Question 1

Discuss some of the differences in the SIFT method provided with the one which we used in assignment 2.

This algorithm can detect and describe local features in images. The SIFT enables reliable image matching with various orientations. The basic steps of the extraction algorithm are scale-space extrema detection, key-point localization, orientation assignment, and key-point descriptor. The key-points will be detected by a cascade approach to ensure the scale invariant. At each scale, we use the difference of Gaussian function as follows.

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$

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

The maxima and minima of $D(x, y, \sigma)$ will be computed across multiple layers.

Then for the key-point localization, there are two steps. First feature points with low contrast will be extracted out with a threshold on a minimum contrast. Then comparing to the surrounding points, we remove the edges produced by the noise.

In the orientation assignment step, the keypoint orientation will be calculated with the surround points and in the final, we got useful information about feature points including location, scale, orientations.

Due to the size of the image are not 1024×1024, the code in assignment 2 should be changed to adjust the size of the image.

Listing 1: Adjust the size of the image to arbitrary image size.

```
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);
% pyramid{1} = double(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}-pyramid_process(pyramid{p+1}, 2, level);
%   pyramid{p} = pyramid{p}-imresize(pyramid{p+1}, 2);
    expand_pading = zeros(size(pyramid{p}));
    expand=imresize(pyramid{p+1}, 2);
```

```
    expand_pading = expand(1:size(expand_pading,1),1:size(expand_pading,2));
    pyramid{p} = pyramid{p}-expand_pading;
    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);
    mask3_pading = zeros(size(pyramid{s+1}));
    mask3 = imresize(pyramid{s+2}, 2);
    mask3_pading = mask3(1:size(mask3_pading,1),1:size(mask3_pading,2));
    mask1_pading = zeros(size(pyramid{s+1}));
    mask1 = imresize(pyramid{s},0.5);
    mask1_pading(1:size(mask1,1),1:size(mask1,2)) = mask1;

    ImageDiff{s}(:,:,1) = mask1_pading;
    ImageDiff{s}(:,:,2) = pyramid{s+1};
    ImageDiff{s}(:,:,3) = mask3_pading;
end
end
```

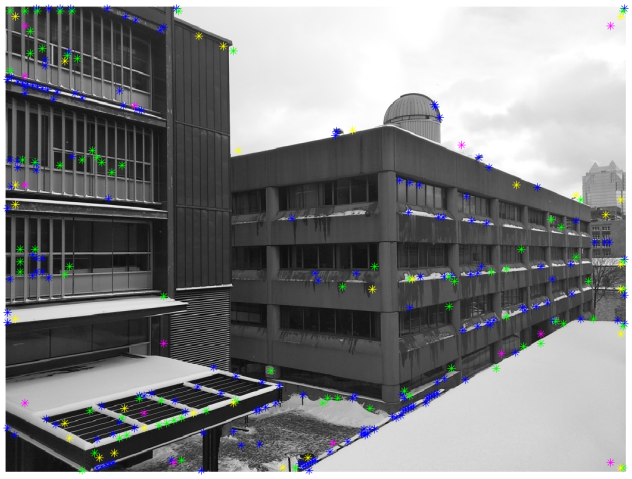


Figure 1: Sift Detected Keypoints by assignment 2

Comparing with the two methods for SIFT keypoints detection, we can find both methods in assignment 2 and the provided code can detect the same locations keypoints on edges and corners of the building. It indicates that feature points are easier to detect where the image gradient changes the most as we considered. For the differences, the approach in assignment 2 is not good at the keypoints detection in the image corner, and you can see some key

