THỊ GIÁC MÁY TÍNH VÀ ỨNG DỤNG

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

5th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

6th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Tóm tắt nội dung*—Thị giác máy tính nghiên cứu cách máy tính nhận biết, phân tích và trích xuất thông tin có ý nghĩa từ hình ảnh và video. Tài liệu này trình bày các khái niệm cơ bản và phương pháp thực nghiệm quan trọng: xử lý ảnh và biến đổi cơ bản, phát hiện và mô tả đặc trưng, khớp điểm, theo dõi, phân đoạn, nhận dạng đối tượng và một số kỹ thuật tái tạo không gian ba chiều. Các chương minh họa bằng ví dụ thực nghiệm và mã nguồn Python/OpenCV để người đọc dễ tiếp cận và triển khai. Tài liệu hướng tới sinh viên, nghiên cứu sinh và kỹ sư cần nắm vững nền tảng lý thuyết cùng kỹ năng thực hành để ứng dụng thị giác máy tính trong y tế, giao thông, robot, an ninh và nhiều lĩnh vực khác.**

I. Giới thiệu

Thị giác máy tính (Computer Vision) nhằm trang bị cho máy tính khả năng "nhìn" và "hiểu" nội dung hình ảnh tương tự cách con người xử lý thông tin thị giác. Với sự phát triển của phần cứng, cảm biến và các phương pháp học sâu, thị giác máy tính đã trở thành nền tảng cho nhiều ứng dụng thực tiễn như phân tích hình ảnh y tế, nhận dạng và theo dõi trong video, dẫn đường cho robot, và hệ thống giám sát thông minh.

Mục tiêu của tài liệu này là cung cấp một cái nhìn tổng hợp cả về lý thuyết và thực hành: từ các phép xử lý tiền xử lý ảnh, trích xuất đặc trưng, đến các phương pháp ghép và mô phỏng ba chiều, cùng các ứng dụng và ví dụ mã nguồn. Nội dung được bố cục thành các chương tương ứng với các chủ đề chính, kèm hướng dẫn cài đặt môi trường và tập dữ liệu mẫu để người đọc thực hành.

II. Xử lý ảnh cơ bản và ngôn ngữ lập trình Python

Chương này giới thiệu những kiến thức cơ bản về xử lý hình ảnh và ngôn ngữ lập trình Python được sử dụng để lập trình. Các ví dụ có thể trong chương này được dùng để giải thích các cơ chế viên Python phổ biến dùng với nhiều mô hình đã xây dựng sẵn để thực nghiệm. Một số công cụ cơ bản để bắt đầu lập trình xử lý ảnh trên ngôn ngữ Python ví dụ như đọc hình ảnh, chuyển đổi và nhận rõng hình ảnh, tính toán đạo hàm, vẽ hoặc lưu kết quả sẽ được giới thiệu trong phần này.

A. PIL - Thư viện hình ảnh Python

Thư viện hình ảnh Python (PIL) cung cấp chức năng cho việc xử lý hình ảnh và nhiều thao tác hình ảnh cơ bản hữu ích như thay đổi kích thước, cắt xén, xoay, chuyển đổi màu sắc và các tác vụ khác. Thư viện PIL có thể tải miễn phí tại:

http://www.pythonware.com/products/pil/

Với thư viện PIL, có thể đọc và ghi các hình ảnh từ hầu hết các định dạng phổ biến nhất hiện nay. Một trong những module quan trọng nhất trong xử lý hình ảnh là Image module. Để đọc một hình ảnh có thể sử dụng đoạn chương trình sau:

```
from PIL import Image
pil_im = Image.open('empire.jpg')
```

Giá trị trả về, `pil_im`, là một đối tượng hình ảnh.

Chuyển đổi màu được thực hiện bằng lệnh `convert()`. Để đọc một hình ảnh và chuyển đổi nó sang thang độ xám, chỉ cần thêm `convert('L')` như sau:

```
pil_im = Image.open('empire.jpg').convert('L')
```

1) Chuyển đổi hình ảnh sang định dạng khác: Sử dụng lệnh `save()`, PIL có thể lưu hình ảnh ở tất cả các định dạng. Tài liệu chi tiết về PIL có sẵn tại:

http://www.pythonware.com/library/pil/handbook/

2) Tạo hình thu nhỏ: Sử dụng PIL để tạo hình thu nhỏ khá đơn giản. Lệnh `thumbnail()` lấy một bộ chỉ định kích thước mới và chuyển đổi hình ảnh thành hình thu nhỏ với kích thước vừa với bộ dữ liệu đó. Để tạo hình ảnh thu nhỏ với kích thước tối đa là 128 pixel, hãy sử dụng cú pháp như sau:

```
pil_im.thumbnail((128,128))
```

3) Sao chép và dán các vùng: Cắt một vùng từ một hình ảnh được thực hiện bằng lệnh `crop()`. Vùng được xác định bởi 4 thông số, trong đó toạ độ là (trái, trên, phải, dưới). PIL sử dụng hệ toạ độ có (0,0) ở góc trên bên trái. Ví dụ, vùng được trích xuất có thể được lật bằng lệnh `paste()` hoặc `rotate` như sau:

```
box = (100,100,400,400)
region = pil_im.crop(box)
region = region.transpose(Image.ROTATE_180)
pil_im.paste(region,box)
```

4) Thay đổi kích thước và xoay: Để thay đổi kích thước hình ảnh, gọi hàm `resize()` bằng một bộ thông số cho kích thước mới:

```
out = pil_im.resize((128,128))
```

Để xoay hình ảnh, sử dụng các góc ngược chiều kim đồng hồ và xoay () như sau:

```
out = pil_im.rotate(45)
```

B. Matplotlib

Khi làm việc với toán học và vẽ đồ thị hoặc các các điểm, đường thẳng và đường cong trên hình ảnh, Matplotlib là một thư viện đồ họa tỏi với các tính năng mạnh hơn nhiều so với thị có sẵn trong PIL. Matplotlib tạo ra các hình ảnh chất lượng cao giống như nhiều hình minh họa được sử dụng trong cuốn sách này. Giao điểm Matplotlib PyLab là tập hợp các chức năng cho phép người dùng tạo các đồ thị. Matplotlib là nguồn mở, xem tại http://matplotlib.sf.net/.

