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.**

Algorithm Implementation

Algorithms of this project is implemented in three parts.

0.1 Interest Point Detection

The interest points of a pair of images is found using Harris corner detector. Full code is shown below. In line 7, a Gaussian filter is created. The gradient of the filter is calculated in lines 9-14. They are then convolbed with the image to find the image gradients. The gradient of the filter is calculated instead of that of the image to reduced the effects of noise in the image. It is still the same operation as finding the gradient of the blurred image directly. In lines 20-22, image gradients are convolved with the Gaussian filter to find the second moment matrix M. In line 24, the cornerness is calculated using the formula $C = det(M) - \alpha trace(M)^2$. In line 26, the indices of pixels at which the cornerness is less than a threshold is returned.

```
1  function [x, y, confidence, scale, orientation] =
       get_interest_points(image,
       descriptor_window_image_width)
2
3  alpha = 0.05;
4  threshold = -0.011;
5
6  %apply double derivative to the filter first then apply
       to the image, to remove the effects of noise
7  gauss_filter = fspecial('Gaussian', 11, 1);
8
9  Dx = imderivative(gauss_filter, [1 0]);
10 Dy = imderivative(gauss_filter, [0 1]);
```

```
11 Dxy = imderivative(gauss_filter, [1 1]);
12
13 Dx2 = Dx.*Dx;
14 Dy2 = Dy.*Dy;
15
16 Ix2 = imfilter(image, Dx2, 'symmetric', 'same', 'conv');
17 Iy2 = imfilter(image, Dy2, 'symmetric', 'same', 'conv');
18 Ixy = imfilter(image, Dxy, 'symmetric', 'same', 'conv');
19
20 Gx2 = imfilter(Ix2, gauss_filter, 'symmetric', 'same', '
      conv');
21 Gy2 = imfilter(Iy2, gauss_filter, 'symmetric', 'same', '
      conv');
22 Gxy = imfilter(Ixy, gauss_filter, 'symmetric', 'same', '
      conv');
23
24 C = Gx2.*Gy2-(Gxy.*Gxy)-alpha*((Gx2+Gy2).*(Gx2+Gy2));
25
26 [y,x] = find(C<threshold);
27
28 end
29
30 function I = imderivative(img,direction)
31 It = imtranslate(img,-direction,'OutputView','full');
32 Ip = padarray(img,flip(direction),0,'pre');
33 I = It-Ip;
34 I = I(1:size(I,1)-direction(2),1:size(I,2)-direction(1));
35 end
```

0.2 Descriptor Calculation

Desriptors at each interest point is calculated into a $4 * 4 * 8 = 128$ dimension feature. Code is shown below. In lines 3-10, blurred image gradients are calculated by convolving the gradients of a Gaussian filter to the image. Then for each interest point, a patch of image of size $16 * 16$ in x and y gradients is found in lines 16 and 17. The orientation at each of the 16 pixels is calculated in line 19 and mapped to range $[0°, 360°]$. The result is divided by $45°$ to convert to bin index. For each of the 16 $4 * 4$ patches inside the $16 * 16$, the number of pixels at each bin slot is calculated in line 25. It was concatenated at the i'th feature at $features(i)$ in line 26. For each of the $1 * 128$ feature, it was normalized by dividing it by the maximum element in lines 30-31. The final array of features was raised to a power of 0.8 following the trick provided in the comment of the *get_descriptors.m* file at the top.

```
1 function [features] = get_features(image, x, y,
      descriptor_window_image_width)
```

```
2
3  features = zeros(size(x,1), 128, 'single');
4  gauss_filter = fspecial('Gaussian', 4, 0.6);
5
6  Dx = imderivative(gauss_filter, [1 0]);
7  Dy = imderivative(gauss_filter, [0 1]);
8
9  Ix = imfilter(image, Dx, 'symmetric', 'same', 'conv');
10 Iy = imfilter(image, Dy, 'symmetric', 'same', 'conv');
11 for i = 1:size(x,1)
12     if(x(i)<8 || y(i)<8 || x(i)+8>size(Ix,2) || y(i)+8>
          size(Iy,1))
13         % skip out of image patches
14         continue
15     end
16     patch_Ix = Ix((y(i)-7):(y(i)+8),(x(i)-7):(x(i)+8));
17     patch_Iy = Iy((y(i)-7):(y(i)+8),(x(i)-7):(x(i)+8));
18     %computing the tangent in degrees
19     orientation = radtodeg(atan2(patch_Iy,patch_Ix))+180;
20     %computing the 8 quandrants
21     quads = ceil(orientation/45);
22     for m = 1:4
23         for n = 1:4
24             patch4 = quads(m*4-3:m*4, n*4-3:n*4);
25             qcounts = histcounts(patch4,0.5:1:8.5);
26             features(i,(4*(m-1)+(n-1))*8+1:(4*(m-1)+(n-1)
                  )*8+8) = qcounts;
27         end
28     end
29     %normalize each feature
30     [mf,~] = max(features(i,:));
31     features(i,:) = features(i,:)/mf;
32 end
33 features = features.^0.8;
34 end
```

