Question 1

The Gaussian pyramid is a technique in image processing that breaks down an image into successively smaller groups of pixels, in repeated steps, for the purpose of blurring it. The `pyramid` is constructed by repeatedly calculating a weighted average of the neighbouring pixels of a source image and scaling the image down.

2D Gaussian

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The results are shown in Figure 1.

Figure 1: Gaussian pyramid composite results

Listing 1: Generate Gaussian or Laplacian pyramid

```
function [pyramid, ImageDiff] = genPyramid(img, type, level)
%   generate Gaussian or Laplacian pyramid
%   img is the input image, convert to greyscale, will be forced to double.
%       type can be 'gauss' or 'laplace'.
%       level is downsampling factor + 1. level = sigma
pyramid = cell(1,level+1);
ImageDiff = cell(1, level-2);
pyramid{1} = im2double(img);
for p = 2:(level + 1)
```

```
        pyramid{p} = pyramid_process(pyramid{p-1}, 0.5, level);
end
if strcmp(type,'gauss'), return; end
for p = 1:level
    pyramid{p} = pyramid{p}-imresize(pyramid{p+1}, 2);
    if p ==level
        pyramid(p+1)=[];
    end
end
for s = 1:level-2
    ImageDiff{s}(:,:,1) = imresize(pyramid{s},0.5);
    ImageDiff{s}(:,:,2) = pyramid{s+1};
    ImageDiff{s}(:,:,3) = imresize(pyramid{s+2},2);
end
end
```

Listing 2: Image pyramid process

```
function imgout = pyramid_process(img, scale, sigma)
%   Image pyramid process
%   If img is M-by-N, then the size of imgout is ceil(M/2)-by-ceil(N/2).
%   scale = 2(expand) or 0.5(reduce)
kernelWidth = 5;
kernel = fspecial('gaussian',[kernelWidth, kernelWidth],2^sigma-2^(sigma-1));
img = im2double(img);
imgout = conv2(img, kernel,'same');
imgout = imresize(imgout, scale);
end
```

Listing 3: Draw composite figure

```
clc
clear
level = 7;
image = imread('manor.png');
image = rgb2gray(image);
I = genPyramid(image, 'gauss', level); % or 'laplace'
m = size(I{1}, 1);
newI = I{1};
for i = 2 : numel(I)
    [q,p,~] = size(I{i});
    I{i} = cat(1,repmat(zeros(1 , p),[m - q , 1]),I{i});
    newI = cat(2,newI,I{i});
end
figure(1)
imshow(newI,[]);
```

Question 2

The Laplacian Pyramid: Compute the difference between upsampled Gaussian pyramid level and Gaussian pyramid level. Use UpSample to make Gaussian Pyramid images at level L+1 the same size as L. Act as band pass filter that each level represents spatial frequencies (largely) unrepresented at other level.

The results are shown in Figure 2.

Figure 2: Laplacian pyramid composite results

Listing 4: Generate Gaussian or Laplacian pyramid

```
function [pyramid, ImageDiff] = genPyramid(img, type, level)
%   generate Gaussian or Laplacian pyramid
%   img is the input image, convert to greyscale, will be forced to double.
%       type can be 'gauss' or 'laplace'.
%       level is downsampling factor + 1. level = sigma
pyramid = cell(1,level+1);
ImageDiff = cell(1, level-2);
pyramid{1} = im2double(img);
for p = 2:(level + 1)
        pyramid{p} = pyramid_process(pyramid{p-1}, 0.5, level);
end
if strcmp(type,'gauss'), return; end
for p = 1:level
```

```
    pyramid{p} = pyramid{p}-imresize(pyramid{p+1}, 2);
    if p ==level
        pyramid(p+1)=[];
    end
end
for s = 1:level-2
    ImageDiff{s}(:,:,1) = imresize(pyramid{s},0.5);
    ImageDiff{s}(:,:,2) = pyramid{s+1};
    ImageDiff{s}(:,:,3) = imresize(pyramid{s+2},2);
end
end
```

Question 3

Search over all pyramid levels to find pixel locations (x, y) where the intensity $I(x, y)$ has an extremal value (is a local minimum or a maximum) both in space (at that level in the pyramid) and also in scale with respect to the levels above and below. When finding maxima or minima at level L, use DownSample or UpSample as necessary to make DOG images at level L-1 and L+1 the same size as L.

This experiment chooses to search a 3x3 spatial neighbourhood and values at extrema to be distinct from their neighbours in scale space by a threshold as 0.019 (the pixel value is 0.019×255).

The results are shown in Figure 3 and 4. Each keypoint is represented by a 3-tuple (x, y, σ) and the full set of detected keypoints are stored in extrema (an Nx3 matrix, where N is the number of keypoints).