1) Vẽ hình ảnh, điểm và đường: Có thể tạo ra các đồ thị cốt, lược đồ hình tròn, các lược đồ phân tán,... với một vài lệnh cho hầu hết các mục đích của thị giác máy tính. Quan trọng nhất, muốn hiển thị những điểm đặc trưng, sự đối xứng và các đối tượng được phát hiện bằng cách sử dụng điểm và đường, đây là một ví dụ về đồ thị với một vài điểm và đường thẳng:

```
from PIL import Image
from pylab import *
im =
    array(Image.open('empire.jpg').convert('L'))
figure()
gray()
contour(im, origin='image')
axis('equal')
axis('off')
show()
```

2) Đường biên ảnh và lược đồ: Sau đây xem xét hai ví dụ về các đồ thị đặc biệt: đường biên hình ảnh và lược đồ hình ảnh. Việc quan sát đường biên ảnh cung cấp nhiều thông tin rất hữu ích. Điều này có ích khi biết hình ảnh được tạo ra như thế nào (ví dụ, độ sáng hoặc độ bão hòa) và các đường biên được thực hiện trên mỗi một lớp. Sau đây là một số tùy chọn để hiển thị đồ thị:

```
figure()
hist(im.flatten(),128)
show()
```

C. NumPy

NumPy (http://scipy.org) là một gọi thư viện được sử dụng phổ biến cho khoa học trong Python. NumPy chứa một số khối nên tảng mà chúng ta sẽ dùng trong hầu hết tất cả các ví dụ trong suốt cuốn sách này. Đối tượng mảng cho phép thực hiện các thao tác quan trọng như nhận ma trận, hoán vị, giải hệ phương trình, nhân vector và chuẩn hóa, cần thiểu hiển những việc như cần chỉnh hình ảnh, làm công ảnh, biến đổi mô hình, phân loại hình ảnh, phân nhóm hình ảnh, v.v.

1) Thể hiện hình ảnh dưới dạng mảng: Hình ảnh trong các ví dụ trước được tải và chuyển đổi thành các đối tượng mảng NumPy với lệnh `array()` nhưng không đề cập đến điều gì. Mảng NumPy là đa chiều và có thể dùng mô tả cho vecto, ma trận và hình ảnh. Một mảng rất giống với một danh sách (hoặc danh sách các danh sách) nhưng nhanh hơn rất nhiều để làm các thành toán hơn.

Trí khi được chỉ định một hình ảnh bên trên, sẽ nhận được kết quả đầu ra sau:

```
255
253
200
255
```

Việc đảo ngược phép biến đổi `array()` có thể được thực hiện bằng cách sử dụng hàm PIL `fromarray()` như sau:

```
pil_im = Image.fromarray(im)
```

2) Biến đổi cấp độ xám: Sau khi đọc hình ảnh cho mảng NumPy, có thể thực hiện bất kỳ thao tác toán học nào muốn thực hiện trên chúng. Một ví dụ đơn giản về điều này là biến đổi cấp độ xám của hình ảnh. Lấy bất kỳ hàm f nào ánh xạ khoảng 0...255 (hoặc 0...1 sau khi đã chuẩn hóa hình ảnh) với chính nó (có nghĩa là đầu ra có cùng khoảng với đầu vào). Dưới đây là một số ví dụ:

```
im2 = 255 - im  # invert image
im3 = (100.0/255) * im + 100
im4 = 255.0 * (im/255.0)**2  # squared
```

3) Phép tính trung bình pixel ảnh: Phép tính trung bình pixel ảnh là một cách đơn giản để giảm nhiễu hình ảnh và cũng thường được coi như một tập các phép toán lọc Gaussian. Tính toán lọc hình ảnh trung bình ở một tập các lần có thể đạt được thực hiện khá đơn giản.

Giá sự tất các các hình ảnh có cùng kích thước, có thể tính trung bình của tất cả các hình ảnh tại bằng cách tổng hợp chúng và chia cho số lượng hình ảnh. Thêm chức năng này vào tập tin `imtools.py`.

4) Phân tích thành phần chính (PCA) của hình ảnh: Phân tích thành phần chính (PCA) là một kỹ thuật hữu ích để giảm kích thước và tối ưu theo nghĩa là nó thể hiện dữ liệu huấn luyện đa dạng với kích thước càng ít càng tốt. Ngay cả một hình ảnh xám có kích thước 100x100 pixel cũng có 10,000 chiều và có thể được xem như là một điểm trong không gian 10,000 chiều. Một hình ảnh có kích thước megapixel sẽ có hàng triệu chiều. Với kích thước như vậy, không có gì ngạc nhiên khi giảm kích thước có ích trong những ứng dụng thị giác máy tính. Ma trận chiếu từ PCA có thể được tái sử dụng với các hình ảnh mới để chiếu vào không gian trong đó rồi so sáp với theo thứ tự quan trọng trong giảm dần.

