UNIVERSIDADE D
COIMBRA

MESTRADO EM ENGENHARIA E DE COMPUTADORES

Relatório de Visão por Computador

Feature Detection and Matching

Duarte de Sousa Cruz, 2017264087

Universidade de Coimbra
06 de Abril de 2022

1 Introdução

Neste trabalho foi-nos proposto implementar um *pipeline* de deteção de *features* e *matching* das mesmas. A *feature* que vamos detetar nas imagens são os *Harris Corners* e para fazer os descritores vamos usar dois tipos: *SIMPLE* e *MOPS*.

Na fase final da pipeline vamos usar duas métricas, *SSD* e *Ratio*, para fazer o matching dos descritores obtidos.

Todo o código deste trabalho vai estar presente em anexo, no fim do relatório.

2 Feature Detection

Este vai ser o primeiro passo do *pipeline*, onde através do método de deteção do *Harris Corner* vamos identificar os pontos de interesse numa determinada imagem. Para tal vão ser desenvolvidas duas funções, a *HarrisCorner* e a *KeypointsDetection*.

2.1 Harris Corner

Esta função tem como objetivo descobrir as coordenadas *(x,y)* dos cantos de uma imagem.

A função vai receber como entrada uma imagem em escala cinza, um treshold, escala das derivadas gaussianas, escala das integradas gaussianas e o tamanho da região de *NMS*. Começámos por calcular as imagens de gradiente em x e em y da imagem, filtrada por um filtro gaussiano com $\sigma = 1$. De forma a obter melhores resultados, em vez de fazermos as derivadas da imagem com o operador de *Sobel* diretamente, optamos por fazer a derivada em x e em y do filtro gaussiano, e depois convolver estas derivadas com a imagem de forma a obter I_x e I_y.

De seguida, já conseguimos construir a matriz M, definida por:

$$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{1}$$

sendo I_x a imagem de gradiente em x, e I_y a imagem de gradiente em y. Os elementos I_x^2, I_y^2 e $I_x I_y$.

O passo seguinte é calcular a força de cada ponto, ou seja, quanto maior o valor da força do canto encontrado, mais provável é este de ser um verdadeiro positivo. Esta força vai ser calculada com a seguinte fórmula, usando os gradientes anteriormente obtidos.

$$c(M) = det(M) - k \cdot trace(M) = I_x^2 I_y^2 - (I_x I_y)^2, \; k \in [0.04 \; 0.06] \tag{2}$$

Para filtrar esses valores, de modo a prevenir guardar falsos cantos, temos que usar um *treshold* que vai entrar na função. Todos os valores abaixo deste vão ser eliminados e não vão ser guardados na matriz c. Por último aplicámos uma supressão não máxima à matriz c para podermos ficar apenas com os valores ótimos para indicar cantos. A supressão não máxima consiste em comparar o valor da magnitude do gradiente do píxel atual com os pixeis que estão na direção positiva e negativa do gradiente. Se a magnitude do gradiente do píxel atual for maior que os dois pixeis nas direções referidas então significa que o píxel atual é o píxel mais intenso e deve ser preservado. Se não o valor dele deve ser suprimido.

A função vai retornar as coordenadas de todos os pontos do *Harris Corners*.

Figure 1: Cantos detetados na imagem bikes

Figure 2: Cantos detetados na imagem Leuven

2.2 Key Points Detection

Esta função tem como objetivo encontrar a orientação e a escala de todos os pontos retornados da função *Harris Corners*.

A função vai receber como entrada a imagem em escala cinza original e os pontos anteriormente determinados. No final a função vai devolver a *struct* recebida como parâmetro, no entanto, para além de conter a localização dos cantos, vai ainda ter a sua orientação e escala.