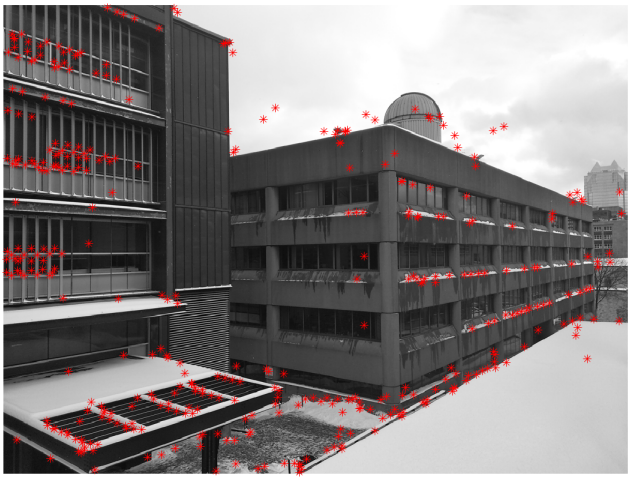


Figure 2: Sift Detected Keypoints by provided code

points are in the sky which is not supposed to be. The provided sift features method is more precisely on the keypoints on the windows of the buildings. Then we compare the feature vectors of the two methods. As we can see, the given code has more vectors than the code in assignments 2, 128 and 36 respectively.

Listing 2: Constructing SIFT feature vectors.

```
%% Q1 constructing SIFT feature vectors
clc;
clear;

rgb1 = imread('assignment3/A3/assign3Datasets/horizontal/2.png');
image1 = single(rgb2gray(rgb1));
[keypoints1,features1] = sift(image1,'Levels',4,'PeakThresh',5);

rgb2 = imread('assignment3/A3/assign3Datasets/horizontal/3.png');
image2 = single(rgb2gray(rgb2));
[keypoints2,features2] = sift(image2,'Levels',4,'PeakThresh',5);

figure(1)
imshow(rgb2gray(rgb1));
hold on
plot(round(keypoints1(1,:)), round(keypoints1(2,:)), 'r*');

figure(2)
% subplot(1,2,1)
x=1:1:128;
bar(x, features1(:, 219))
title('histogram')
```