D. SciPy

SciPy (http://scipy.org) là một gọi thư viên nguồn mở dành cho toán học xây dựng trên NumPy và cung cấp các các công

tận cần thiết bao gồm phép tính tích phân, tối ưu, thống kê, xử lý tín hiệu và xử lý hình ảnh, tọa là quan trọng nhất là số lý học. Nhập tiếp theo sau đây sẽ trình bày nhiều module hữu ích trong thư viện SciPy.

1) Làm mờ ảnh: Một ví dụ kinh điển và rất hữu ích về tích chập hình ảnh là làm mờ hình ảnh dùng mặt na Gaussian. Vẽ bản chất, hình ảnh (thang độ xám) I tích chập với mặt na Gaussian được định nghĩa là:

$$I_\sigma = I * G_\sigma \tag{1}$$

Trong đó * biểu thị tích chập và G_σ là mặt na 2D Gaussian với độ lệch chuẩn σ được định nghĩa là:

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

Việc làm mờ ảnh dùng mặt na Gaussian được sử dụng để định nghĩa không gian tỷ lệ hình ảnh phảng trên những ứng dụng khác và trong nhiều ứng dụng khác.

SciPy đi kèm với một module để loc ảnh được gọi là `scipy.ndimage.filters` có thể được sử dụng để phân tách nhanh một chiều. Đoạn chương trình mẫu được mô tả ở bên dưới:

```
from PIL import Image
from numpy import *
from scipy.ndimage import filters
im =
    array(Image.open('empire.jpg').convert('L'))
im2 = filters.gaussian_filter(im,5)
```

2) Tính đạo hàm của ảnh (Image derivatives): Giá trị cường độ pixel trên ảnh thay đổi như thế nào là thông tin quan trọng, được sử dụng cho nhiều ứng dụng như thể thấy trong suốt cuốn sách này. Sự thay đổi cường độ được mô tả với các đạo hàm theo trục x và ylà I_x và I_y của ảnh mẫu I (đôi với ảnh màu, các đạo hàm thường được lấy cho mỗi kênh màu).

Vector gradient của ảnh $\nabla I = [I_x \quad I_y]^T$. Gradient có hai thuộc tính quan trọng đó là độ lớn vector gradient:

$$\nabla I = \sqrt{I_x^2 + I_y^2} \tag{3}$$

Và góc của vector gradient:

$$\alpha = \arctan 2(I_y, I_x) \tag{4}$$

cho biết hướng thay đổi cường độ lớn nhất tại mỗi điểm (pixel) trong ảnh.

Việc tính toán các đạo hàm hình ảnh có thể được thực hiện bằng cách sử dụng các phép sai phân với đạo hàm hữu hạn cơ bản. Phép tính để tính toán đạo hàm được áp dụng thực hiện nó là phép tích chập như sau:

$$I_x = I * D_x, \quad I_y = I * D_y \tag{5}$$

Hai lựa chọn phổ biến cho Dx và Dy là các bộ lọc Prewitt:

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad D_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Và các bộ lọc Sobel:

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad D_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

3) Hình thái học: Hình thái học là một framework và tập hợp các phương pháp xử lý hình ảnh để đo lường và phân tích các hình dạng cơ bản. Hình thái thường được áp dụng cho hình ảnh nhị phân nhưng cũng có thể được sử dụng cho hình ảnh thang độ xám. Hình ảnh nhị phân là hình ảnh trong đó mỗi pixel chỉ có hai giá trị, thường là 0 và 1. Hình ảnh nhị phân thường là kết quả của việc lấy ngưỡng một hình ảnh, ví dụ với ý định cảm các đối tượng hoặc kích thước của chúng.

Các toán tử hình thái được trình bày trong `scipy.ndimage` với module morphology. Các chức năng được đặt tên để làm lượng cho hình ảnh nhị phân trong các phép đo mô hình `scipy.ndimage`. Phần tích một ví dụ đơn giản về cách sử dụng chúng dưới đây.

III. MÔ TẢ ĐẶC TRƯNG HÌNH ẢNH

Nội dung chính trong chương này là tìm các điểm và vùng tương ứng giữa các hình ảnh khác nhau. Hai loại đặc trưng mà phương pháp phổ biến trung sẽ được giới thiệu. Các đặc trưng cục bộ này sẽ được sử dụng trong nhiều xuyên suốt cuốn sách này và là một khái quan trọng trong nhiều ứng dụng như tìm hình ảnh tương tự (similar image), thực tế ảo và tính toán xây dựng hình ảnh 3 chiều.

A. Xác định đặc trưng góc Harris

Thuật toán phát hiện góc Harris (hoặc đôi khi là máy dò góc Harris & Stephens) là một trong những thuật toán tìm vị trí đặc trưng góc đơn giản nhất được sử dụng. Ý tưởng chính là các góc trong hình ảnh là các điểm quan trọng ở khu vực lân cận thì các cạnh theo nhiều hướng, đây là các góc hình ảnh.

Xác định ma trận $M_I = M_I(x)$ trên các điểm x trong miền hình ảnh, là ma trận mũa xác định mô tả các trọng đổi xung:

$$M_I = \nabla I^T I^T = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{6}$$

trong đó ∇I là gradient hình ảnh chứa các đạo hàm I_x và I_y (đã định nghĩa các đạo hàm và gradient ở chương trước). Do cấu trúc này, M_I có hạng là một với trị riêng $\lambda_1 \neq 0$ và $\lambda_2 \approx 0$. Như vậy sẽ thu được ma trận cho mỗi pixel trong ảnh.

