Homework 4 Writeup

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- Use as many pages as you need, but err on the short side If you feel you only need to write a short amount to meet the brief, th
- **Please make this document anonymous.**

In the beginning...

Implementing this assignment is divided into three parts. First of all, getting interest points by calculating Harris value, secondly getting descriptors by using SIFT-like methods, and lastly matching features by calculating Euclidean distance and confidence. First of all, calculating the Harris value is useful to find out the interest points. As written below in equation 1, Harris matrix is given using the partial determinants. Harris value could be calculated as given under from equation 2.

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{1}$$

$$h = det(A) - \alpha * tr^2(A) \tag{2}$$

As suggested from the textbook by Szleski, alpha is recommended as 0.06

To discuss about SIFT-like progress, dividing into 16 cells and calculate each of the gradient values for 8 directions, adding up from all values inside the cell is needed.

For understanding the implementation detail written below, we should have a look on Sobel filter. Sobel filter is used to approximately calculate the derivatives for both x and y axis, easily can both calculate magnitude and angle.

To add on, we should use NNDR (Nearest Neighbor Distance Ratio) to calculate as confidences. The equation is written as below, d1 is the nearest neighbor distance, d2 is the second nearest neighbor distance.

$$NNDR = d_1/d_2 \tag{3}$$

Interesting Implementation Detail

- get interest points

I made up with two different Gaussian filters, large one with width of 9 and sigma of 2, small one with width of 3 and sigma of 1. Two Gaussian filters would improve blurring. To calculate both gradients of x and y axis, I used sobel filters and made convolution. It is implemented as below

```
% small, large gaussian filter -> sigma 2, 1 for each
   large, small filter
large_gaussian = fspecial('gaussian',9,2);
small_gaussian = fspecial('gaussian',3,1);

% gradient values of small-gaussian-filtered image
new_image = imfilter(image,small_gaussian);
image_db=im2double(new_image);
sobely= [1 ,0 ,-1; 2,0,-2; 1, 0 ,-1];
sobelx= [1,2,1; 0,0, 0; -1, -2 ,-1];
x_grad = conv2(image_db,sobelx,'same');
y_grad = conv2(image_db,sobely,'same');

% calculate grads : filter with large gaussian filter
xy_grad = y_grad .* x_grad;
xy_grad = imfilter(xy_grad, large_gaussian);
xx_grad = x_grad .* x_grad;
xx_grad = imfilter(xx_grad, large_gaussian);
yy_grad = y_grad .* y_grad;
yy_grad = imfilter(yy_grad, large_gaussian);
```

- get descriptors

To calculate gradient values for all 8 directions, I made 8 different sobel filters. After it, filtering with Gaussian filter blurred the picture and made it better to calculate.

```
% Use sobel filter to rotate image in certain direction
   and calculate gradient
filter = [];
m = fspecial('sobel');
filter(:,:,1) = m;
for i=2:8
    % rotating sobel filter 45 degrees from former one
    m = [m(4) m(7) m(8); m(1) m(5) m(9); m(2) m(3) m(6)];
    filter(:,:,i) = m;
end

% new_image is an image containing 8 copies of images
   filtered by 8 sober
```

```
% filters
new_image = zeros(image_size(1),image_size(2),8);
for i=1:8
    new_image(:,:,i) = imfilter(image,filter(:,:,i));
end

% filter with gaussian, sigma = half_width
gauss_filter = fspecial('gaussian', [half_width,
   half_width], half_width);
new_image = imfilter(new_image, gauss_filter);
```

- match features

As mentioned before, matching features will calculate confidences using NNDR, used find function and flipped it after finding.

```
% confidences > 1
[B,I] = sort(K,2);
neighbor1 = B(:,1);
neighbor2 = B(:,2);
confidences = neighbor1 ./ neighbor2;

% use find
i = find(confidences)
s = size(i);
matches = zeros(s(1),2);
matches(:,1) = i;
matches(:,2) = I(i);

% confidences in correct value
confidences = 1./confidences(i)
```

A Result

Three pictures after three functions are described under as figure and the accuracy was 86%, 96%, 3% for the best 100 points.

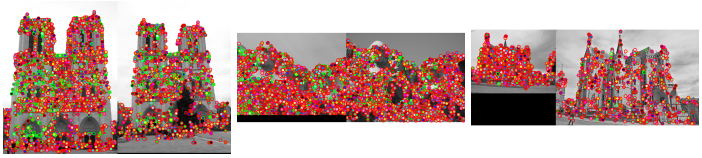

Figure 1: *Left:* Notre Dame de Paris *Center:* Mount Rushmore *Right:* Gaudi's Episcopal Palace