**Full-name: Phan Thi My Dung**

**Lecturer: Lee Chil Woo**

**Class: Advance in Pattern Recognition**

**FACE RECOGNITION USING SIFT FEATURE**

1. **Flowchart**

Sift descriptor

Input Face Image

Images in Database

Matching

Find the best matching

For each image in database

1. **Implementation**

**2.1. Reference**

### Source code: SIFT Keypoint Detector by [David Lowe](http://www.cs.ubc.ca/~lowe/home.html): <http://www.cs.ubc.ca/~lowe/keypoints/>

* + The demo software uses PGM format for image input. It can output keypoints and all information needed for matching them to a file in a simple ASCII format.
  + Features are extracted from each of the two images, and lines are drawn between features that have close matches.
* Paper: David G. Lowe, **"Distinctive image features from scale-invariant keypoints,"** International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

**2.2. Wrapper source code for face recognition**

* Database: face images which are closed on face region.
* Run demo\_faceReg.m for running a testing of face recognition.

function demo\_faceReg

% indicate test images

tDir = 'test/';

listTest = dir(strcat(tDir,'\*.\*p\*'));

mT = size(listTest,1);

% indicate database

dbDir = 'database/';

listDB = dir(strcat(dbDir,'\*.\*p\*'));

mDB = size(listDB,1);

for i=1:mT

tFile = strcat(tDir,listTest(i).name);

for j=1:mDB

dbFile = strcat(dbDir,listDB(j).name);

num(j) = match(tFile,dbFile,0);

end

% find the best matching

[vmax,nmax] = max(num);

dbFile = strcat(dbDir,listDB(nmax).name);

match(tFile,dbFile,1);

end

end

* More detail about data and experimental result, please refer to experimental result part.
* Sift descriptor:
  + The "sift" command calls the appropriate binary to extract SIFT features (under Linux or Windows) and returns them in matrix form.

function [image, descriptors, locs] = sift(imageFile)

% Load image

image = imread(imageFile);

% If you have the Image Processing Toolbox, you can uncomment the following

% lines to allow input of color images, which will be converted to grayscale.

%if isrgb(image)

image = rgb2gray(image);

%end

[rows, cols] = size(image);

% Convert into PGM imagefile, readable by "keypoints" executable

f = fopen('tmp.pgm', 'w');

if f == -1

error('Could not create file tmp.pgm.');

end

fprintf(f, 'P5\n%d\n%d\n255\n', cols, rows);

fwrite(f, image', 'uint8');

fclose(f);

% Call keypoints executable

if isunix

command = '!./sift ';

else

command = '!siftWin32 ';

end

command = [command ' <tmp.pgm >tmp.key'];

eval(command);

% Open tmp.key and check its header

g = fopen('tmp.key', 'r');

if g == -1

error('Could not open file tmp.key.');

end

[header, count] = fscanf(g, '%d %d', [1 2]);

if count ~= 2

error('Invalid keypoint file beginning.');

end

num = header(1);

len = header(2);

if len ~= 128

error('Keypoint descriptor length invalid (should be 128).');

end

% Creates the two output matrices (use known size for efficiency)

locs = double(zeros(num, 4));

descriptors = double(zeros(num, 128));

% Parse tmp.key

for i = 1:num

[vector, count] = fscanf(g, '%f %f %f %f', [1 4]); %row col scale ori

if count ~= 4

error('Invalid keypoint file format');

end

locs(i, :) = vector(1, :);

[descrip, count] = fscanf(g, '%d', [1 len]);

if (count ~= 128)

error('Invalid keypoint file value.');

end

% Normalize each input vector to unit length

descrip = descrip / sqrt(sum(descrip.^2));

descriptors(i, :) = descrip(1, :);

end

fclose(g);

* Matching
  + The "match" function is given two image file names. It extracts SIFT features from each image, matches the features between the two images, and displays the results.

function num = match(image1, image2, show)

% Find SIFT keypoints for each image

[im1, des1, loc1] = sift(image1);

[im2, des2, loc2] = sift(image2);

% For efficiency in Matlab, it is cheaper to compute dot products between

% unit vectors rather than Euclidean distances. Note that the ratio of

% angles (acos of dot products of unit vectors) is a close approximation

% to the ratio of Euclidean distances for small angles.

%

% distRatio: Only keep matches in which the ratio of vector angles from the

% nearest to second nearest neighbor is less than distRatio.

distRatio = 0.6;

% For each descriptor in the first image, select its match to second image.

des2t = des2'; % Precompute matrix transpose

for i = 1 : size(des1,1)

dotprods = des1(i,:) \* des2t; % Computes vector of dot products

[vals,indx] = sort(acos(dotprods)); % Take inverse cosine and sort results

% Check if nearest neighbor has angle less than distRatio times 2nd.

if (vals(1) < distRatio \* vals(2))

m\_match(i) = indx(1);

else

m\_match(i) = 0;

end

end

if (show == 1)

% Create a new image showing the two images side by side.

im3 = appendimages(im1,im2);

% Show a figure with lines joining the accepted matches.

figure('Position', [100 100 size(im3,2) size(im3,1)]);

colormap('gray');

imagesc(im3);

hold on;

cols1 = size(im1,2);

for i = 1: size(des1,1)

if (m\_match(i) > 0)

line([loc1(i,2) loc2(m\_match(i),2)+cols1], ...

[loc1(i,1) loc2(m\_match(i),1)], 'Color', 'c');

end

end

hold off;

end

num = sum(m\_match > 0);

fprintf('Found %d matches.\n', num);

1. **Experimental result**

Table 1: Face database

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

Table 2: Test Images

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

* Look at the experimental result below:
  + For scaled image, sift descriptor can describe matching features very well.
  + For rotated image, 2 images have a little matching points.
  + So, sift descriptor is not good to applying for face recognition because there are a little sift feature on face

Table 3: Result of face recognition. Lines connect matching points between 2 images

|  |  |
| --- | --- |
| **Test case** | **Input image (Left) – Best Matching (Right)** |
| Scaled Input Image |  |
| Rotated Input Image |  |
| Scaled Input Image |  |
| Scaled Input Image |  |
| Rotated Input Image |  |