Đặt W là ma trận trọng số (thường là bộ lọc Gaussian), tích chập thành phần:

$$\tilde{M}_I = W * M_I \tag{7}$$

đưa ra mức trung bình cục bộ của M_I sau với các pixel lân cận. Ma trận kết quả $\tilde{M}_I$ đôi khi được gọi là ma trận Harris. Chiều rộng của W xác định vùng quán tâm xung quanh x. Ý tưởng lấy trung bình ma trận M_I trên một vùng như thế này là giá trị riêng sẽ thay đổi tùy thuộc vào các thuộc tính hình

ảnh cục bộ. Nếu đó đặc khác nhau trong khu vực, giá trị riêng thứ hai của $\tilde{M}_I$ sẽ không còn bằng không.

Tùy thuộc vào các giá trị của ∇I trong khu vực, có ba trường hợp cho các giá trị riêng λ_1 và λ_2 đều là các giá trị dương lớn, thì có một góc tại x.

- Nếu $\lambda_1 \gg \lambda_2$ (hoặc ngược lại), thì có một cạnh.
- Nếu λ_1 và λ_2 đều nhỏ, thì không có gì.

Để phân biệt trường hợp quan trọng với các trường hợp khác mà không thực sự phải tính toán các giá trị riêng, Harris và Stephens [12] đã giới thiệu một hàm chi tiết:

$$\det(\tilde{M}_I) - \kappa \text{trace}(\tilde{M}_I)^2 \tag{8}$$

Để tránh trường hợp trung số k không thay đổi, cách để dùng phải là thay bằng việc sử dụng thương số để làm thông số chi thị vị trí góc:

$$\frac{\det(\tilde{M}_I)}{\text{trace}(\tilde{M}_I)^2} \tag{9}$$

Để thực hiện thuật toán này trên chương trình, cần dùng module `scipy.ndimage` có sẵn các đạo hàm bằng các bộ lọc Gaussian như mô tả ở chương trên đã loại bộ đô này nhiều trong quá trình phát hiện góc.

Trước tiên, thêm hàm tính góc vào một tệp `harris.py` sử dụng đạo hàm Gaussian. Một lần nữa thêm số σ xác định thông số tỷ lệ của các bộ lọc Gaussian được sử dụng. Cũng có thể thử lại ngưỡng phù hợp với lượng kích thước khác nhau theo hướng x và y cũng như tỷ lệ để lấy trung bình trong việc tính ma trận Harris.

```
from scipy.ndimage import filters

def compute_harris_response(im,sigma=3):
    """Compute the Harris corner detector
    response
    for each pixel in a grayscale image."""

    # derivatives
    imx = zeros(im.shape)
    filters.gaussian_filter(im,
    (sigma,sigma), (0,1), imx)
    imy = zeros(im.shape)
    filters.gaussian_filter(im,
    (sigma,sigma), (1,0), imy)

    # compute components of the Harris matrix
    Wxx =
    filters.gaussian_filter(imx*imx,sigma)
    Wxy =
    filters.gaussian_filter(imx*imy,sigma)
    Wyy =
    filters.gaussian_filter(imy*imy,sigma)

    # determinant and trace
    Wdet = Wxx*Wxy - Wxy**2
    Wtr = Wxx + Wyy

    return Wdet / Wtr
```

Kết quả chương trình cho ra một hình ảnh với mỗi pixel chứa giá trị của hàm phản hồi Harris. Bây giờ vấn đề chỉ là chọn ra thông tin cần thiết từ hình ảnh này bằng cách lấy tất cả các điểm có giá trị trên ngưỡng đặt trước. Để làm điều này, hãy lấy tất cả các pixel ứng viên cho đặc trưng góc, sắp xếp chúng theo thứ tự giảm dần và đánh dấu các vùng quả gần với các vị trí có giá trị lớn nhất làm các góc. Thêm hàm sau vào `harris.py`.

```
def
    get_harris_points(harrisim,min_dist=10,threshold=0.1)
    """Return corners from a Harris response
    image
    min_dist is the minimum nbr of pixels
    separating
    corners and image boundary."""

    # find top corner candidates above a
    threshold
    corner_threshold = harrisim.max() *
    threshold
    harrisim_t = (harrisim >
    corner_threshold) * 1

    # get coordinates of candidates
    coords = array(harrisim_t.nonzero()).T

    # ...and their values
    candidate_values = [harrisim[c[0],c[1]]
    for c in coords]

    # sort candidates
    index = argsort(candidate_values)

    # store allowed point locations in array
    allowed_locations = zeros(harrisim.shape)
    allowed_locations[min_dist:-min_dist,
                      min_dist:-min_dist] = 1

    # select the best points taking
    min_distance
    # into account
    filtered_coords = []
    for i in index:
        if
    allowed_locations[coords[i,0],coords[i,1]]
    == 1:
            filtered_coords.append(coords[i])
            allowed_locations[

    (coords[i,0]-min_dist):(coords[i,0]+min_dist),

    (coords[i,1]-min_dist):(coords[i,1]+min_dist)]
    = 0

    return filtered_coords
```

Bây giờ chúng ta đã có tất cả những gì cần để phát hiện các điểm góc trong hình ảnh. Để hiển thị các điểm góc trong ảnh, có thể thêm hàm vẽ lược đồ vào `harris.py` bằng Matplotlib như sau.

```
def plot_harris_points(image,filtered_coords):
    """Plots corners found in image."""

    figure()
    gray()
    imshow(image)
```

```
    plot([p[1] for p in filtered_coords],
         [p[0] for p in filtered_coords],'*')
    axis('off')
    show()
```

Thử chạy các lệnh trong đoạn chương trình sau để thực hiện phát hiện góc đây dù cho một hình ảnh:

```
im =
    array(Image.open('empire.jpg').convert('L'))
harrisim = harris.compute_harris_response(im)
filtered_coords =
    harris.get_harris_points(harrisim,6)
harris.plot_harris_points(im, filtered_coords)
```

Hình ảnh được mở và chuyển đổi sang thang độ xám. Sau đó, tính toán và lựa chọn các điểm Harris. Cuối cùng, các điểm được vẽ chồng lên hình ảnh gốc. Kết quả được trình bày trong Hình 2.1.

