Prova 3 - Construção de um Panorama

Fábia Isabella Pires Enembreck
Universidade Tecnológica Federal do Paraná
Ponta Grossa-PR, Brasil
E-mail: fabia.isape@gmail.com

1. Introdução

A construção de panoramas é um problema bastante comum da visão computacional e pode ser descrito como sendo um problema de alinhar imagens correspondentes, no entanto é necessário identificar quais partes das imagens devem se juntar [1]. O objetivo deste trabalho é estudar uma técnica para construção de panoramas, bem como a implementação da técnica estudada para gerar um panorama a partir de um vídeo.

A técnica estudada neste trabalho corresponde a geração de um panorama a partir de descritores de características e matriz de homografia, assim são verificados os pontos correspondentes entre imagens. A técnica estudada é apresentada na Seção 2. A implementação do algoritmo é abordada na Seção 3 e a sua utilização na Seção 4. Os resultados e conclusão deste trabalho são mostrados nas Seções 5 e 6, respectivamente.

2. Construção de um Panorama

Para a construção de um panorama é preciso seguir várias etapas até chegar a um resultado final. Com isso, nesta seção serão descritas as etapas realizadas neste trabalho.

2.1. Correspondências

Inicialmente deve-se encontrar correspondências entre os *frames* do vídeo, através de seus pontos-chave ou *keypoints* e descritores de características. Para isso, neste trabalho, foi utilizado um método conhecido como *SIFT - Scale Invariant Feature Transform*. Este método basicamente gera, a partir de uma imagem, *keypoints* associados a um descritor de características, que são invariantes à escala, rotação e parcialmente invariantes a iluminação [2]. O *SIFT* possui 4 etapas principais para extração desses recursos, são elas:

1) Detecção de espaços de escala;
2) Localização de *keypoints*;
3) Atribuir orientação aos *keypoints*;
4) Descritor de características.

O primeiro passo é encontrar os espaços de escala de uma imagem, para criar variações internas de uma imagem, tornando o o método invariante a escala. Isso pode ser feito criando imagens em diferentes escalas, reduzindo a imagem original pela metade, em seguida reduz a metade novamente e assim sucessivamente. No entanto, para cada escala devem ser geradas oitavas da imagem aplicando o Filtro da Gaussiana para suavização, ou seja, dentro de cada escala temos n oitavas suavizadas progressivamente [2], [3]. Segundo descrito por [2], recomenda-se utilizar 4 oitavas e 5 variações de escala. Em seguida, calcula-se a Diferença de Gaussiana das imagens obtidas, por escala, conforme apresentado na Figura 1.

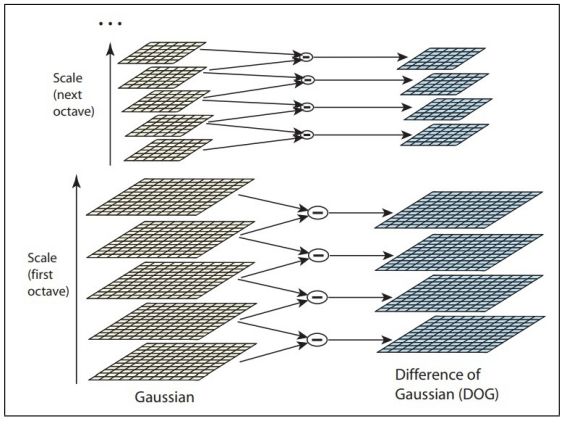

Figura 1: Espaço de escala e diferença de *Gaussianas*. Exemplo de retirado de [2]

Em seguida, os *keypoints* candidatos são identificados. Isso é feito, localizando os pontos máximos ou mínimos das imagens resultantes da Diferença de Gaussianas. Para localizar estes pontos, compara-se um ponto com seus 8 vizinhos e também com o pontos correspondentes da imagem de cima e de baixo de uma mesma escala e os vizinhos de cada ponto, com isso no total são comparados 26 pontos. Se o ponto analisado for o valor máximo ou mínimo obtido pela comparação com os 26 pontos, então considera-se que esse ponto é um *keypoint* candidato. Após obter todos os candiados, elimina-se aqueles com baixo contraste e pertencentes a borda [2].