Listing 5: Find local minimum and maximum

```
function extrema = getExtrema(A, thresh)
%   A is a cell
%   A{level}(:,:,1) DoG of level-1, A{level}(:,:,2) DoG of level, A{level}(:,:,3)
    DoG of level+1
extrema = [];
octave = size(A, 2);
for sigma = 1:octave
    [height, width, ~]=size(A{sigma});
    for row = 2:height-1
        for col = 2:width-1
            center = A{sigma}(row, col,2);
%             if center < thresh
%                 continue;
%             end
            A{sigma}(row, col, 2) = A{sigma}(row, col-1, 2);
            compare = A{sigma}(row-1:row+1, col-1:col+1,:);
```

Figure 3: Detected Keypoints

Figure 4: Detected Keypoints

```
            maxValue = max(compare(:));
            minValue = min(compare(:));
            if center < minValue - thresh
                extrema = [extrema;[row, col, 2^sigma]];
            end
            if center > maxValue + thresh
                extrema = [extrema;[row, col, 2^sigma]];
            end
            A{sigma}(row, col, 2) = center;
        end
    end
end
```

Listing 6: Draw detected keypoints

```
%% Question 3
thresh = 0.019;
extrema = getExtrema(ImageDiff, thresh);
figure(3)
imshow(image);
hold on
% use a circle of a different color:
% blue (level 1), green (level 2), yellow (level 3), magenta (level 4), red (level
    5).
style = {'bo', 'go', 'yo', 'mo'};
for i = 1:level-2
    point = extrema((extrema(:,3) == 2^i),:);
    plot(point(:,2) * 2^i, point(:,1)* 2^i, style{i});
    hold on
end
```

Question 4

Using a 17×17 window centred at each keypoint. For each pixel in this neighbourhood calculate: 1) the gradient magnitude (using central differences), 2) the gradient orientation, 3) a 2D Gaussian weighted version of the gradient magnitude. We choose the σ equals 4.

First we calculate orientation of gradient in each scale. And then compute the orientation and magnitude of pixels at key point vicinity (17×17). In order to easy to calculate, we fix overflow key points near corners and edges of image.

The results are shown as follows.

Listing 7: Calculate the orientation and weighted magnitude

```
%% Question 4
```

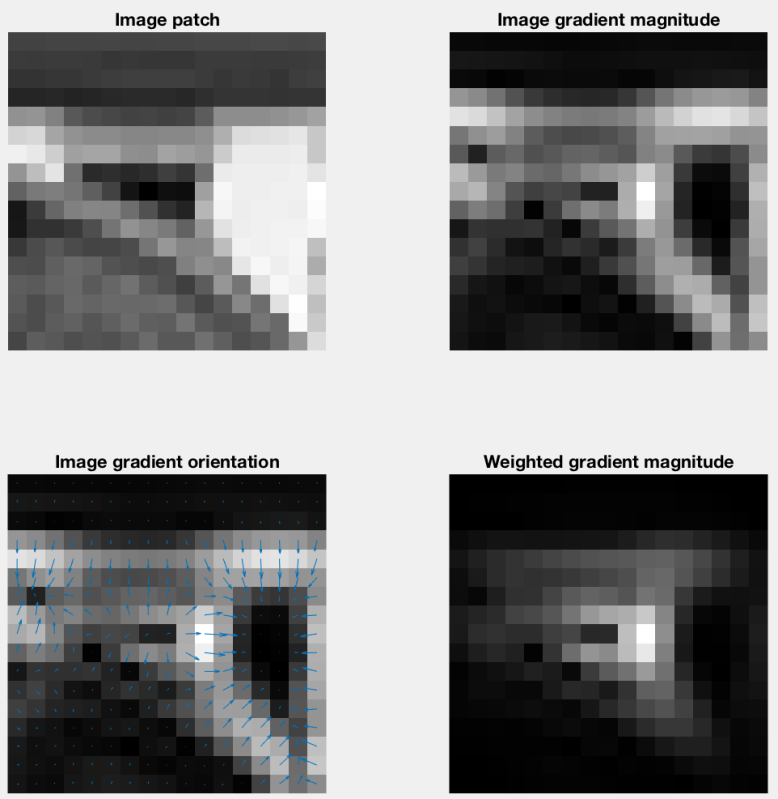
```
% Calculating orientation of gradient in each scale
GO = cell(1,level); % Gradient Orientation
GM = cell(1,level);
GX = cell(1,level);
GY = cell(1,level);
for i = 1:level
    [row,col] = size(ImageGauss{i});
    [tempM,tempO] = imgradient(ImageGauss{i});
    [tempX,tempY] = imgradientxy(ImageGauss{i});
    GO{i} = tempO;
    GM{i} = tempM;
    GX{i} = tempX;
    GY{i} = tempY;
end

% Calculating orientation and magnitude of pixels at key point vicinity
% Fixing overflow key points near corners and edges
% of image.
radius=8;
n = 1;
number = size(extrema, 1);
Magnitude = cell(1, number);
Orientation = cell(1, number);
WeightedMagnitude = cell(1, number);
X = cell(1, number);
Y = cell(1, number);
imagePatch = cell(1, number);
for i = 1:level-2
    imageSize = imresize(image, 2^(-i));
    GradientOrientations = GO{i+1};
    GradientMagnitudes = GM{i+1};
    GradientX = GX{i+1};
    GradientY = GY{i+1};
    weights = getGaussKernel(4, radius);
    [row,col] = size(GradientMagnitudes);
    point = extrema((extrema(:,3) == 2^i),:);
    for j = 1: size(point,1)
        x = point(j, 1);
        y = point(j, 2);
        a=0;b=0;c=0;d=0;
        if x-1-radius < 0
            a = -(x-1-radius);
        end
        if y-1-radius < 0
```