1) Tìm điểm tương ứng giữa các hình ảnh: Trình phát hiện góc Harris đưa ra các điểm đặc trưng nhưng muốn trong hình ảnh nhưng không chứa cách để so sánh vấn đề các điểm đặc trưng này trên các hình ảnh. Bước tiếp theo là thêm một mô tả cho mỗi điểm và mô tả điểm đặc trưng dùng đó để tìm kiếm điểm đặc trưng tương ứng của hình còn lại.

Nổi chung, cách tính tương quan giữa hai điểm hình ảnh (có kích thước bằng nhau) I1 (x) và I2 (x) được định nghĩa là:

$$c(I_1, I_2) = \sum_x f(I_1(x), I_2(x)) \tag{10}$$

trong đó hàm f thay đổi tùy theo phương pháp tương quan. Tổng được lấy trên tất cả các vị trí x trong các điểm hình ảnh. Đối với phép tính tương quan chéo $f(I_1, I_2) = I_1 \cdot I_2$ và như vậy $(I_1, I_2) = I_1 \cdot I_2$ với là tích vô hướng. Giá trị của (I_1, I_2) càng lớn, các điểm I_1 và I_2 càng giống nhau.

Tương quan chéo chuẩn hóa là một biến thể của tương quan chéo được định nghĩa là:

$$ncc(I_1, I_2) = \frac{1}{n-1} \sum_x \frac{(I_1(x) - \mu_1)(I_2(x) - \mu_2)}{\sigma_1 \sigma_2} \tag{11}$$

Trong đó n là số pixel trong một điểm, μ_1 và μ_2 là cường độ trung bình và σ_1, σ_2 là độ lệch chuẩn trong mỗi điểm. Bằng cách trừ trung bình và chia tỷ lệ với độ lệch chuẩn, phương pháp này bền vững với sự thay đổi cường độ sang của hình ảnh.

Để trích xuất các phần vùng ảnh và so sánh chúng bằng cách sử dụng tương quan chéo được chuẩn hóa, cần thêm hai hàm trong `harris.py`. Thêm vào các câu lệnh trong chương trình sau:

```
def
    get_descriptors(image,filtered_coords,wid=5):
    """For each point return pixel values
    around the point using a neighbourhood
    of width 2*wid+1. (Assume points are
    extracted with min_distance > wid)."""

    desc = []
    for coords in filtered_coords:
        patch =
    image[coords[0]-wid:coords[0]+wid+1,

    coords[1]-wid:coords[1]+wid+1].flatten()
        desc.append(patch)

    return desc

def match(desc1,desc2,threshold=0.5):
    """For each corner point descriptor in the
    first image, select its match in the
    second
    image using normalized cross
    correlation."""

    n = len(desc1[0])

    # pair-wise distances
    d = -ones((len(desc1),len(desc2)))
    for i in range(len(desc1)):
        for j in range(len(desc2)):
            d1 = (desc1[i] - mean(desc1[i]))
    / std(desc1[i])
            d2 = (desc2[j] - mean(desc2[j]))
    / std(desc2[j])
            ncc_value = sum(d1 * d2) / (n-1)
            if ncc_value > threshold:
                d[i,j] = ncc_value

    ndx = argsort(-d)
    matchscores = ndx[:,0]

    return matchscores
```

Hàm đầu tiên lấy một phần vùng của ảnh xám hình vuông có chiều dài cạnh là tập trung quanh điểm đặc trưng. Sau đó các mô tả này được sử dụng trong hàm thứ hai để khớp điểm với điểm tìm kiếm tốt nhất trong ảnh khác bằng cách sử dụng tương quan chéo được chuẩn hóa. Lưu ý việc sử dụng phần từ cuối cùng (chỉ số -1) của cdf để chuẩn hóa no trong khoảng từ 0...1. Thực hiện mô tả khi bằng trọng của các hình ảnh như sau:

```
from PIL import Image
from numpy import *
import harris

im1 =
    array(Image.open('image1.jpg').convert('L'))
im2 =
    array(Image.open('image2.jpg').convert('L'))

wid = 5
harrisim =
    harris.compute_harris_response(im1,5)
filtered_coords1 = harris.get_harris_points(
    harrisim,wid+1)
d1 =
    harris.get_descriptors(im1,filtered_coords1,wid)

harrisim =
    harris.compute_harris_response(im2,5)
filtered_coords2 = harris.get_harris_points(
    harrisim,wid+1)
d2 =
    harris.get_descriptors(im2,filtered_coords2,wid)
```

```
print('starting matching')
matches = harris.match_twosided(d1,d2)

figure()
gray()
harris.plot_matches(im1,im2,filtered_coords1,
                    filtered_coords2,matches)
show()
```

Nếu chỉ muốn vẽ một tập hợp con của các điểm tương đồng để quan sát rõ ràng hơn, thay thế các kết quả điểm tương đồng trong ví dụ `matches[:100]` hoặc một bộ chỉ số ngẫu nhiên.

Như có thể thấy trong Hình 2.2, có khá nhiều kết quả khớp không chính xác. Điều này là do cách tính tương quan chéo trên các điểm hình ảnh không được điều chỉnh chính xác như các phương pháp hiện đại. Do đó, điều quan trọng là sử dụng các phương pháp bền vững để khớp tương đồng trong các bài toán thực tế. Mô tả khác để làm điều đồ này không phải là bất biến đổi với việc như nhớ hoặc xoay và việc lựa chọn kích thước điểm ảnh hướng đến kết quả.