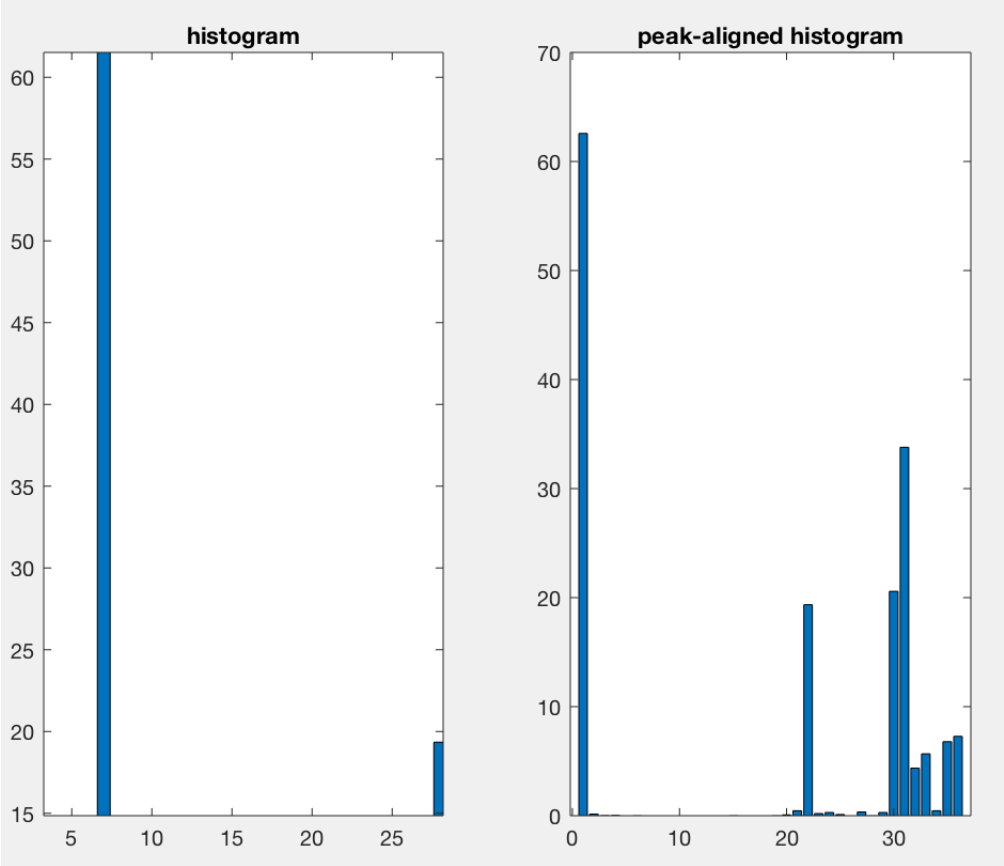


Figure 3: Histograms on assignment 2

```
23
24 descpt1 = sift_my(rgb2gray(rgb1));
25 descpt2 = sift_my(rgb2gray(rgb2));
26
27 figure(8)
28 subplot(1,2,1)
29 imshow(rgb1);hold on;
30 viscircles(keypoints1(1:2,:)',keypoints1(3,:)');
31
32 theta = 45;
33 [keypoints11,~] = sift(imrotate(image1,theta,'crop'),'Levels',4,'PeakThresh',5);
34 subplot(1,2,2)
35 imshow(imrotate(rgb1,theta,'crop'));hold on;
36 viscircles(keypoints11(1:2,:)',keypoints11(3,:)');
37
38 figure(9)
39 subplot(1,2,1)
```

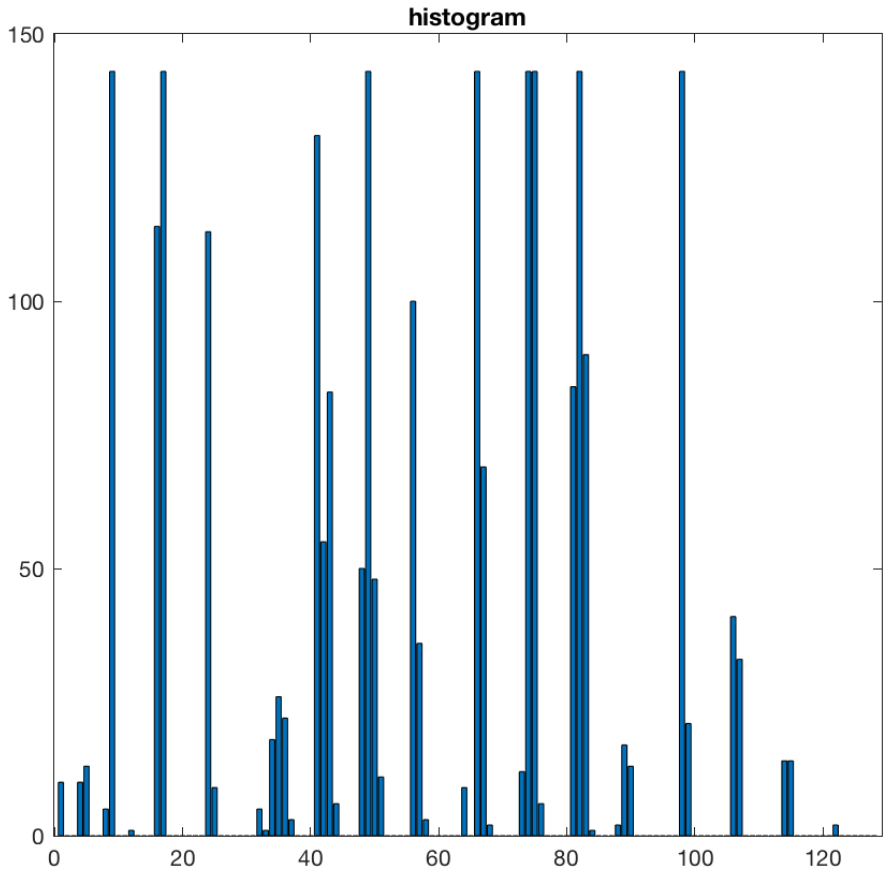


Figure 4: Histograms by provided code

```
imshow(rgb1);hold on;
viscircles(fliplr(descpt1(:,1:2)).*descpt1(:,3),descpt1(:,3));

theta = 45;
descpt11 = sift_my(imrotate(rgb2gray(rgb1),theta,'crop'));
subplot(1,2,2)
imshow(imrotate(rgb1,theta,'crop'));hold on;
viscircles(fliplr(descpt11(:,1:2)).*descpt11(:,3),descpt11(:,3));
```

For the scale invariable attributions, as we can see in Figure 6 and 5, it seems the results of the provided code better than the code in assignments in the building corners and edges.