Com os *keypoints* localizados, atribui-se uma orientação a eles, tornando o método invariante à escala. Para isso, calcula-se a magnitude e orientação dos gradientes em torno de cada *keypoint*, para descobrir a orientação mais proeminente em cada região. Um histograma para cada *keypoint* é produzido com os valores de orientação

do gradiente calculados, os 360° podem ser divididos em intervalos, como 36 intervalos de 10 em $10e$ graus para representar no histograma. Assim, o pico do histograma corresponde a orientação do *keypoint*. Vale ressaltar que, se houver algum ponto do histograma que seja acima de 80% do pico, então cria-se um novo *keypoint* no mesmo local, porém com orientação diferente [2].

Com as orientações dos *keypoints* calculadas, é possivel determinar os descritores de características de cada um deles. Para isso, em torno de cada *keypoint* detecta-se cria-se novamente um histograma com as orientações de uma área 16x16, sob regiões de amostragem $4x4$, com isso obtém-se 8 direções para cada região, obtendo assim um vetor de tamanho 128. Na Figura 2 é apresentado um exemplo deste processo, mas com uma área de tamanho $8x8$. Para um melhor desempenho deste método em situações de variação de iluminação, normaliza-se os valores do vetor [2].

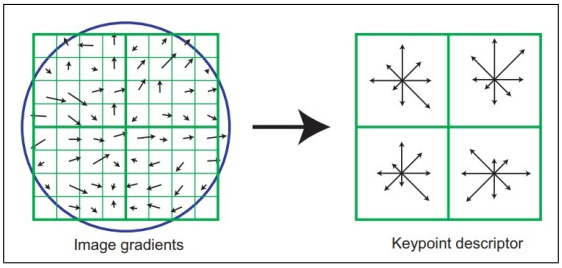

Figura 2: Descritor de *keypoint*. Exemplo de retirado de [2]

Para cada *frame* do video aplica-se o *SIFT*, compara-se os descritores de cada *frame* com os descritores de outro *frame*, utilizando o conceito de vizinho mais próximo, através da Distância Euclidiana. Assim encontram-se *keypoints* correspondentes se a distância calculada for menor do que um limiar pré-definido.

2.2. Matriz de Homografia

A geometria epipolar entre duas imagens está relacionada com a interseção de planos da imagem, como por exemplo buscar pontos correspondentes em estereoscopia. Uma matriz fundamental é a representação algébrica da geometria epipolar. Assim, para um par de pontos correspondentes, representados por vetores $3x1$ vetores homogêneos $(x, x^{'})$ temos que $x^{'T}Fx = 0$, sendo F a matriz fundamental [4], [5].

Uma homografia é uma transformação matemática representada como uma matriz 33. Pontos correspondentes que pertencem a um plano estão relacionadas por uma matriz H correspondente a esse plano, em que $x^{'} = H$. A homografia por meio de transformações matemáticas consegue mapear coordenadas de um ponto de um plano para outro, ou seja de uma imagem para outra. Para um conjunto de pontos $pi = (x, y, z)^T$ sobre um plano π e um conjunto de pontos correspondentes $pi^{'} = (x^{'}, y^{'}, z^{'})^T$ em um plano $\pi^{'}$ a matriz de homografia H $3x3$ pode mapear cada ponto de pi para $pi^{'}$ [6], [4]:

$$\begin{bmatrix} x^{'} \\ y^{'} \\ z^{'} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (1)$$

3. Implementação

Neste trabalho, a construção de um panorama foi implementada em linguagem *C++*, utilizando a biblioteca *OpenCV 3.4*[1]. O panorama foi gerado a partir do vídeo disponível em [7].