Trong những năm gần đây, đã có rất nhiều sự phát triển trong việc cải thiện đặc tính phát hiện và mô tả điểm đặc trưng. Hãy cùng xem một trong những thuật toán tối nhất trong phần tiếp theo sau đây.

B. Đặc trưng bất biến với sự thay đổi tỷ lệ ảnh (SIFT)

Một trong những điểm đặc trưng thành công nhất trong thập kỷ qua là đặc trưng bất biến với sự thay đổi tỷ lệ ảnh (Scale-Invariant Feature Transform-SIFT), được giới thiệu bởi David Lowe trong [17]. SIFT sau đó đã được tinh chỉnh và mô tả chi tiết trong bài báo [18]. SIFT bao gồm cả trình phát hiện điểm đặc trưng và mô tả điểm đặc trưng. Nhìn chung, thuật toán SIFT được mô tả dựa trên các bước chính sau: phát hiện điểm cực trị không gian tỷ lệ, định vị điểm then chốt xác, gán định hướng và mô tả điểm then chốt.

1) Phát hiện điểm cực trị trong không gian tỷ lệ: Đầu tiên, chúng ta xây dựng tập kết quả của hình ảnh góc được làm mịn liên tục với mặt na Gaussian với giá trị độ lệch chuẩn thay đổi khác nhau. Tập hình ảnh Differential of Gaussian (DoG) (sai phân của mặt na Gaussian) của hình ảnh sẽ được bằng cách trừ liên tục các hình ảnh được làm mịn liên kế. Bằng cách so sánh từng pixel của thang độ hiển tại với tỷ lệ trên và dưới trong khu vực 3x3, tức là 26 pixel, chúng ta có thể tìm thấy giá trị đỗ là hoặc tối thiểu trong 8 chúng. Những điểm này cũng được coi là một ứng cử viên của điểm đặc trưng. Các phương trình dưới đây sẽ được sử dụng để mô tả chức năng Gaussian, không gian tỷ lệ và DoG:

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{12}$$

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{13}$$

Ở đây * là hàm convolution của x và y.

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma) \tag{14}$$

2) Định vị chính xác điểm mẫu chốt: Kết quả ban đầu của thuật toán này, đặc giá chi xem vị trí điểm then chốt là ở trung tâm của điểm mẫu. Tuy nhiên, đây không phải là vị trí cho giá trị cực đại chính xác của một điểm then chốt. Nó cần một hàm xấp xỉ hạc hai để nối suy ra chính xác vị thực, tức là mức độ chính xác của điểm đặc trưng có thể ở mức nhỏ hơn đơn vị pixel (sub-pixel). Các giá đã sử dụng sự mở rộng Taylor của hàm không gian tỷ lệ đã cách chuyển thích vài chi số:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \tag{15}$$

ở đây D và đạo hàm tại giá trị điểm mẫu và $x = (x, y, \sigma)^T$ là điểm dịch chuyển từ điểm mẫu này. Vị trí cực trị, $\hat{x}$, được xác định bằng cách đạo hàm của hàm này với biến x và tính đạo hàm này tại 0,

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \tag{16}$$

3) Mô tả điểm đặc trưng: Trong bước này, điểm đặc trưng sẽ được mô tả. Mỗi điểm chủ chốt sẽ được giới hạn trong một vùng quanh kích thước khoảng 16x16, và được vùng nhớ 4x4 được tính toán trong biểu đồ với 8 hướng khác nhau. Sau khi tích lũy lại độ lớn gradient trong vùng 4x4, ta có thể tìm được điểm mẫu. Mỗi điểm mẫu sẽ đại diện cho 4x4 hướng, và xây vựng mô tả sẽ bao gồm tổng cộng 16x16x8 điểm.

4) Lập trình xác định đặc trưng SIFT trong ảnh: Để tính toán các đặc trưng SIFT cho hình ảnh, sẽ sử dụng gói nguồn mở VLFeat [**?**]. Việc triển khai đây đủ tất cả các bước trong thuật toán sẽ không hiệu quả làm và không thuộc phạm vi của cuốn sách này. Tại gói VLFeat tại http://www.vlfeat.org/, chứa cả tài liệu SIFT.

Thư viện được viết bằng C nhưng có giao diện dùng các dòng lệnh chúng ta có thể sử dụng. Ngoài ra còn có giao diện Matlab và trình ngôn ngữ Python http://github.com/mm/mkmikic có các cách thức hiện SIFT có sẵn tại trang web Lowe http://www.cs.ubc.ca/~loweschi dành cho Windows và Linux).

Tạo một tập tin `sift.py` và thêm hàm sau đây để gọi tệp thực thi.

```
def process_image(imagename,resultname,
                  params="--edge-thresh 10
                          --peak-thresh 5"):
    """Process an image and save the results
    in a file."""

    if imagename[-3:] != 'pgm':
        # create a pgm file
        im =
    Image.open(imagename).convert('L')
        im.save('tmp.pgm')
        imagename = 'tmp.pgm'

    cmnd = str('sift '+imagename+' --output='
               +resultname+' '+params)
    os.system(cmnd)
    print('processed', imagename, 'to',
    resultname)
```

