CS 696 - Applied Computer Vision

INTRODUCTION :

This project includes 3 parts : Interest point detection, Local feature description and Feature matching. For the interest point detection, Harris corner detector is implemented. It will detect some Interest points that will be used for describing and matching features later.

For the local feature descriptor, a SIFT-like local feature description is implemented. It will generate Local feature description for each interest points that are detected in part 1.

For the feature matching, nearest neighbor distance ratio test method is implemented. it will try to match local features that are generated from part2. In this way, the program try to find all the possible matching point pair from two or more similar images

PART 1 : Interest Point Detection (get_interest_points.m)

To find the interesting points, we first need to preprocess the image pixels using Harris Detector. The method I used there is to filter the original image with the derivatives using sobel filter and filter with large gaussian.

```
%1. Compute derivatives using sobel filter

s = [1, 0, -1; 2, 0, -2; 1, 0, -1];
dx = imfilter(image, s);
dy = imfilter(image, s');
dx2 = imfilter(dx, s);
dy2 = imfilter(dy, s');
dxdy = imfilter(dx, s’);

% 2. Filter with large guassian

g = fspecial('gaussian', 25, 1.5);
Ix2 = imfilter(dx2, g);
Iy2 = imfilter(dy2, g);
Ixy = imfilter(dxdy, g);
```

The Harris's score of a point represents the distinction compared with its neighbors. Knowing Ix2, Iy2 and Ixy, I calculated Harris's score according to the following formula:

$$g(I_x^2)g(I_y^2)-[g(I_xI_y)]^2-\alpha[g(I_x^2)+g(I_y^2)]^2$$

To get rid of the influence of the border points, I masked the harris matrix with the border matrix. Applied the threshold on the masked harris matrix. The threshold is adaptive according to the mean value of the masked harris matrix.

```
% 3. Compute harris value

alpha = 0.06;
harris = (Ix2 .* Iy2) - axy .^ 2 - alpha .* (Ix2 + Iy2) .^ 2;

% 4. Suppress features close to edges
harris(1 : feature_width, :) = 0;
harris(end - feature_width : end, :) = 0;
harris(:, 1 : feature_width) = 0;
harris(:, end - feature_width : end) = 0;
```

Find the connected components and pick the local maxima of each component. The confidence is based on the harris score of each maxima.

```
% 5a. Find connected components
CC = bwconncomp(im2bw(harris, graythresh(harris)));

% 5b. Take the max from each component region
x = zeros(CC.NumObjects, 1);
y = zeros(CC.NumObjects, 1);
for i = 1 : CC.NumObjects
     region = CC.PixelIdxList{i};
     [~, ind] = max(harris(region));
     [y(i), x(i)] = ind2sub(size(harris), region(ind));
end
```

EXTRA CREDIT : Scaling

For scaling, sum up the distances and then divide the square of the number of points by the sum to get the inverse of the scale. The scale will be used later on the feature_width during getting features.

```
% 6. Scaling
% Calculate the inverse of the scale
sum = 0;
for ii = 1: length(x)
    for jj = 2 : length(x)
        sum = sum + ((x(ii)-x(jj))^2 + (y(ii)-y(jj))^2)^(1/2);
    end
end
scale = (length(x) ^ 2) / sum;
```

PART 2 : Local feature description (get_features.m)

Apply the SIFT Algorithm to calculate the feature of each interseting point. The first step is to construct a 4 * 4 grid, of which each unit has 8 direction bins. Each bin is a magnitude representing the intensity of that direction. Thus for each interesting point, its feature is a 1 * 128 vector. To emphasize the importance of the close neighbors, I applied a gaussian filter with the same size of filter_width/2

```
% for each keypoint
for i = 1 : length(x)
% Get window
window = image(y(i) - feature_width/2 : y(i) + feature_width/2 - 1, ...
x(i) - feature_width/2 : x(i) + feature_width/2 - 1);

% Get gradient of window
[gmag, gdir] = imgradient(window);

% Weigh magnitudes with gaussian
gmag_weighted = imfilter(gmag, fspecial('gaussian', 10, sqrt(feature_width/
2)));

% Transform to cells
gmag_cols = im2col(gmag_weighted, [feature_width/4, feature_width/4],
'distinct');
gdir_cols = im2col(gdir, [feature_width/4, feature_width/4], 'distinct');

% for each cell in 4x4 array
descriptor = zeros(1, 128);
for j = 1 : size(gdir_cols, 1)
    col = gdir_cols(:, j);
    [~, inds] = histc(col, angle_bins);
    buckets = zeros(1, 8);
% Sum magnitudes for each histogram bucket
    for k = 1 : length(inds)
       buckets(inds(k)) = buckets(inds(k)) + gmag_cols(k, j);
    end
% Compute start and end indices into descriptor
    start_ind = (j - 1) * 8 + 1;
    end_ind = (j - 1) * 8 + 8;
    descriptor(1, start_ind : end_ind) = buckets;
end

% Normalize descriptor
descriptor = descriptor / norm(descriptor);
features(i, :) = descriptor; % Add to features
end

% raise each element of feature matrix to number < 1
power = 0.8;
features = features .^ power;

end
```