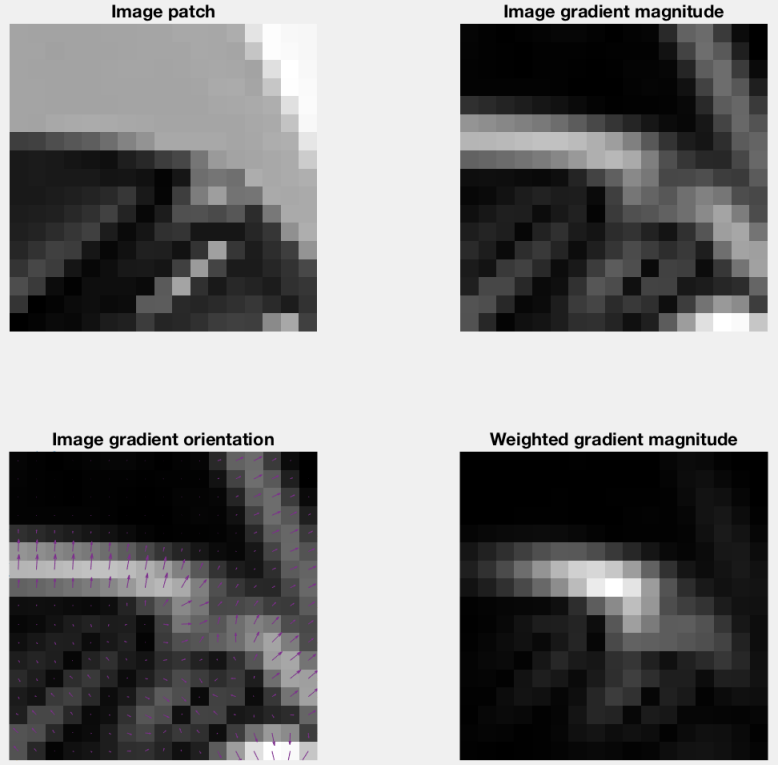
```
            b = -(y-1-radius);
        end
        if row-x-radius < 0
            c = -(row-x-radius);
        end
        if col-y-radius < 0
            d = -(col-y-radius);
        end
        patch = imageSize(x-radius+a:x+radius-c,y-radius+b:y+radius-d);
        tempMagnitude = GradientMagnitudes(x-radius+a:x+radius-c,y-radius+b:y+
            radius-d);
        tempWeightedMagnitude = GradientMagnitudes(x-radius+a:x+radius-c,y-radius+
            b:y+radius-d).*weights(1+a:end-c,1+b:end-d);
        tempOrientation = GradientOrientations(x-radius+a:x+radius-c,y-radius+b:y+
            radius-d);
        tempGradientX = GradientX(x-radius+a:x+radius-c,y-radius+b:y+radius-d);
        tempGradientY = GradientY(x-radius+a:x+radius-c,y-radius+b:y+radius-d);
        Magnitude{n} = tempMagnitude;
        Orientation{n} = tempOrientation;
        WeightedMagnitude{n} = tempWeightedMagnitude;
        X{n} = tempGradientX;
        Y{n} = tempGradientY;
        imagePatch{n}= patch;
        n = n+1;
    end
end
figure(4)
subplot(2,2,1)
imshow(imagePatch{20}, [])
title('Image patch')
subplot(2,2,2)
imshow(Magnitude{20}, [])
title('Image gradient magnitude')
subplot(2,2,3)
imshow(Magnitude{20}, []); hold on
[x, y] = meshgrid(1:17, 1:17);
quiver(x, y,  X{20}, Y{20});
title('Image gradient orientation')
subplot(2,2,4)
imshow(WeightedMagnitude{20}, [])
title('Weighted gradient magnitude')
```

