esperávamos o contrário. O motivo dessa diferença pode estar relacionado a maneira como foi feita a implementação das técnicas na biblioteca do *OpenCV*.

Outro detalhe interessante é quando avaliamos a "qualidade"dos "matches"de similaridade para cada uma das técnicas. É visível que, usando um mesmo *threshold* a quantidade de pontos selecionados para encontrar a matriz de homografia é significantemente diferente. Com um *threshold* que escolhe todas as ocorrências na qual a distância na imagem "A"é menor que $threshold \cdot distancianaimagem$"$B$". Para os descritores escolhidos com a técnica *SIFT* foram selecionados 306 pontos chaves, enquanto que para o *SURF* foram apenas 73. Tal situação nos leva a crer que, muito provavelmente, a técnica *SIFT* possui maior acurácia nos pontos que encontra, novamente, esse é um resultado esperado pela literatura quando se compara ambas as técnicas.

Após encontrar todos os pontos de similaridade entre as imagens foi preciso encontrar a matriz de homografia, que é a matriz que relaciona duas imagens diferentes que se encontram no mesmo plano. Ao compararmos as matrizes encontradas para ambos os métodos (vide Matrizes 1 e 2), é perceptível que ambas são bem parecidas, especialmente as duas primeiras linhas, que se diferenciam apenas em casas decimais que podem até ser ditas insignificantes. A maior diferença entre as matrizes está na última linha, contudo, os valores são extremamente pequenos, na casa de 10^{-5} ou 10^{-6}, logo, é possível dizer que as matrizes encontradas são praticamente a mesma, tendo o mesmo efeito quando aplicadas a função que irá realizar a perspectiva das imagens.

Os últimos dois resultados que são passíveis de comentários e que evidenciam, mais uma vez, as diferenças na aplicação de ambas as técnicas são as figuras panorâmicas e as com as linhas conectando pontos semelhantes demarcadas. A perspectiva foi feita através de uma função do *OpenCV* que utiliza a Equação 3 para realizar a transformação. Analisando a equação é possível perceber que o resultado obtido (vide Figura 6) foi condizente com o esperado e a semelhança entre as imagens é justificada pela similaridade das matrizes de homografia.

Por fim, é possível comparar as imagens com os pontos similares conectados por uma reta (vide Figura 5), a técnica *SIFT* foi capaz de conectar um número muito maior de pontos chaves entre as imagens "A"e "B", o que reforça um aspecto já comentado anteriormente, que é a maior acurácia desta quando comparada com outras técnicas usadas nesse trabalho.

5 Conclusão

Após a finalização do trabalho, é possível concluir que a aplicação de técnicas de detecção de pontos de interesse em duas imagens apresenta resultados bastante satisfatórios e possuem uma implementação bastante simples. Portanto, são um bom caminho para atingir o objetivo proposto, que é a criação de uma imagem panorâmica a partir do alinhamento e união de duas imagens distintas.

Dada as restrições comentadas ao longo deste relatório é possível perceber que, para iterações futuras, alguns pontos tem a possibilidade de implementações mais complexas que trariam resultados ainda mais expressivos do que os já obtidos. Uma das melhorias que podem ser implementadas, por exemplo, é o uso de técnicas mais aprimoradas, como um aprendizado de máquina ou algo semelhante, na obtenção dos *thresholds* usados para classificação dos *matches* que foram considerados como bons. Além disso, podem ser usadas outras técnicas para o encontro de pontos chaves e descritores, como a *BRIEF - Robust Independent Elementary Features* ou a *ORB - Oriented FAST, Rotated BRIEF*.

Referências

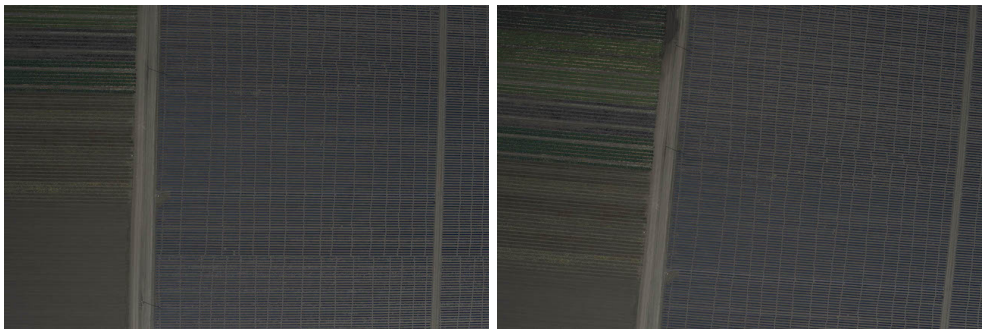
[1] Funções sift e surf. Disponível em: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html.

[2] PEDRINI Helio. Projeto 4. Disponível em: http://www.ic.unicamp.br/~helio/disciplinas/MC920/trabalho4.pdf.

