Homework X 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.**

Files

1. get_interest_point.m

   input : image, descriptor_window_image_width.
   output : x, y, confidence, (scale), (orientation).

1. get_descripotrs.m

   input : image, x, y, descriptor_window_image_width.
   output : features.

1. match_features.m

   input : features1, features2.
   output : matches, confidences.

Code

1. my_imfilter.m

```
function [x, y, confidence, scale, orientation] =
    get_interest_points(image,
    descriptor_window_image_width)
B_gau = fspecial('gaussian',[3,3],1);
image = imfilter(image,B_gau);


dx = [-1 0 1;
```

```
    -1 0 1;
    -1 0 1];
dy = dx.';

Ix = imfilter(image,dx,'same','conv');
Iy = imfilter(image,dy,'same','conv');

% gradients near the edges
Ix([(1:descriptor_window_image_width) end-descriptor_window_image_width+(1:descriptor_window_image_width)],:) = 0;
Ix(:, [(1:descriptor_window_image_width) end-descriptor_window_image_width+(1:descriptor_window_image_width)]) = 0;
Iy([(1:descriptor_window_image_width) end-descriptor_window_image_width+(1:descriptor_window_image_width)],:) = 0;
Iy(:, [(1:descriptor_window_image_width) end-descriptor_window_image_width+(1:descriptor_window_image_width)]) = 0;

large_gaussian = fspecial('Gaussian', [25 25], 2);
Ix2 = imfilter(Ix.^2, large_gaussian);
Ixy = imfilter(Ix.*Iy, large_gaussian);
Iy2 = imfilter(Iy.^2, large_gaussian);

alpha = 0.05;
C = Ix2.^2 - Ixy.^2 - alpha.*(Ix2+Iy2).^2;
threshold = C > mean2(C);

comp = bwconncomp(threshold);
x = zeros(comp.NumObjects, 1);
y = zeros(comp.NumObjects, 1);

confidence = zeros(comp.NumObjects, 1);
width = comp.ImageSize(1);
for i=1:(comp.NumObjects)
    pix_i = comp.PixelIdxList{i};
    pix_val = C(pix_i);
    [max_val, max_id] = max(pix_val);
    x(i) = floor(pix_i(max_id)/ width);
    y(i) = mod(pix_i(max_id), width);
    confidence(i) = max_val;
end
```

1. At first, it blurs on the image.

2. Compute image derivatives. We can't use the imgradient() function, so I use [-1 0 1].
3. Compute M and C with Gaussian filter.
4. Threshold on C to pick high cornerness.

1. get_descripotrs.m

```
function [features] = get_features(image, x, y, descriptor_window_image_width)
features = zeros(size(x,1), 128, 'single');


S_gau = fspecial('Gaussian', [descriptor_window_image_width descriptor_window_image_width], 1);
L_gau = fspecial('Gaussian', [descriptor_window_image_width descriptor_window_image_width], descriptor_window_image_width/2);

image = imfilter(image,S_gau);

dx = [-1 0 1;
    -1 0 1;
    -1 0 1];
dy = dx.';

Ix = imfilter(image,dx,'same','conv');
Iy = imfilter(image,dy,'same','conv');

octant = @(x,y) (ceil(atan2(y,x)/(pi/4)) + 4);
orient = arrayfun(octant, Ix, Iy);
mag = hypot(Ix, Iy);
widthp4 = descriptor_window_image_width/4;
forend = size(x,1);
for i = 1:forend
    x_range = (x(i) - 2*widthp4): (x(i) + 2*widthp4-1);
    y_range = (y(i) - 2*widthp4): (y(i) + 2*widthp4-1);
    f_mag = mag(y_range, x_range);
    f_mag = f_mag.*L_gau;
    f_orient = orient(y_range, x_range);
    for j = 0:3
        for k = 0:3
            cell1 = f_orient(j*4+1:j*4+4, k*4+1:k*4+4);
            cell2 = f_mag(j*4+1:j*4+4, k*4+1:k*4+4);
            for l = 1:8
```

```
                f = cell1 == l;
                features(i, (j*32 + k*8) + l) = sum(sum(cell2(f)));
            end
        end
    end
end
features = diag(1./sum(features,2))*features;
```

1. At first, compute Ix, Iy, with [-1 0 1].
2. Compute SIFT descriptor Extraction.(with 4x4 orients and mag.)

1. match_features.m

```
function [matches, confidences] = match_features(features1, features2)
threshold = 0.70;
[x1,y1] = size(features1);
[x2,y2] = size(features2);
dist = zeros(x1,x2);
for i = 1:x1
    for j =1:x2
        dist(i,j) = dot((features1(i,:) - features2(j,:)),(features1(i,:) - features2(j,:))).^(1/2);
    end
end

[s1,s2] = sort(dist,2);
NN1 = s1(:,1);
NN2 = s1(:,2);
confidences = NN1 ./ NN2;
i = find(confidences < threshold);
size_i = size(i);
matches = zeros(size_i(1),2);
matches(:,1) = i;
matches(:,2) = s2(i);
confidences = 1./confidences(i);

% Sort the matches so that the most confident onces are at the top of the
% list. You should probably not delete this, so that the evaluation
% functions can be run on the top matches easily.
[confidences, ind] = sort(confidences, 'descend');
```

```
matches = matches(ind,:);
```

1. At first, compute distance between two features.
2. Compare closest Nearest Neighbor and second closest feature vetor neighbor.(NN1 and NN2)

Result

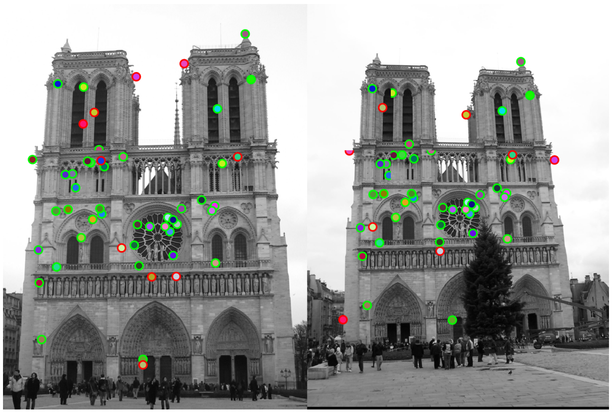

Figure 1: eval_ND

1. Notre Dame de Paris
Uniqueness: Pre-merge: 51 Post-merge: 51
Total: Good matches: 41 Bad matches: 10
Accuracy: 80.39% (on all 51 submitted matches)
Accuracy: 41% (on first 100 matches sorted by decreasing confidence)

2. Mount Rushmore

Figure 2: eval_MR

Uniqueness: Pre-merge: 77 Post-merge: 77
Total: Good matches: 68 Bad matches: 9
Accuracy: 88.31% (on all 77 submitted matches)
Accuracy: 68% (on first 100 matches sorted by decreasing confidence)

3. Gaudi's Episcopal Palace

Figure 3: eval_EG

Uniqueness: Pre-merge: 2 Post-merge: 2
Total: Good matches: 0 Bad matches: 2
Accuracy: 0% (on all 2 submitted matches)
Accuracy: 0% (on first 100 matches sorted by decreasing confidence)