Começámos por calcular a *multi-scale pyramid* de sigmas, com a eq.3, que consiste em avaliar a *Laplacian-of-Gaussian* a todos os *Harris points* e ficar apenas com os pontos que a laplace é extrema. Para obter o *log kernel* usámos a função *fspecial* e aplicámos à imagem usando a convolução entre a imagem e o *kernel* obtido.

$$\sigma = 2^j * 0.25, \quad j = 1, ..., 6 \tag{3}$$

Figure 3: $\sigma = 2^1 * 0.25$

Figure 4: $\sigma = 2^2 * 0.25$

Figure 5: $\sigma = 2^3 * 0.25$

Figure 6: $\sigma = 2^4 * 0.25$

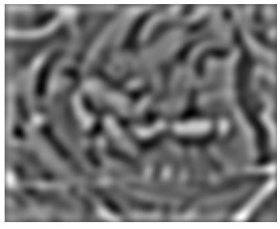

Figure 7: $\sigma = 2^5 * 0.25$

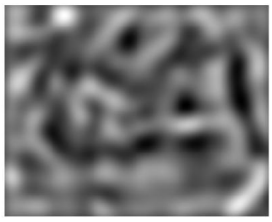

Figure 8: $\sigma = 2^6 * 0.25$

De seguida, para cada coordenada da estrutura *Pts* vamos verificar em qual das 6 imagens se encontra o seu máximo local. O valor do sigma da imagem onde se encontra esse máximo vai guardado na *struct* como o valor da escala do canto que se encontra nessa coordenada.

Se o valor do sigma fosse muito elevado, a variação da escala ia ser mínima acabando com quadrados de tamanhos semelhantes.

Para calcular a orientação dos pontos primeiro temos de suavizar a imagem com um filtro gaussiano para podermos calcular as derivadas *x, y* usando o operador *Sobel*. Sendo que a orientação vai ser:

$$\theta = atan2(I_y, I_x), \tag{4}$$

com I_y igual à derivada y da imagem e I_x igual à derivada x da imagem. Sendo apenas depois necessário guardar na estrutura a orientação correspondente a cada coordenada já existente nessa mesma estrutura *Pts*.

Figure 9: Key Points da imagem bikes

Figure 10: Key Points da imagem leuven

Figure 11: Key Points da imagem graf

3 Feature Description

Agora que identificámos os pontos de interesse, o próximo passo é arranjar um descritor, centrado em cada ponto, para a *feature*. Vão ser implementados dois descritores diferentes, *'SIMPLE' e 'S-MOPS'*.

A função vai receber como entrada a imagem em tons de cinza original, os pontos de interesse anteriormente calculados, o método desejado e o *Patch_ size*. Vai retornar os descritores para cada ponto em matriz.

3.1 Simple feature description

Neste método vamos usar uma janela quadrada *5x5*, invariante à orientação e vai estar centrada em cada ponto de interesse. Para isso temos que aumentar a imagem em metade do tamanho da janela em todos os lados, para o caso de haver pontos no limiar da imagem, usando a função *padarray*.

Para obter os descritores basta pegar nessa janela dos pontos e aplicar um *reshape* para obter um descritor de *25x25*. Cada linha desta matriz vai ser um vetor de *1x25* contendo os níveis de cinzento de cada *píxel* das janelas.

Figure 12: Patch de tamanho 5 em torno da feature

3.2 Simple MOPS feature descriptor

Neste método vamos usar uma janela quadrada *40x40*, variante à orientação e vai estar centrada em cada ponto de interesse. Temos que aumentar outra vez em metade do tamanho da janela em todos os lados, para o caso de haver pontos no limiar da imagem. De seguida temos de rodar este *patch* para a sua orientação canónica, onde 0^{o} corresponde ao versor $(1, 0)$, aponta para a direita. Portanto, vamos rodar o *patch* da *feature* por um ângulo igual a $-\theta$, sendo θ o ângulo de orientação da *feature* detetada, neste caso do canto. A matriz de transformação usada é:

$$T_{rot} = \begin{bmatrix} \cos(-\theta) & \sin(-\theta) & 0 \\ -\sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Depois vamos rodar a janela quadrada pelo ângulo de rotação obtido pela matriz anterior, usando a função *imwarp* e fazer um *imresize* à janela para a tornar num tamanho de *8x8*.

Para obter os descritores basta pegar nessa janela dos pontos e aplicar um *reshape* para obter um descritor de *64x64*. Por último, normalizamos o *patch* para ter uma média nula e uma variância unitária.