PART 3: Feature matching (match_features.m)

For each image, the euclidean distance between each pair is calculated in feature space. The distance represents the similarity of the features and the ratio of the smallest distance and second small one represents the reliability. If the ratio is greater than the a threshold, its picked as a match.

```
% find the euclidian distance between each pair of descriptors in feature
space
      distances = zeros(num_features2, num_features1);
      for i = 1 : num_features1
          for j = 1 : num_features2
              distances(j, i) = norm(features1(i, :) - features2(j, :));
          end
      end

      threshold = 0.8;
      [sorted_distances, inds] = sort(distances);
% sort distances in ascending order

% calculate ratio of nearest neighbor to second nearest neighbor
% if above threshold, add to matches list and save ratio as confidence value
      for i = 1 : num_features1
          ratio = sorted_distances(1, i) / sorted_distances(2, i);
          if ratio < threshold
              matches(i, :) = [i, inds(1, i)];
              confidences(i) = 1 - ratio;
          end
      end

% keep only those that matched
      match_inds = find(confidences > 0);
      matches = matches(match_inds, :);
      confidences = confidences(match_inds);

% Sort the matches so that the most confident onces are at the top of the
list
      [confidences, ind] = sort(confidences, 'descend');
      matches = matches(ind, :);
```

RESULTS:

vis.jpg	eval.jpg

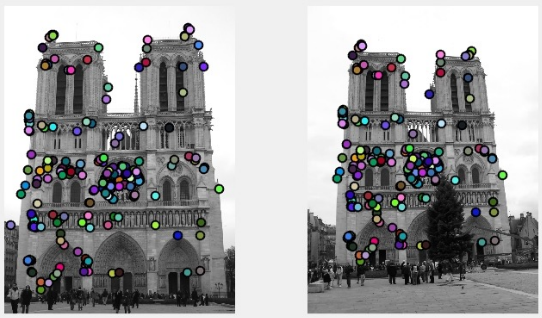
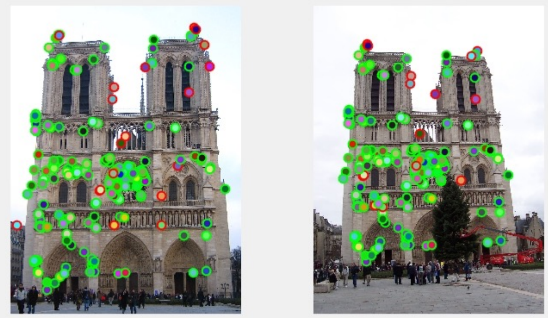
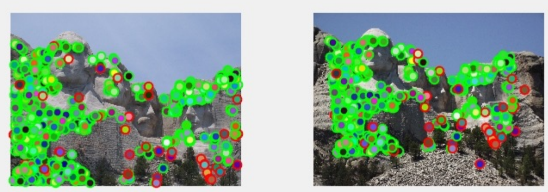

	

Image 1 : Notre Dame	**Evaluation Results : 201 total good matches, 19 total bad matches Accuracy : 91.36 %**
	
Image 2 : Mount Rushmore	**Evaluation Results : 435 total good matches, 40 total bad matches Accuracy : 91.58 %**

CONCLUSION:

Thus I was able to implement the features matching in a relatively simpler way and also incorporate scaling to an extent in my algorithm. It improved the results accordingly.