

Bài tập lớn 2: Local Feature Matching

Giới thiệu

Bài tập lớn này tập trung vào việc so trùng đặc trưng giữa hai ảnh. Bao gồm ba bước chính:

- Interest detection point: Xác định các vị trí key points trong ảnh bằng cách hiện thực giải thuật Harris Corner Detector.
- Local feature description: Biểu diễn các đặc trưng, các vùng xung quanh vị trí trong ảnh dưới dạng một vector gọi là vector đặc trưng bằng cách hiện thực giải thuật Scale-Invariant feature transform (SIFT).
- Feature matching: So trùng các đặc trưng bằng cách tính toán khoảng cách giữa các điểm đặc trưng và tìm ra các điểm tương đồng bằng cách hiện thực giải thuật Nearest Neighbor Distance Ratio.

Chi tiết hiện thực

Interest detection point

Tìm ra tập các điểm được xem là quan trọng. Các bước hiện thực giải thuật Harris Corner Detector:

1. Blur ảnh trước để tránh nhiễu trong ảnh bằng cách sử dụng gaussian filter.
2. Tính toán đạo hàm của ảnh theo hai trục x trục y bằng sobel filter theo 2 trục.
3. Tính toán các tham số $g(I_x^2)$, $g(I_y^2)$, $g(I_{xy})$ trong đó g là gaussian filter.
4. Trên mỗi điểm ảnh:

- Tính các tham số sau:

$$S_{xx} = \sum_{x,y} g(I_x^2)$$

$$S_{yy} = \sum_{x,y} g(I_y^2)$$

$$S_{xy} = \sum_{x,y} g(I_{xy})$$

- Tính độ đo C bằng công thức sau:

$$C = S_{xx} \circ S_{yy} - S_{xy}^2 - alpha * (S_{xx} + S_{yy})^2$$

- Chọn các điểm key points dựa vào ngưỡng threshold:

- Áp dụng non-maximal suppression để loại bỏ các điểm có khoảng cách dưới min-distance.

Chi tiết hiện thực giải thuật Harris Corner Detector trên python:

```

alpha = 0.06
threshold = 0.01
stride = 2
sigma = 0.1
min_distance = 3
sigma0 = 0.1

# step1: blur image (optional)
filtered_image = filters.gaussian(image, sigma=sigma)

# step2: calculate gradient of image
I_x = filters.sobel_v(filtered_image)
I_y = filters.sobel_h(filtered_image)

# step3: calculate Gxx, Gxy, Gyy
I_xx = np.square(I_x)
I_xy = np.multiply(I_x, I_y)
I_yy = np.square(I_y)

I_xx = filters.gaussian(I_xx, sigma=sigma0)
I_xy = filters.gaussian(I_xy, sigma=sigma0)
I_yy = filters.gaussian(I_yy, sigma=sigma0)

listC = np.zeros_like(image)

# step4: caculate C matrix
for y in range(0, image.shape[0]-feature_width, stride):
    for x in range(0, image.shape[1]-feature_width, stride):
        # matrix 17x17
        Sxx = np.sum(I_xx[y:y+feature_width+1, x:x+feature_width+1])
        Syy = np.sum(I_yy[y:y+feature_width+1, x:x+feature_width+1])
        Sxy = np.sum(I_xy[y:y+feature_width+1, x:x+feature_width+1])

        detC = (Sxx * Syy) - (Sxy**2)
        traceC = Sxx + Syy
        C = detC - alpha*(traceC**2)

        if C > threshold:
            listC[y+feature_width//2, x+feature_width//2] = C

# step5: using non-maximal suppression
ret = feature.peak_local_max(listC, min_distance=min_distance,
                             threshold_abs=threshold)
return ret[:, 1], ret[:, 0]

```

Local feature description

Mô tả các điểm bằng một vector đặc trưng 128 chiều. Các bước hiện thực giải thuật Scale-Invariant feature transform (SIFT):

1. Blur ảnh trước để tránh nhiễu trong ảnh bằng cách sử dụng gaussian filter.
2. Tính toán đạo hàm của ảnh theo hai trục x trục y bằng sobel filter theo 2 trục.

$$\nabla I_0(x_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y} \right)(x_i)$$

3. Tính độ lớn và góc của gradient dựa theo đạo hàm của ảnh theo hai trục.

$$magnitude_gradient = \sqrt{I_x^2 + I_y^2}$$

$$direction_gradient = \arctan(I_x, I_y)$$

4. Với từng key points:

- Áp dụng gaussian filter trên cửa sổ kích thước 16x16 tương ứng với toạ độ của key point.
- Chia cửa sổ 16x16 thành 16 cửa sổ 4x4.
- Trên mỗi cửa sổ 4x4 đó, tính histogram với bins=8. Kết quả tương ứng với 8 feature của vector 128 chiều tại mỗi key point.