Listing 8: Calculating Gaussian value given Sigma

```
function result = getGaussKernel(sigma, Radius)
```

```
% Returns a gaussian kernet
% Calculating Gaussian value given Sigma
% By default a radius will be chosen to so kernel covers 99.7 % of data.
    if nargin < 2
        Radius = ceil(3*sigma);
    end
    side = 2*Radius+1;
    result = zeros(side);
    for i = 1:side
        for j = 1:side
            x = i-(Radius+1);
            y = j-(Radius+1);
            result(i,j)=(x^2+y^2)^0.5;
        end
    end
    result = exp(-(result .^ 2) / (2 * sigma * sigma));
    result = result / sum(result(:));
end
```





Question 5

Create an orientation histogram for each SIFT keypoint, using the weighted gradient magnitudes. We add up the weighted gradient magnitudes of all gradients in the 16x16 neighbourhood whose orientations fall within that particular bin. Create a plot of the 1D orientation histogram (on the left side). Select the strongest peak and use it as the first entry in the histogram vector (on the right side). The name of siftVector is SIFT feature vector, represented by a 39-tuple $(x, y, \sigma, w_0, w_1, ..., w_{35})$.

The results are shown in Figure 5 and 6.

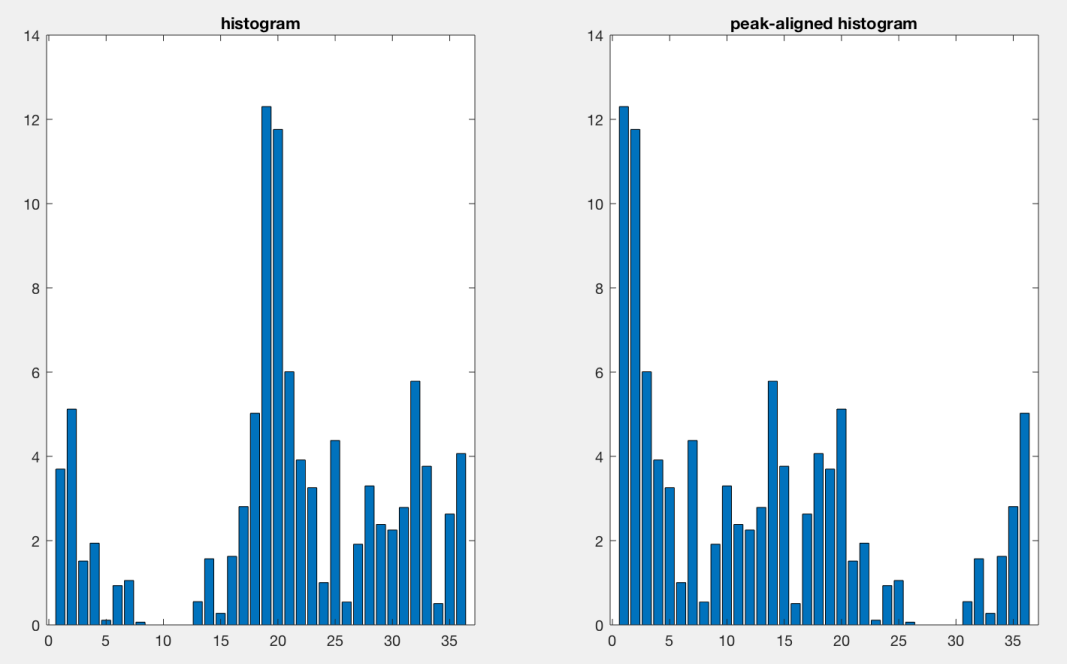


Figure 5: Orientation histogram results

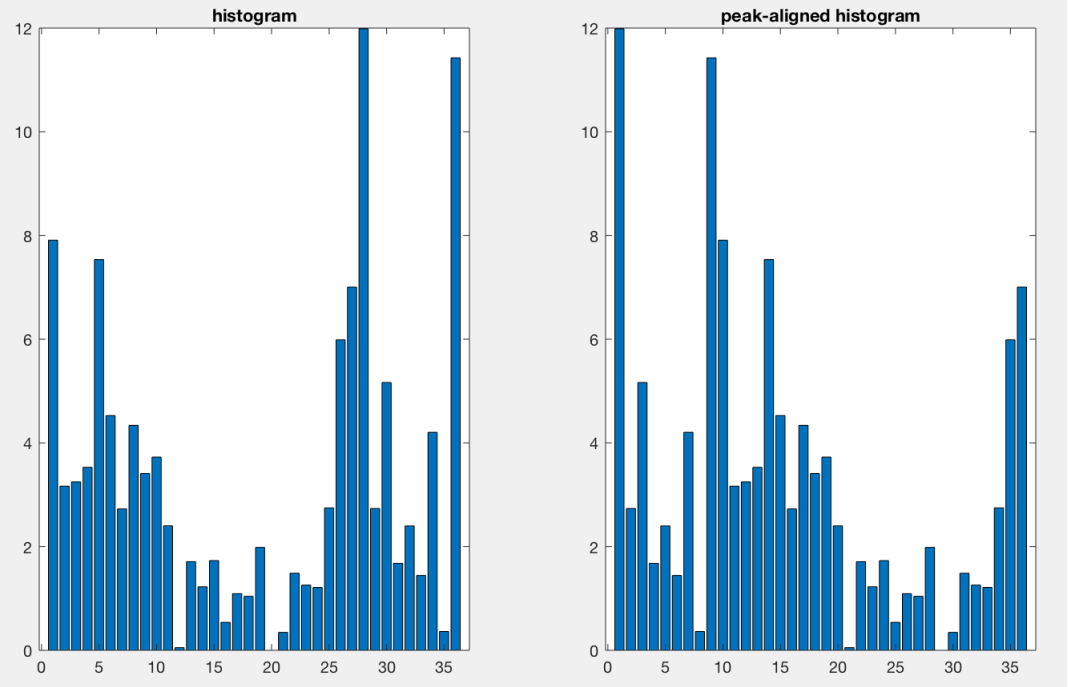


Figure 6: Orientation histogram results

Listing 9: Generate Orientation histogram

```
%% Question 5
GHistogram = cell(1, number);
peak = zeros(number,36);
for n = 1:number
    tempWeightedMagnitude = WeightedMagnitude{n};
    tempOrientation = Orientation{n};
    [wRows, wCols] = size(tempWeightedMagnitude);
    gHist = zeros(1,36);
    for i = 1:wRows
        for j = 1:wCols
            % Converting orientation calculation window
            temp = tempOrientation(i,j);
            if temp < 0
                temp = 360 + temp;
            end
            bin = floor(temp/10)+1;
            gHist(bin) = gHist(bin) + tempWeightedMagnitude(i,j);
        end
    end
    GHistogram{n} = gHist;
    Maxima = findpeaks(gHist);
    strongest = max(Maxima);
    location = find(gHist==strongest);
    peak(n,:) = [gHist(location:end),gHist(1:location-1)];
end

siftVector = zeros(number, 39);
siftVector(:,1:3) = extrema;
siftVector(:,4:39) = peak;

figure(5)
subplot(1,2,1)