Các tệp nhị phân cần hình ảnh mức xám ở định dạng .pgm, vì vậy nếu sử dụng định dạng hình ảnh khác, trước tiên hãy chuyển tên file để đọc. Kết quả được lưu trữ trong một tệp văn bản ở định dạng để đọc. Các tệp tin trong giống như thế này:

```
318.861 7.40227 1.12001 1.68523 0 0 0 1 0 0 0 0 0 0 11 16 0 ...
318.861 7.40227 1.12001 0.99965 11 0 0 3 0 0 2 0 0 0 42 0 ...
54.2821 14.8586 0.895827 4.29821 60 46 0 0 0 0 0 0 99 42 0 0 ...
155.714 23.8575 1.03040 5.54095 6 0 0 0 150 11 0 0 141 45 8 1 ...
42.9729 24.2012 0.969313 4.68892 90 29 0 0 1 2 10 79 45 5 11 ...
229.837 23.7693 0.921754 1.80754 0 0 0 1 41 31 0 0 141 45 0 0 ...
232.362 24.0691 1.0578 1.65689 11 1 0 16 134 11 0 0 106 21 15 23 ...
281.256 25.5857 1.04879 2.01664 10 4 1 8 14 2 1 6 88 13 0 0 ...
...
...
```

trong đó mỗi hàng chứa toa độ, tỷ lệ và góc xoay cho mỗi điểm đặc trưng là bốn giá trị đầu tiên, theo sau đó là 128 giá trị của mô tả tương ứng. Bộ mô tả được biểu diễn với các giá trị nguyên thô và không được chuẩn hóa. Đôi khi đây là điều chúng ta muộn muốn khi so sánh các vùng đặc trưng với nhau. Ví dụ trên cho thấy ví trị X đặc trưng là 8 đặc trưng đầu tiên được tìm thấy trong một hình ảnh.

Lưu ý rằng hai hàng đầu tiên có cùng tọa độ nhưng góc xoay khác nhau. Điều này có thể xảy ra một số hướng mạnh được tìm thấy tại cùng một điểm đặc trưng.

Dưới đây, cách đọc các đặc trưng vào mảng NumPy từ một tệp đầu ra như trình bày ở trên. Thêm hàm này vào `sift.py`.

```
def read_features_from_file(filename):
    """Read feature properties and return
    in matrix form."""

    f = loadtxt(filename)
    return f[:,:4], f[:,4:]  # feature
    locations, descriptors
```

Ở đây, NumPy `loadtxt()` đã được sử dụng.

Nếu sử dụng gói nguồn mở là phiên bản Python, việc viết kết quả trở lại vào các tệp tin có thể hữu hơn đó là sử dụng hàm này. Hãm bên dưới thực hiện điều này bằng NumPy, `savetxt()`.

```
def write_features_to_file(filename,locs,desc):
    """Save feature location and descriptor
    to file."""
    savetxt(filename,hstack((locs,desc)))
```

Phần này đã sử dụng hàm `hstack()` sắp xếp theo chiều ngang hai mảng bằng cách đặt hàng sao cho phần mô tả đi sau các vị trí trên mỗi hàng.

Sau khi có được các điểm đặc trưng, việc hiển thị để quan sát trực quan kết quả của chúng ta thì cần thêm `plot_features()` như bên dưới vào tệp `sift.py`.

```
def plot_features(im,locs,circle=False):
    """Show image with features. input: im
    (image as array), locs (row, col, scale,
    orientation of each feature)."""

    def draw_circle(c,r):
        t = arange(0,1.01,.01)*2*pi
        x = r*cos(t) + c[0]
        y = r*sin(t) + c[1]
        plot(x,y,'b',linewidth=2)

    imshow(im)
    if circle:
        for p in locs:
            draw_circle(p[:2],p[2])
    else:
        plot(locs[:,1],locs[:,0],'ob')
    axis('off')
```

Đoạn chương trình này sẽ vẽ vị trí của các điểm đặc trưng SIFT dưới dạng các chấm màu xanh được phủ lên trên hình ảnh. Nếu vòng tròn tham số được đặt bằng True, mỗi điểm sẽ được vẽ như một vòng tròn bằng tỷ lệ của đặc trưng được vẽ thay vì sử dụng hàm trợ giúp `draw_circle()`.

Các lệnh sau đây sẽ tạo ra một đồ thị giống như trong Hình 2.4b với các vị trí đặc trưng SIFT được hiển thị. Để thấy sự khác biệt so với các góc Harris, các đặc trưng góc Harris cũng được hiển thị trên cùng một hình ảnh bên phải (Hình 2.4c). Như có thể thấy hai thuật toán cho ra vị trí các điểm đặc trưng có thể khác nhau.

```
import sift

imname = 'empire.jpg'
im1 = array(Image.open(imname).convert('L'))
sift.process_image(imname,'empire.sift')
l1,d1 =
    sift.read_features_from_file('empire.sift')

figure()
gray()
sift.plot_features(im1,l1,circle=True)
show()
```

5) Đối sánh các mô tả của điểm đặc trưng: Một tiêu chí bên vững của Lowe giúp so khớp một điểm đặc trưng trong hình ảnh này với một đặc trưng trong một hình ảnh khác là sử dụng tỷ số khoảng cách của hai đặc trưng gần nhất. Điều này đảm bảo rằng chỉ các đặc trưng độc đặc biệt so với các đặc trưng khác trong hình ảnh mới được sử dụng. Do đó, số lượng các kết quả đối sánh sai được giảm xuống. Thêm hàm `match()` vào `sift.py` trong đoạn chương trình bên dưới.

```
def match(desc1,desc2):
    """For each descriptor in the first image,
    select its match in the second image.
    input: desc1 (same for first image),
           desc2 (same for second image)."""

    desc1 = array([d/linalg.norm(d) for d in
    desc1])
    desc2 = array([d/linalg.norm(d) for d in
    desc2])

    dist_ratio = 0.6
    desc1_size = desc1.shape

    matchscores =
    zeros((desc1_size[0],1),'int')
    desc2t = desc2.T  # precompute matrix
    transpose
```

```
    for i in range(desc1_size[0]):
        dotprods = dot(desc1[i,:],desc2t)
        dotprods = 0.9999*dotprods
        # inverse cosine and sort, return
    index
        indx = argsort(arccos(dotprods))

        # check if nearest neighbor has angle
    less
        # than dist_ratio times 2nd
        if arccos(dotprods)[indx[0]] < \
           dist_ratio *
    arccos(dotprods)[indx[1]]:
            matchscores[i] = int(indx[0])

    return matchscores
```