0.3 Feature Matching

Features calculated from a pair of images by *get_descriptors.m* is matched using the *NNDR* method. Code is shown below. To match features, distance from every feature in one image to every feature in another image should be calculated, called the pairwise distance. The pairwise distance between the two feature arrays was calculated using a custom function called *pairwise_distance* in lines 31-39. Internally, the norm function of MATLAB was used to calculate sidtance. Such distances were sorted in ascending order to move close features to the front. The *NNDR* value is calculated using the formula

$\frac{\|D_A - D_B\|}{\|D_A - D_C\|}$ in line 6. The matched indices are filled in lines 8-11 using the sorted index. In line 13, the good indices are defined as indices at which *NNDR* is less than a threshold. The confidence value was calculated by the element-wise inverse because a low *NNDR* means a confident match. Then the confidence and matches are sorted and taken only the top 100 because the top 100 is the minimum requirement for grading.

```
1  function [matches, confidences] = match_features(
       features1, features2)
2
3  num_features = min(size(features1, 1), size(features2,1))
       ;
4  pd = pairwise_distance(features1, features2);
5  [sortdist, sortindex] = sort(pd,2,'ascend');
6  nndr = sortdist(:,1)./sortdist(:,2);
7
8  matches = zeros(size(features1, 1), 2);
9
10 matches(:,1) = 1:size(features1);
11 matches(:,2) = sortindex(:,1);
12
13 good_indices = find(nndr<0.93);
14
15 matches = matches(good_indices,:);
16
17 confidences = 1./nndr(good_indices);
18
19 [confidences, ind] = sort(confidences, 'descend');
20
21 % get top 100 indices
22 confidences = confidences(1:min(100,size(confidences)));
23 ind = ind(1:min(100,size(matches)));
24 matches = matches(ind,:);
25
26 end
27
28 function pd = pairwise_distance(X,Y)
29     pd = zeros(size(X,1),size(Y,1));
30     for m=1:size(X,1)
31         for n=1:size(Y,1)
32             %sum of squares of elements
33             pd(m,n) = norm(X(m,:)-Y(n,:));
34         end
35     end
36 end
```

Results

The interest points are matched in Notre Dame, Mount Rushmore, and Episcopal Gaudi images in Figures 1, 2 and 3, respectively. Because scale and orientation invariance of feature matching is not implemented, accuracy in Mount Rushmore and Episcopal Gaudi images are not high.