Figure 13: Patch de tamanho 40 à volta da feature

Figure 14: Patch orientado com a orientação canónica

4 Feature Matching

Agora que já detetámos e descrevemos as *features*, o próximo passo vai ser comparar dois descritores e retornar a distância entre eles. Vamos implementar duas possíveis métricas de calcular as distâncias, a *SSD (Sum of Squared Distances)* e a *Ratio*. A menor distância calculada entre descritores de imagens diferentes é a melhor correspondência.

4.1 SSD

Esta métrica calcula a soma dos quadrados da distância entre dois descritores. Quanto mais pequeno for este valor mais provável é de as duas *features* serem iguais. Para isso, usámos a função *min* do *MATLAB* que vai retornar o menor valor do vetor *SSD*.

Por último, vamos encontrar o índice do valor mínimo no segundo descritor, e caso esse valor seja mais pequeno que o *treshold* vai armazenar na matriz *Match*. A matriz *Match* vai então conter o índice da *query image*, o índice da *feature* da imagem teste e a distância entre eles.

4.2 Ratio

A diferença entre esta métrica e a *SSD* é que aqui o algoritmo vai buscar os dois melhores valores das distâncias, ou seja, as duas menores distâncias e fazer a fração entre elas.

$$RT = \frac{SSD_{best}}{SSD_{2best}} \tag{6}$$

Caso esse valor esteja abaixo de um determinado *treshold* vai ser guardado na matriz *Match* juntamente com o *índex* onde a *feature* se encontra em cada uma das imagens.

5 Show Matching

Para finalizar vamos avaliar o desempenho do algoritmo com esta função. Esta vai receber como entrada a matriz *Match*, as duas imagens em escala cinza, os descritores das duas imagens e os *keypoints* das mesmas.

Começamos por colidir as duas imagens num único **plot**, usando a função *cat* e fazemos a escala de erro para ser usado nas coordenadas x dos pontos da segunda imagem. Em seguida vamos dar *plot* dos *keypoints* das duas imagens e desenhar uma linha entre os *matches*. Por último, vamos dar *plot* de seis descritores de cada ponto, após fazermos um *resize* para uma escala de *8x8*. Só mostramos seis, pois são demasiados pontos para mostrar.