Hàm này sử dụng góc giữa các vector mô tả làm thước đo khoảng cách. Điều này chỉ có ý nghĩa sau khi đã chuẩn hóa các vector thành đơn vị chiều dài. Vì so khớp theo một hướng, có nghĩa là khớp từng đặc trưng với tất cả các đặc trưng khác nhau và sẽ không dịch nghĩa là hai vector mô tả. Những vector này chứa các điểm đặc trưng trong hình ảnh thứ hai dễ như vậy không cần phải lặp lại thao tác này cho từng đặc trưng.

Để tăng thêm độ bền vững của việc đối sánh hay so khớp các điểm đặc trưng, có thể đảo ngược quy trình và so khớp theo cách khác (từ các đặc trưng trong hình 2 đến hình 1) và chỉ giữ các đặc trưng thỏa mãn các tiêu chí theo cả hai cách (giống như những gì đã làm cho các điểm Harris). Hàm `match_twosided()` thực hiện điều này như sau:

```
def match_twosided(desc1,desc2):
    """Two-sided symmetric version of
    match()."""

    matches_12 = match(desc1,desc2)
    matches_21 = match(desc2,desc1)

    ndx_12 = matches_12.nonzero()[0]

    # remove matches that are not symmetric
    for n in ndx_12:
        if matches_21[int(matches_12[n])] !=
    n:
            matches_12[n] = 0

    return matches_12
```

Để vẽ các kết quả so khớp điểm đặc trưng, có thể sử dụng các hàm tương tự được sử dụng trong `harris.py`. Chỉ cần sao chép các hàm `appendimages()` và `plot_matches()` và thêm chúng vào `sift.py`, để thuận tiện cũng có thể nhập `harris.py` và sử dụng chúng từ đó nếu muốn.

Hình 2.5 và 2.6 cho thấy một số ví dụ về các điểm đặc trưng SIFT được phát hiện trong các cặp hình ảnh cùng với các kết hợp tìm cặp được trả về từ hàm `match_twosided()`.

Hình 2.7 cho thấy một ví dụ khác về các trưng phù hợp được tìm thấy trong hai hình ảnh sử dụng `match()` và `match_twosided()`. Như có thể thấy, sử dụng điều kiện khớp cả hai chiều (hai mặt) loại bỏ các kết quả khớp tốt (tuy nhiên có một số kết quả khớp chính xác cũng bị loại bỏ).

Với việc phát hiện và kết hợp các điểm đặc trưng, chúng ta đã có tất cả những gì cần thiết cho số sung các ràng buộc hình học trên các đặc trưng tương đồng để lọc ra những cặp điểm không so khớp chính và ứng dụng vào các vị dụ có như tạo ra ảnh toàn cảnh tự động, ước tính vị trí và góc camera và tính toán cấu trúc mô hình ba chiều của đối tượng.

IV. Tài liệu tham khảo

Tài liệu

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In European Conference on Computer Vision, 2006.

[2] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23:2001, 2001.

[3] Gary Bradski and Adrian Kaehler. Learning OpenCV. O'Reilly Media Inc., 2008.

[4] Martin Byröd. An optical sudoku solver. In Swedish Symposium on Image Analysis, SSBA, 2007.

[5] Antonin Chambolle. Total variation minimization and a class of binary mrf models. In Energy Minimization Methods in Computer Vision and Pattern Recognition, Lecture Notes in Computer Science, pages 136-152. Springer Berlin, 2005.

[6] T. Chan and L. Vese. Active contours without edges. IEEE Trans. Image Processing, 10(2):266-277, 2001.

[7] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[8] D. Cremers, T. Pock, K. Kolev, and A. Chambolle. Convex relaxation techniques for segmentation, stereo and multiview reconstruction. In Advances in Markov Random Fields for Vision and Image Processing. MIT Press, 2011.

[9] Nello Cristianini and John Shawe-Taylor. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press,2000.

[10] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion in Proceedings of the 13th Scandinavian Conference on Image Analysis, pages 363-370, 2003.

[11] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications-of-the-ACM, 24(6):381-95, 1981.

[12] C. Harris and M. Stephens. A combined corner and edge detector. In Proc. Alvey Conf., pages 189-192, 1988.

[13] R. I. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[14] Richard Hartley. In defense of the eight-point algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19:580-593, 1997.

[15] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. Artifical Intelligence, 17:185-203, 1981.

[16] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26:65-81, 2004.

[17] David G. Lowe. Object recognition from local scale-invariant features. In International Conference on Computer Vision, pages 1150-1157, 1999.

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

[19] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision, pages 674-679, 1981.

[20] Mark Lutz. Learning Python. O'Reilly Media Inc., 2009.

Acknowledgment

Các chương trong cuốn sách này ra đời với mục đích giúp người học để dàng tiếp cận và thực hành các giác máy tính. Cơ sở lý thuyết và thuật toán cơ bản được trình bày theo cách đơn giản nhất để sinh viên, nhà nghiên cứu và những đối tượng khác có cùng đam mê về lĩnh vực này có thể đạng thực hành trên những ứng dụng cụ thể. Một văn ứng dụng cụ thể có thể

kể ra như truy vấn ảnh, phân loại ảnh, xây dựng mô hình ba chiều, điều hướng cho Robot, phân tích hình ảnh y tế, v.v.