x=1:1:36;
bar(x, GHistogram{20}*255)
title('histogram')
subplot(1,2,2)
bar(x, peak(20,:)*255)
title('peak-aligned histogram')
```

Question 6

Take five arguments as inputs (image,x0,y0,theta,s), where the second two arguments represent the location about which the transformation is to occur, and s represents the scale factor. The results are shown in Figure 7. The location (x0,y0) was marked on the left side image. theta=10, s=1.

Figure 7: Compare the original and the transformation images

Listing 10: Rotation and Scale

```
function imageOut = rotationScale(image, x0, y0, theta, s)
[Height,Width] = size(image);
centerX = floor(Width/2+1);
centerY = floor(Height/2+1);
dy = centerY-y0;
dx = centerX-x0;
% How much would the rotate around point shift if the
% image was rotated about the image center.
[angle, rho] = cart2pol(-dx,dy);
[newX, newY] = pol2cart(angle+angle*(pi/180), rho);
shiftX = round(-newX - dx);
shiftY = round(newY - dy);
% Pad the image to preserve the whole image during the rotation.
padX = abs(shiftX);
padY = abs(shiftY);
padded = padarray(image, [padY padX]);
```

```
% Rotate the image around the center.
rotation = imrotate(padded, theta, 'crop');
% Crop the image.
imageRot = rotation(padY+1-shiftY:end-padY-shiftY, padX+1-shiftX:end-padX-shiftX);
imageOut = imresize(imageRot, s);
% figure(1)
% imshow(imageOut)
end
```

Listing 11: Draw two images

```
clc
clear
image = imread('manor.png');
image = rgb2gray(image);

%% Question 6
x0 = 100;y0 = 100; theta = 10; s = 0.5;
image2 = rotationScale(image, x0, y0, theta, s);
figure(6)
subplot(1,2,1)
imshow(image); hold on
title('Original Image')
plot(x0, y0, 'r*')
subplot(1,2,2)
imshow(image2); hold on
title('theta = 10, s = 1')
```

Question 7

The centre point (x0ïijN̆y0) equals (100,100). For each SIFT feature vector in the original image greedily select the SIFT feature vector in the transformed image, which is its best match in the sense of maximizing the Bhattacharyya coefficient between their associated feature vectors. In order to get the strongest matching, we choose ratio as 0.8 acting as non-maximum suppression and use num (the number of matching points) to select the top num matching points, you can see that the points in the picture match well. The results are shown in Figure 8 and 9.