5. Reshape ma trận 16x8 thành vector 128 chiều.

6. Chuẩn hoá vector features về [0, 1]

7. Đặt tất cả các giá trị nhỏ hơn 0.2 trong feature vector về 0.2.

8. Chuẩn hoá lại feature vector.

Chi tiết hiện thực giải thuật Scale-Invariant feature transform (SIFT) trên python:

```
sigma_gradient_image = 0.1
sigma_16x16 = 0.4
threshold = 0.2

features = np.zeros((len(x), 4, 4, 8))

# step0: blur image (optional)
filtered_image = filters.gaussian(image, sigma=sigma_gradient_image)

# step1: compute the gradient of image
d_im_x = filters.sobel_v(filtered_image)
d_im_y = filters.sobel_h(filtered_image)

magnitude_gradient = np.sqrt(np.add(np.square(d_im_x), np.square(d_im_y)))
```

```

direction_gradient = np.arctan2(d_im_y, d_im_x)
direction_gradient[direction_gradient < 0] += 2 * np.pi

# step2:
# image.shape[0] = 1024
# image.shape[1] = 768
# x (0 -> 768)
# y (0 -> 1024)
for n, (x_, y_) in enumerate(zip(x, y)):
    # get windows of key point(x, y)
    rows = (y_ - feature_width//2, y_ + feature_width//2 + 1)
    cols = (x_ - feature_width//2, x_ + feature_width//2 + 1)

    if rows[0] < 0:
        rows = (0, feature_width+1)
    if rows[1] > image.shape[0]:
        rows = (image.shape[0]-feature_width-1, image.shape[0]-1)

    if cols[0] < 0:
        cols = (0, feature_width+1)
    if cols[1] > image.shape[1]:
        cols = (image.shape[1]-feature_width-1, image.shape[1]-1)

    # get gradient and angle of key point
    magnitude_window = magnitude_gradient[rows[0]:rows[1], cols[0]:cols[1]]
    direction_window = direction_gradient[rows[0]:rows[1], cols[0]:cols[1]]

    # Gaussian filter on window
    magnitude_window = filters.gaussian(
        magnitude_window, sigma=sigma_16x16)
    direction_window = filters.gaussian(
        direction_window, sigma=sigma_16x16)

    for i in range(feature_width//4):
        for j in range(feature_width//4):
            current_magnitude = magnitude_window[i*feature_width//4: (
                i+1)*feature_width//4, j*feature_width//4:(j+1)*
                                            feature_width//4]

            current_direction = direction_window[i*feature_width//4: (
                i+1)*feature_width//4, j*feature_width//4:(j+1)*
                                            feature_width//4]

            features[n, i, j] = np.histogram(current_direction.reshape(
                -1), bins=8, range=(0, 2*np.pi), weights=current_magnitude
                .reshape(-1))[0]

    # Extract 8 x 16 values into 128-dim vector
    features = features.reshape((len(x), -1,))

    # Normalize vector to [0...1]
    norm = np.sqrt(np.square(features).sum(axis=1)).reshape(-1, 1)
    features = features / norm

    # Clamp all vector values > 0.2 to 0.2

```

```

features[features >= threshold] = threshold

# Re-normalize
norm = np.sqrt(np.square(features).sum(axis=1)).reshape(-1, 1)
features = features / norm

return features

```

Feature matching

Tiến hành so trùng các điểm đặc trưng trên ảnh thứ nhất với tất cả các điểm trên ảnh thứ 2, tìm ra vị trí trên ảnh thứ nhất tương ứng với vị trí nào trên ảnh thứ 2. Các bước thực hiện giải thuật Nearest Neighbor Distance Ratio:

- Trên các feature vector của ảnh thứ nhất:

- Tính khoảng cách Euclidean với tất cả các feature vector trên ảnh thứ 2 theo công thức:

$$\begin{aligned}
 d(p, q) &= \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2} \\
 &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}
 \end{aligned}$$

- Sắp xếp tất cả các khoảng cách theo thứ tự tăng dần.
- Nếu khoảng cách nhỏ nhất nhỏ hơn 0.8 lần khoảng cách nhỏ thứ 2 thì feature vector tương ứng với khoảng cách nhỏ nhất được tính là match giữa hai key point.