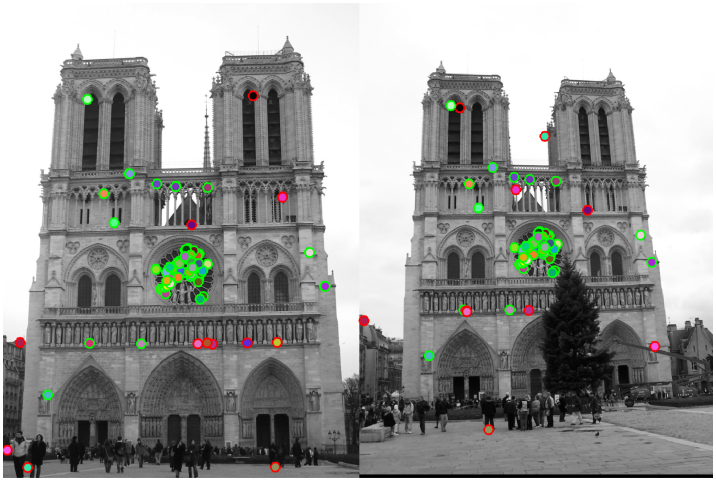

Figure 1: Notre Dame 87%

Figure 2: Mount Rushmore 37%

When trying different parameters such as filter size, sigma and thresholds of feature matching, I realized it influences accuracy of the matching. Time was also measured to make sure the computation does not exceed the time limit. Relevant code shown below. As shown in Figure 4, maximum accuracy is obtained at filter size of 4. Time was not influenced by the filter size seemingly because the number of features did not change. Figure 5 compares different amount of sigma while the window size is fixed at 4. The maximum accuracy is therefore attained at sigma of 0.6 and filter size of 4. Time was

Figure 3: Episcopal Gaudi 5%

not influenced by the sigma either.

```
size_range = 3:30;
%size_range = 0.1:0.1:1.5;
i = 1;
for s=size_range
        tic;
        % 2) Create feature descriptors at each interest
           point. Szeliski 4.1.2
        [image1_features] = get_descriptors(image1, x1,
           y1, descriptor_window_image_width, s);
        [image2_features] = get_descriptors(image2, x2,
           y2, descriptor_window_image_width, s);

        % 3) Match features. Szeliski 4.1.3
        [matches, confidences] = match_features(
           image1_features, image2_features);

        % Evaluate matches
        [~,~,accAll,accMPEND] = evaluate_correspondence(
           image1, image2, eval_file, scale_factor, ...
                x1, y1, x2, y2, matches, confidences, ...
                maxPtsEval, visualize, 'eval_ND.png' );
        accs(s) = accAll
        ts(s) = toc
        i = i+1;
end
```

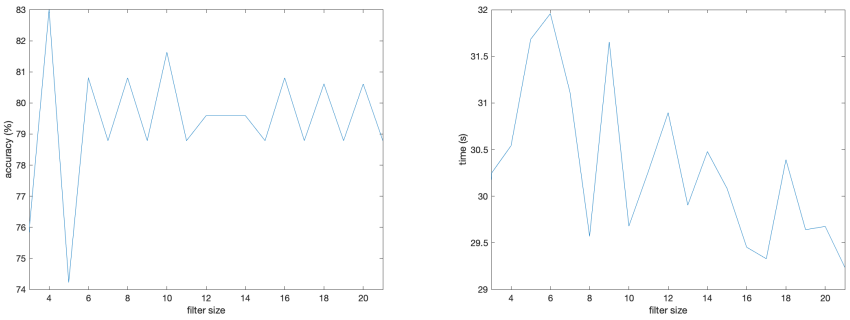

Figure 4: Size vs Accuracy (left) and Time (right)

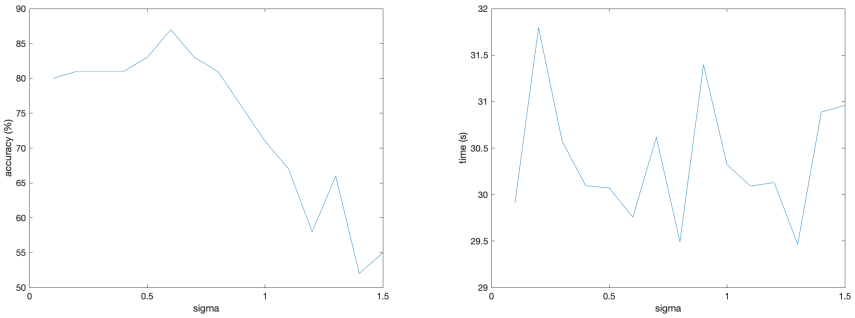

Figure 5: Size vs Accuracy (left) and Time (right)