[3] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

ANEXOS

(a) Imagem usada para os resultados mostrados neste relatório.



(b) Imagem que originou problemas para o método SIFT.

Figura (1) Imagens de entrada.

$$matriz\_sift = \begin{bmatrix} 1.15472093e+00 & 1.09376533e+00 & -2.19464853e+02 \\ -1.09520251e+00 & 1.15488333e+00 & 2.13150512e+02 \\ -2.81408432e-06 & 1.02945162e-06 & 1.00000000e+00 \end{bmatrix} \quad (1)$$

(1) Matriz de Homografia para a técnica *SIFT*.

$$matriz\_surf = \begin{bmatrix} 1.15015922e+00 & 1.10521490e+00 & -2.20175797e+02 \\ -1.10102046e+00 & 1.15705785e+00 & 2.14567966e+02 \\ -3.29758618e-05 & 4.06997512e-05 & 1.00000000e+00 \end{bmatrix} \quad (2)$$

(2) Matriz de Homografia para a técnica *SURF*.

$$dst(x,y) = src(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}) \quad (3)$$

(3) Equação de tranformação de perspectiva.

Figura (2) Imagens de entrada convertidas para escala de cinza.

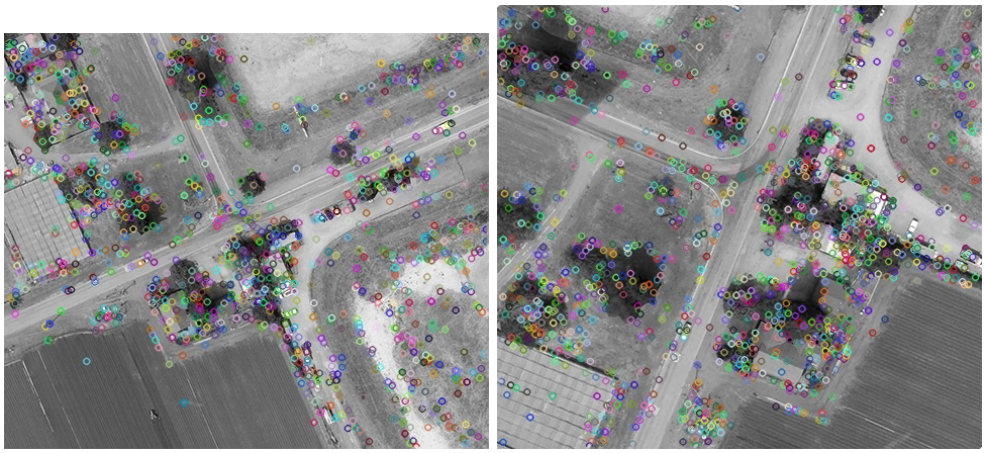


Figura (3) Imagens de entrada após demarcação dos descritores usando o método *SIFT* .

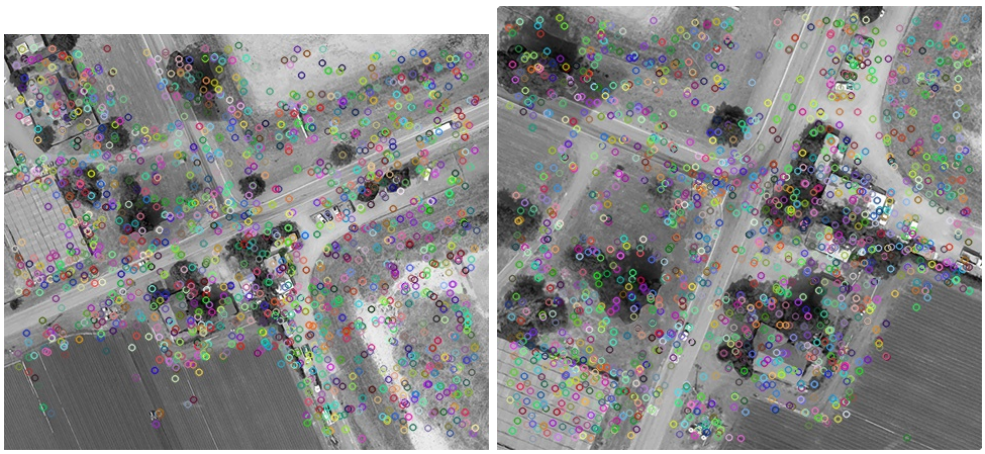


Figura (4) Imagens de entrada após demarcação dos descritores usando o método *SURF* .

Figura (5) Imagens de entrada após demarcadas as linhas unindo pontos semelhantes das duas imagens através das técnicas *SIFT* e *SURF*

.

Figura (6) Imagens de entrada após serem serem alinhadas e unidas para criar uma imagem panorâmica através das técnicas *SIFT* e *SURF*

.