Para utilizar o método *SIFT* e obter os pontos correspondentes de cada *frame*, foi utilizando o módulo *NonFree* que faz parte do pacote extra do *OpenCV*, que contém os algoritmos patenteados para detecção e descrição de características. No Pseudocódigo 1 são apresentados os passos realizados para encontrar correspondências entre duas imagens, utilizando a classe *SiftFeatureDetector*. Com isso foram obtidos os *keypoints* e descritores de $image1$ e $image2$, nas linhas 6 e 7. Por meio da biblioteca *FLANN* do *OpenCV*(*Fast Library for Approximate Nearest Neighbors*) foram obtidos as distâncias entre os pontos das imagens, linha 11. Para determinar os pontos correspondentes verificou-se a menor distância, atribuída à variável min_dist (linha 14) e se caso a distância fosse menor do que $3 * min_dist$ então encontra-se uma correspondência.

O Pseudocódigo 2 apresenta a etapa da implementação da matriz de homografia das correspondências geradas. Nas linhas 5 e 6 considera-se os melhores *keypoints*. A matriz de homografia H é encontrada pela função $findHomography$ na linha 8. A função $warpPerpective$ da linha 9 é responsável por gerar as transformações de perspectivas na imagem, o terceiro parâmetro da função é a matriz H obtida anteriormente.

Para construir um panorama de um vídeo, não foram considerados todos os *frames*, com objetivo de tornar o algoritmo mais rápido, além disso de um *frame* para outro não haviam tantas mudanças significativas. Neste trabalho, para o vídeo [7], o panorama foi sendo construído a cada 50 *frames*.

4. Utilização

O algoritmo implementado foi executado em sistema operacional Ubuntu 16.04.3. Para executar o algoritmo, deve-se ir até o diretório contendo o código e o vídeo por meio do terminal e digitar o seguinte comando:

- sh ./run-panorama.sh

No fim da execução do algoritmo, o panorama será salvo no diretório contendo os arquivos com o nome de "panorama.jpg".

1. Disponível em: https://opencv.org/

Algoritmo 1: SIFT

```
1  usingnamespace cv :: xfeatures2d;
2  início
3      Ptr < SiftFeatureDetector > detector =
         SiftFeatureDetector :: create();
4      vector < KeyPoint > keypoints_1, keypoints_2;
5      Matdescriptors_1, descriptors_2;
6      detector− >
         detectAndCompute(image1, Mat(),
         keypoints_1, descriptors_1);
7      detector− >
         detectAndCompute(image2, Mat(),
         keypoints_2, descriptors_2);
8      //;
9      FlannBasedMatcher matcher;
10     vector < DMatch > matches;
11     matcher.match(descriptors_1, descriptors_2, matches);

12     para i ← 0 até descriptors_1.rows faça
13         doubledist ← matches[i].distance;
14         se dist < min_dist então
15             min_dist ← dist;
16         fim
17         se dist > max_dist então
18             max_dist ← dist;
19         fim
20     fim
21     vector < DMatch > good_matches;
22     para i ← 0 até descriptors_1.rows faça
23         se matches[i].distance < 3 ∗ min_dist então
24             good_matches.push_back(matches[i]);
25         fim
26     fim
27 fim
```

5. Resultados

Na Figura 3 é apresentado um exemplo de detecção de correspondências entre dois *frames* do vídeo, utilizando o método *SIFT*, descrito na Seção 2. Na imagem apenas os *keypoints* que encontraram sua correspondência na outra imagem estão representados. Como pode ser observado cada *keypoint* é identificado por um circulo colorido da mesma cor do seu correspondente, em que ambos estão conectados.