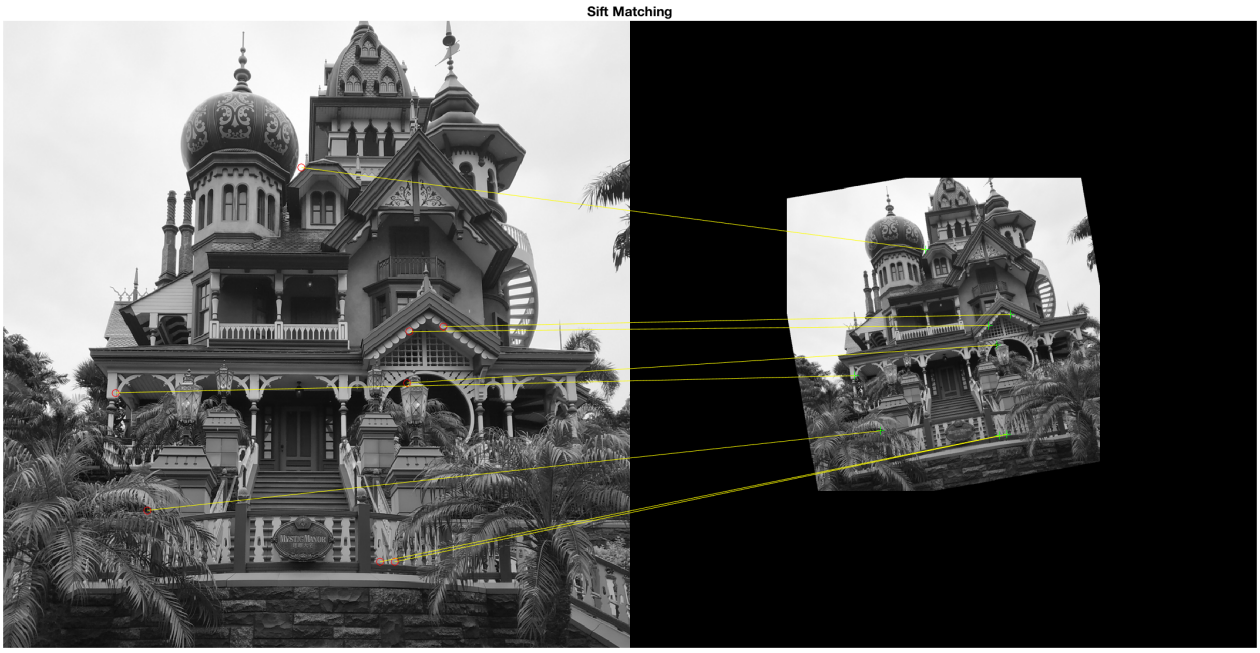


Figure 8: SIFT feature matching results, theta=10, s=0.5

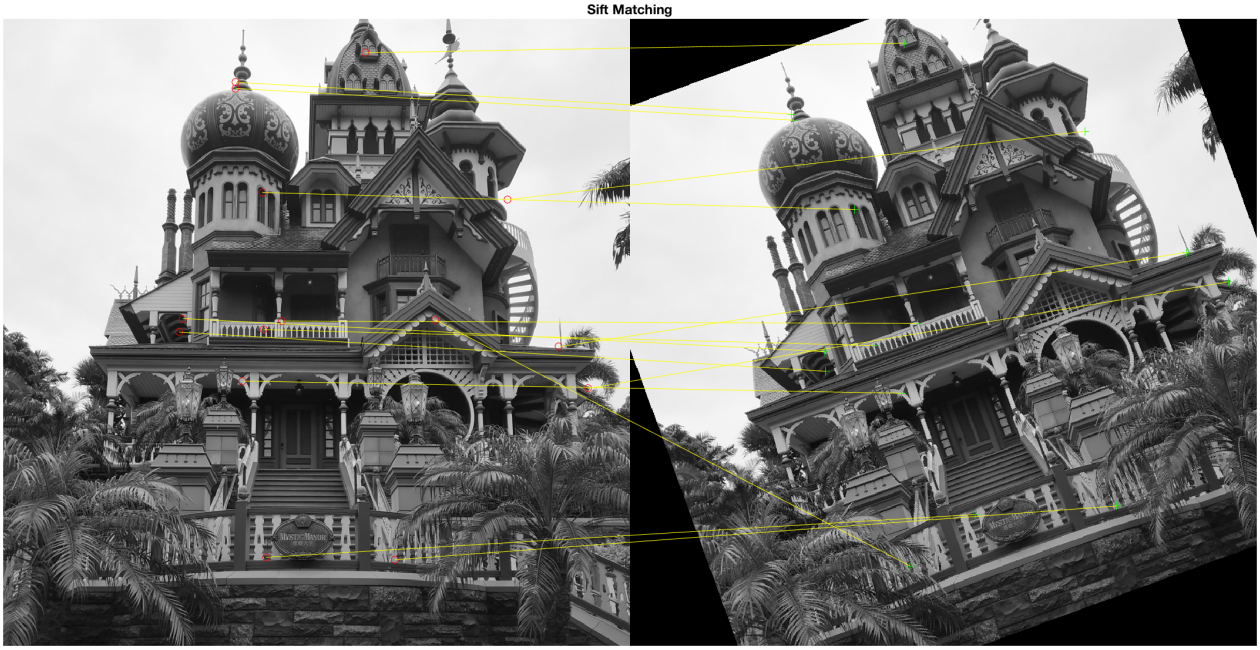


Figure 9: SIFT feature matching results, theta=20, s=1

Listing 12: Calculate Bhattacharyya Co-efficient

```
function bdist = bhattacharyya(histogram1, histogram2)
    % get number of bins
    bins = size(histogram1, 2);
    % calculate bhattacharyya co-efficient
    bcoeff = 0;
    for i=1:bins
        bcoeff = bcoeff + sqrt(histogram1(i) * histogram2(i));
    end
    % get the distance between the two distributions
    bdist = sqrt(1 - bcoeff);
end
```

Listing 13: Get SIFT feature Vector

```
function peak_align = sift(image)
%% Question 1,2
level = 6;
[ImageGauss,~] = genPyramid(image, 'gauss', level);
[ImageDoG, ImageDiff] = genPyramid(image, 'laplace', level); % or 'laplace'

% Draw composite figure
I = ImageDoG;
m = size(I{1}, 1);
newI = I{1};
for i = 2 : numel(I)
    [q,p,~] = size(I{i});
    I{i} = cat(1,repmat(zeros(1 , p),[m - q , 1]),I{i});
    newI = cat(2,newI,I{i});
end
figure(1)
imshow(newI,[]);
figure(2)
imshow(newI);

%% Question 3
thresh = 0.019;
extrema = getExtrema(ImageDiff, thresh);
figure(3)
imshow(image);
hold on
% use a circle of a different color:
% blue (level 1), green (level 2), yellow (level 3), magenta (level 4), red (level
    5).
style = {'bo', 'go', 'yo', 'mo'};
```

```
for i = 1:level-2
    point = extrema((extrema(:,3) == 2^i),:);
    plot(point(:,2) * 2^i, point(:,1)* 2^i, style{i});
    hold on
end

%% Question 4
% Calculating orientation of gradient in each scale
GO = cell(1,level); % Gradient Orientation
GM = cell(1,level);
GX = cell(1,level);
GY = cell(1,level);
for i = 1:level
    [row,col] = size(ImageGauss{i});
    [tempM,tempO] = imgradient(ImageGauss{i});
    [tempX,tempY] = imgradientxy(ImageGauss{i});
    GO{i} = tempO;
    GM{i} = tempM;
    GX{i} = tempX;
    GY{i} = tempY;
end

% Calculating orientation and magnitude of pixels at key point vicinity
% Fixing overflow key points near corners and edges
% of image.
radius=8;
n = 1;
number = size(extrema, 1);
Magnitude = cell(1, number);
Orientation = cell(1, number);
WeightedMagnitude = cell(1, number);
X = cell(1, number);
Y = cell(1, number);
imagePatch = cell(1, number);
for i = 1:level-2
    imageSize = imresize(image, 2^(-i));
    GradientOrientations = GO{i+1};
    GradientMagnitudes = GM{i+1};
    GradientX = GX{i+1};
    GradientY = GY{i+1};
    weights = getGaussKernel(4, radius);
    [row,col] = size(GradientMagnitudes);
    point = extrema((extrema(:,3) == 2^i),:);
    for j = 1: size(point,1)
```

```