Chi tiết hiện thực giải thuật Nearest Neighbor Distance Ratio trên python:

```

threshold = 0.8

matches = []
confidences = []

for i in range(im1_features.shape[0]):
    distances = np.sqrt(np.square(np.subtract(
        im1_features[i, :], im2_features)).sum(axis=1))
    index_sorted = np.argsort(distances)
    if distances[index_sorted[0]] / distances[index_sorted[1]] <
        threshold:
        matches.append([i, index_sorted[0]])
        confidences.append(
            1.0 - distances[index_sorted[0]] / distances[index_sorted[1]])
matches = np.asarray(matches)
confidences = np.asarray(confidences)
return matches, confidences

```

Kết quả

Notre Dame

Ảnh gốc:



Figure 1: *Left: Ảnh 1. Right: Ảnh 2.*

Kết quả:

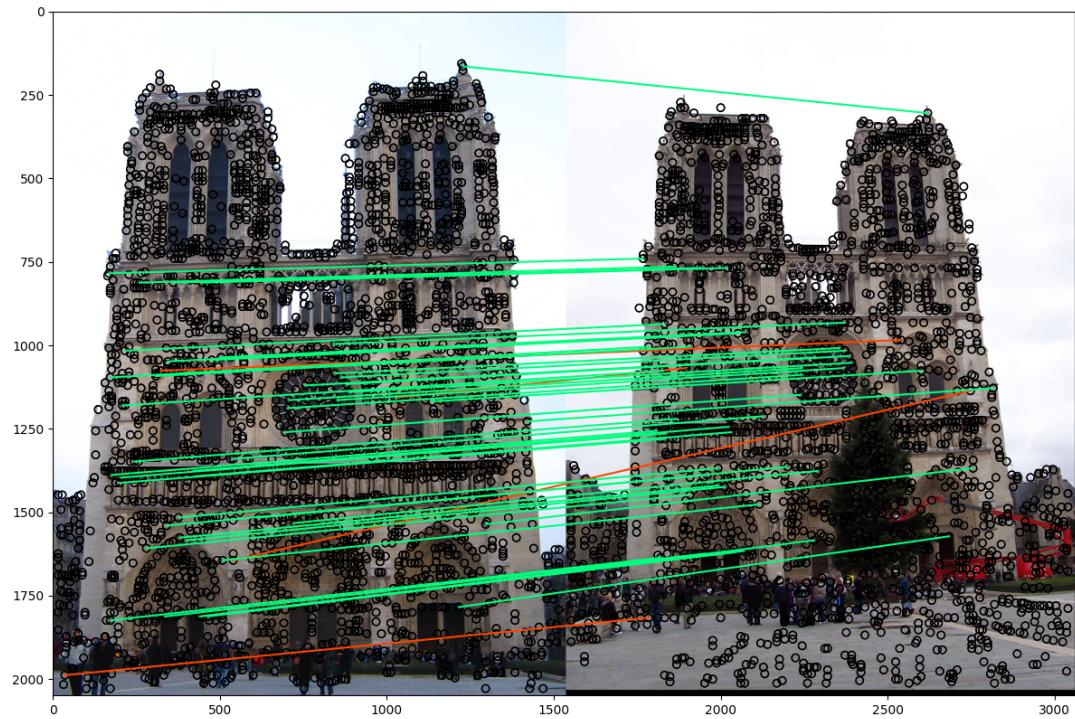


Figure 2: Kết quả sau khi so trùng

```
Getting interest points...
alpha: 0.06, threshold: 0.01, stride: 2, sigma: 0.1, sigma0: 0.1,
      min_distance: 3
alpha: 0.06, threshold: 0.01, stride: 2, sigma: 0.1, sigma0: 0.1,
      min_distance: 3
Done!
Getting features...
sigma_gradient_image: 0.1, sigma_16x16: 0.4, threshold: 0.2
sigma_gradient_image: 0.1, sigma_16x16: 0.4, threshold: 0.2
Done!
Matching features...
threshold: 0.8
Done!
Matches: 134
Accuracy on 50 most confident: 92%
Accuracy on 100 most confident: 83%
Accuracy on all matches: 82%
```

MountRushmore

Ảnh gốc:



Figure 3: *Left: Anh 1. Right: Anh 2.*

Kết quả:

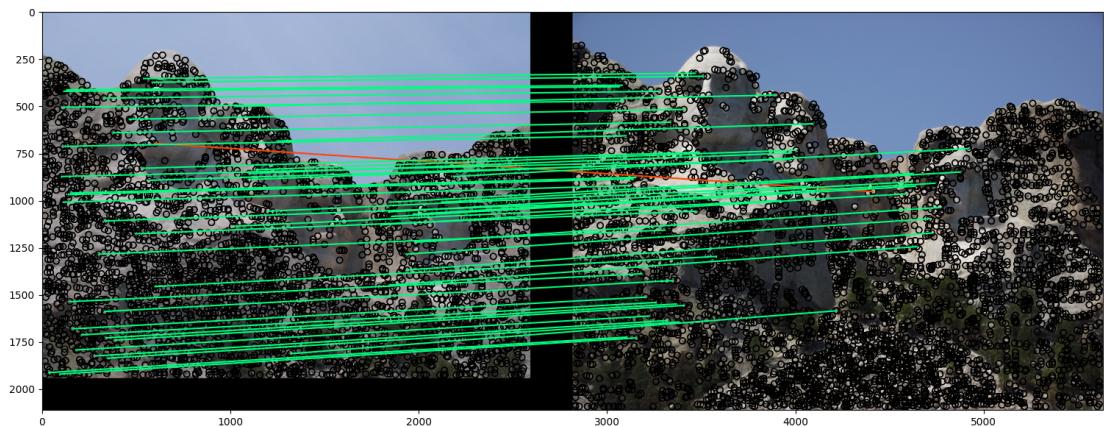


Figure 4: Kết quả sau khi so trùng

```
Getting interest points...
alpha: 0.06, threshold: 0.01, stride: 2, sigma: 0.1, sigma0: 0.1,
      min_distance: 3
alpha: 0.06, threshold: 0.01, stride: 2, sigma: 0.1, sigma0: 0.1,
      min_distance: 3
Done!
Getting features...
sigma_gradient_image: 0.1, sigma_16x16: 0.4, threshold: 0.2
sigma_gradient_image: 0.1, sigma_16x16: 0.4, threshold: 0.2
Done!
```

```
Matching features...
threshold: 0.8
Done!
Matches: 168
Accuracy on 50 most confident: 98%
Accuracy on 100 most confident: 99%
Accuracy on all matches: 96%
```

EpiscopalGaudi

Ảnh gốc:



Figure 5: Left: Ảnh 1. Right: Ảnh 2.

Kết quả:

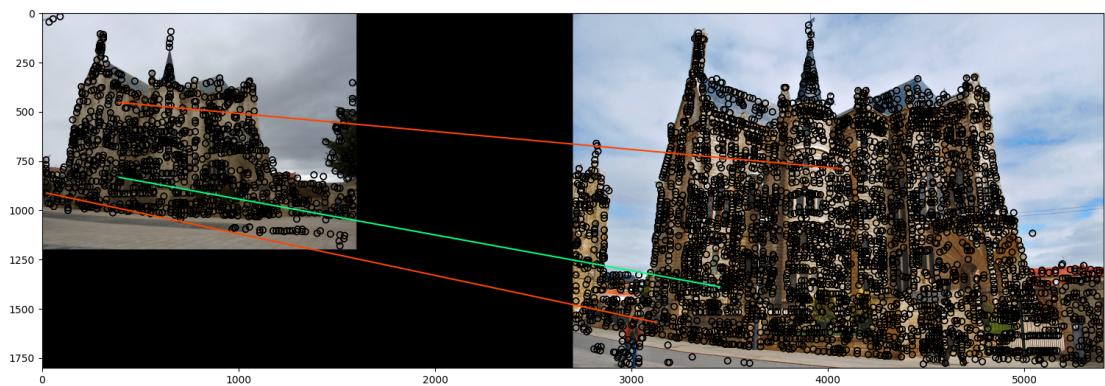


Figure 6: Kết quả sau khi so trùng

```
Getting interest points...
alpha: 0.06, threshold: 0.01, stride: 2, sigma: 0.1, sigma0: 0.1,
```

```
min_distance: 3
alpha: 0.06, threshold: 0.01, stride: 2, sigma: 0.1, sigma0: 0.1,
      min_distance: 3
Done!
Getting features...
sigma_gradient_image: 0.1, sigma_16x16: 0.4, threshold: 0.2
sigma_gradient_image: 0.1, sigma_16x16: 0.4, threshold: 0.2
Done!
Matching features...
threshold: 0.8
Done!
Matches: 3
Accuracy on all matches: 33%
```