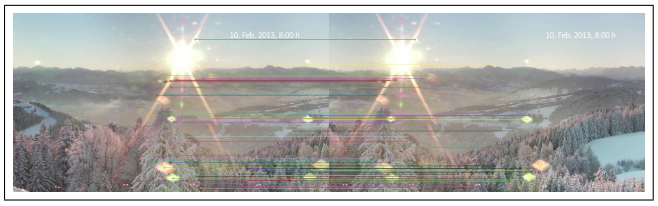

Figura 3: Correspondências entre 2 *frames* encontradas utilizando o método *SIFT*.

Algoritmo 2: MATRIZ DE HOMOGRAFIA

```
1  início
2      vector < Point2f > scene;
3      vector < Point2f > object;
4      para i ← 0 até good_matches.size() faça
5          scene.push_back(
             keypoints_1[good_matches[i].queryIdx].pt);
6          object.push_back(
             keypoints_2[good_matches[i].trainIdx].pt);
7      fim
8      MatH ← findHomography(Mat(object),
         Mat(scene), CV_RANSAC);
9      warpPerspective(image2, warped, H,
         Size(image1.cols ∗ 2, image1.rows))$;
10     panorama ← warped.clone();
11     Mat roi(panorama,
         Rect(0, 0, image1.cols, image1.rows));
12     image1.copyTo(roi);
13 fim
```

A construção do panorama foi realizada a cada 50 *frames* do vídeo. Assim, lia-se um *frame* e criava-se um panorama, que ia sendo utilizado para os panoramas gerados a partir do próximo *frame* considerado, ou seja, 50 *frames* depois. O resultado pode ser observado na Figura 4.

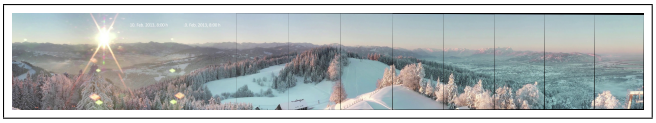

Figura 4: Panorama gerado a cada 50 *frames* de um vídeo.

Além disso, durante os experimentos, o algoritmo também foi executado para gerar um panorama a cada 100 *frames*, como pode ser observado na Figura 5

Figura 5: Panorama gerado a cada 100 *frames* de um vídeo.

6. Conclusão

Neste trabalho, foi estudada e implementada uma técnica para construção de panoramas e aplicada a um vídeo. A técnica consiste em obter descritores de pontos de imagens para encontrar correspondências, além disso, utilizou-se uma matriz homográfica para estabelecer relações para o mapeamento de pontos correspondentes de uma imagem para outra.

Como pode ser observado nas Figuras 4 e 5, a construção do panoramas foi realizada de forma que os pontos da imagem fossem realmente correspondentes, dando a imagem

uma sensação de continuidade, como se fosse o mesmo *frame*. Apesar disso, o algoritmo implementado acabou não tratando da melhor forma a união desses *frames*, causando em alguns pontos da imagem faixas pretas. Comparando ambas as figuras, o panorama gerado a cada 100 *frames* possui menos faixas, isso pode ser explicado pelo fato de que como mais *frames* de passaram, a união acaba ocorrendo mais no final de um *frame*. Como trabalhos futuros, sugere-se a melhoria do algoritmo implementado, principalmente, buscando evitar as faixas pretas no panorama final.

Referências

[1] M. Brown, D. G. Lowe *et al.*, "Recognising panoramas." in *ICCV*, vol. 3, 2003, p. 1218.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[3] R. Szelis ki, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[4] E. Vincent and R. Laganiére, "Detecting planar homographies in an image pair," in *Image and Signal Processing and Analysis, 2001. ISPA 2001. Proceedings of the 2nd International Symposium on*. IEEE, 2001, pp. 182–187.

[5] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[6] M. SANTOS, "Revisão de conceitos em projeção, homografia, calibração de câmera, geometria epipolar, mapas de profundidade e varredura de planos. 2012."

[7] (2013, feb) perfect winter day / perfekter tag im winter. [Online]. Available: https://www.youtube.com/watch?v=VDTEyQhZzKA