        x = point(j, 1);
        y = point(j, 2);
        a=0;b=0;c=0;d=0;
        if x-1-radius < 0
            a = -(x-1-radius);
        end
        if y-1-radius < 0
            b = -(y-1-radius);
        end
        if row-x-radius < 0
            c = -(row-x-radius);
        end
        if col-y-radius < 0
            d = -(col-y-radius);
        end
        patch = imageSize(x-radius+a:x+radius-c,y-radius+b:y+radius-d);
        tempMagnitude = GradientMagnitudes(x-radius+a:x+radius-c,y-radius+b:y+
            radius-d);
        tempWeightedMagnitude = GradientMagnitudes(x-radius+a:x+radius-c,y-radius+
            b:y+radius-d).*weights(1+a:end-c,1+b:end-d);
        tempOrientation = GradientOrientations(x-radius+a:x+radius-c,y-radius+b:y+
            radius-d);
        tempGradientX = GradientX(x-radius+a:x+radius-c,y-radius+b:y+radius-d);
        tempGradientY = GradientY(x-radius+a:x+radius-c,y-radius+b:y+radius-d);
        Magnitude{n} = tempMagnitude;
        Orientation{n} = tempOrientation;
        WeightedMagnitude{n} = tempWeightedMagnitude;
        X{n} = tempGradientX;
        Y{n} = tempGradientY;
        imagePatch{n}= patch;
        n = n+1;
    end
end
figure(4)
subplot(2,2,1)
imshow(imagePatch{20}, [])
title('Image patch')
subplot(2,2,2)
imshow(Magnitude{20}, [])
title('Image gradient magnitude')
subplot(2,2,3)
imshow(Magnitude{20}, []); hold on
[x, y] = meshgrid(1:17, 1:17);
quiver(x, y,  X{20}, Y{20});
```

```
title('Image gradient orientation')
subplot(2,2,4)
imshow(WeightedMagnitude{20}, [])
title('Weighted gradient magnitude')


%% Question 5
GHistogram = cell(1, number);
peak = zeros(number,36);
for n = 1:number
    tempWeightedMagnitude = WeightedMagnitude{n};
    tempOrientation = Orientation{n};
    [wRows, wCols] = size(tempWeightedMagnitude);
    gHist = zeros(1,36);
    for i = 1:wRows
        for j = 1:wCols
            % Converting orientation calculation window
            temp = tempOrientation(i,j);
            if temp < 0
                temp = 360 + temp;
            end
            bin = floor(temp/10)+1;
            gHist(bin) = gHist(bin) + tempWeightedMagnitude(i,j);
        end
    end
    GHistogram{n} = gHist;
    Maxima = findpeaks(gHist);
    strongest = max(Maxima);
    location = find(gHist==strongest);
    peak(n,:) = [gHist(location:end),gHist(1:location-1)];
end

peak_align = zeros(number, 39);
peak_align(:,1:3) = extrema;
peak_align(:,4:39) = peak;

figure(5)
subplot(1,2,1)
x=1:1:36;
bar(x, GHistogram{20}*255)
title('histogram')
subplot(1,2,2)
bar(x, peak(20,:)*255)
title('peak-aligned histogram')
```

```
159 end
```