6 Resultados

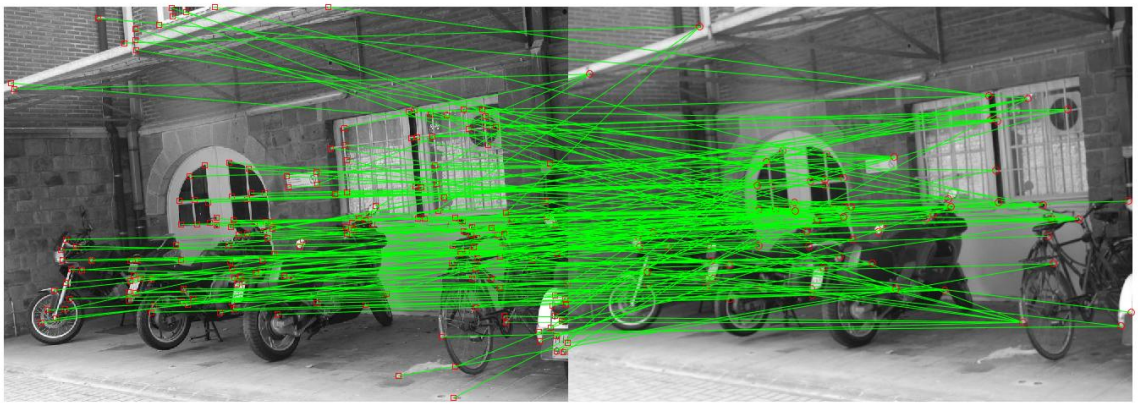

Figure 15: linhas dos *Matched Keypoints*

Figure 16: linhas dos *Matched Keypoints*

Figure 17: linhas dos *Matched Keypoints*

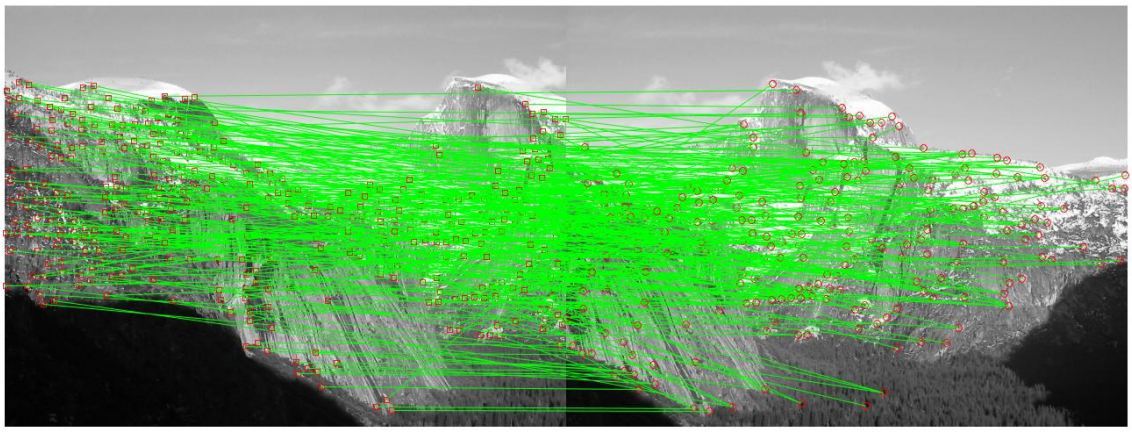

Figure 18: linhas dos *Matched Keypoints*

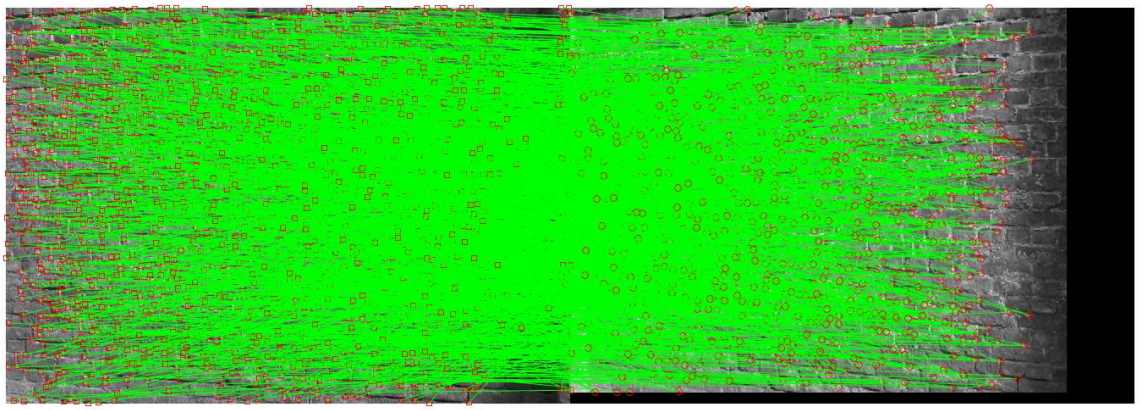

Figure 19: linhas dos *Matched Keypoints*

Observando os resultados podemos ver que para as imagens do dataset bikes, leuven e yosemite, o algoritmo tem um funcionamento expectável fazendo boas correspondências entre as *features* detetadas nas imagens. No entanto, na imagem do grafiti o comportamento já não é o desejável, isto pode ser porque na imagem do grafiti as correspondências sofrem uma transformação mais acentuada em relação à primeira. Isto pode significar que o nosso descritor *S-MOPS* não é invariante a alguns tipos de transformações como rotações. Estes resultados podem também ser verificados em relação à imagem Wall.

7 Conclusão

Devido ao diverso conjunto de imagens torna-se complicado arranjar parâmetros que deem os melhores resultados em todas as imagens, no entanto, tentamos usar parâmetros que dessem bons resultados em todas as imagens.

Para concluir, acho que consegui atingir todos os objetivos, exceto na última função que tive de passar os *keypoints* para conseguir resolver o problema. A precisão do *matching* de algumas imagens não é a melhor, nomeadamente na imagem do grafiti, onde o algoritmo tem dificuldades em fazer corresponder as *features*.