Listing 14: Find the sift matching points

```
1  function [index1, index2] = siftMatch(descpt1, descpt2, num)
2  % Function: Match the SIFT descriptors between two images
3  %       index1 - indices of the matched features in the first set
4  %       index2 - indices of the matched features in the second set
5  %       num - number of valid matches
6  %       descpt1 - first descriptor set
7  %       descpt2 - second descriptor set
8  num1 = size(descpt1, 1);
9  num2 = size(descpt2, 1);
10 % Number of valid matches
11 nmatch = 0;
12 % Ratio for non-maximum suppression
13 r = 0.8;
14 % bhattacharyya co-efficient between one feature and the entire second feature set
15 dis = zeros(1, num2);
16 % Smallest bhattacharyya co-efficient between two features in each feature set
17 min_dis = zeros(1, num1);
18
19 % Compute bhattacharyya co-efficient
20 for i = 1: num1
21     for j = 1: num2
22         histogram1 = descpt1(i, 4:39)/sum(descpt1(i, 4:39));
23         histogram2 = descpt2(j, 4:39)/sum(descpt2(j, 4:39));
24         dis(j) = bhattacharyya(histogram1, histogram2);
25     end
26     [dis, ind_min] = sort(dis);
27     min_dis(i) = dis(1);        % save the smallest distance for comparison
28     if (dis(1) < r * dis(2))
29         nmatch = nmatch + 1;    % one more valid match
30         ind1(nmatch) = i;       % add the index of the matched feature
31         ind2(nmatch) = ind_min(1);
32     end
33 end
34 % Keep only the valid matches
35 ind1 = ind1(1: nmatch);
36 ind2 = ind2(1: nmatch);
37
38 % Valid matches fewer than required
39 if (nmatch < num)
40     fprintf('There are only %d valid matches.\n', nmatch);
41     % Return all valid matches
```

```
    index1 = ind1;
    index2 = ind2;
% Valid matches more than required
else
    % bhattacharyya co-efficient for valid matches
    val_dis = min_dis(ind1);
    % Get the index of distance for valid matches sorted ascendingly
    [~, ind_dis] = sort(val_dis);
    % Return the index of best [num] matches
    index1 = ind1(ind_dis(1: num));
    index2 = ind2(ind_dis(1: num));
end
end
```

Listing 15: Draw the sift matching keypoints

```
%% Question 7
% Match the SIFT descriptor accross two images
descpt1 = sift(image);
descpt2 = sift(image2);
num = 8;
[index1, index2] = siftMatch(descpt1, descpt2, num);
figure;
showMatchedFeatures(image, image2, descpt1(index1, [2 1]).* descpt1(index1, 3),
    descpt2(index2, [2 1]).* descpt1(index1, 3)*s, 'montage');
title('Sift Matching');
```

Question 8

1. After looking for local minimum and maximum, we can add some operations, use image filters to remove some keypoints at the edges, and make the feature points found better. 2. Choose the better threshold when finally looking for the best match.