Question 2

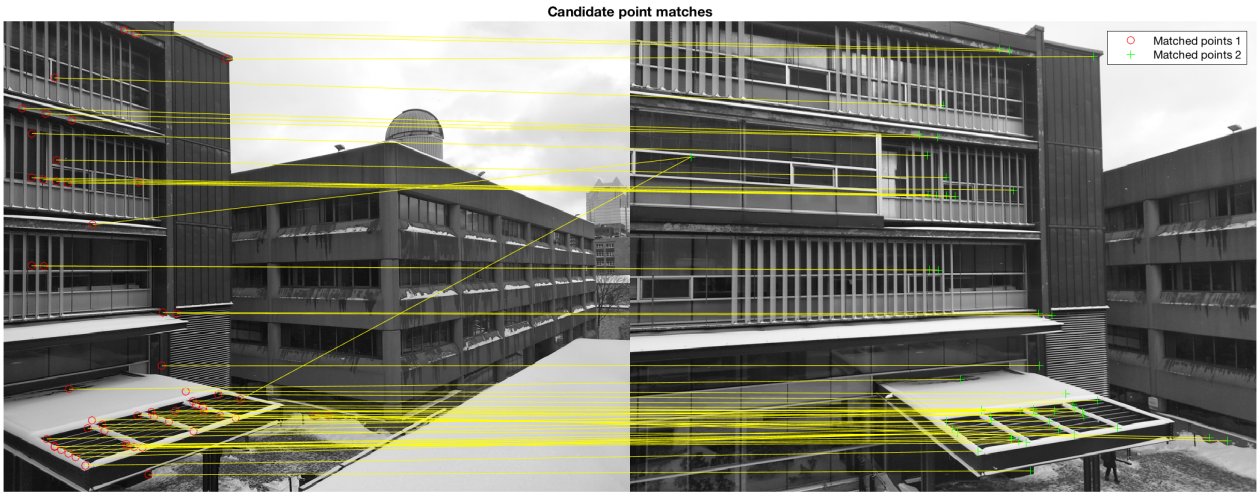
For a successive image pair in the sequence, illustrate the process of matching features. In this report, we use Matlab function *matchFeatures*. It returns indices of the matching features in

Figure 5: Rotation invertible on assignment 2

Figure 6: Rotation invertible by provided code

the two input feature sets. The implementation of the match feature method in assignment 2 is to calculate the Bhattacharyya coefficient between one feature and the entire second feature set and to find the smallest Bhattacharyya co-efficient between two features in each feature set. The implementation of the given function is to specify how nearest neighbors between features1 and features2. Two feature vectors match when the distance between them is less than the threshold set. Finding the top two closest features in the second image for a given feature in the first image, we compare the ratio of the best feature's distance to the second-best feature's distance.

Listing 3: Feature matching strategy




```
%% Q2 feature matching strategy
[indexPairs,matchmetric] = matchFeatures(features1',features2');
img1_points = keypoints1(1:2, indexPairs(:, 1))';
img2_points = keypoints2(1:2, indexPairs(:, 2))';

figure; ax = axes;
showMatchedFeatures(image1,image2,img1_points,img2_points,'montage','Parent',ax);
title(ax, 'Candidate point matches');
legend(ax, 'Matched points 1','Matched points 2');

num = 15;  s = 1;
[index1, index2] = siftMatch(descpt1, descpt2, num);
figure;
showMatchedFeatures(rgb2gray(rgb1), rgb2gray(rgb2), descpt1(index1, [2 1]).*
    descpt1(index1, 3), descpt2(index2, [2 1]).* descpt1(index1, 3)*s, 'montage');
title('Sift Matching');
```

As you can see in the results of the two methods, the Matlab *matchFeatures* function matches better than the code implemented in assignment 2. Because in the assignment 2 we only use Bhattacharyya co-efficient to measure the difference between the two pair points and use fewer features about these pair points, however in the Matlab *matchFeatures* function, they use other different methods and the features extracted by the provided sift code are more than the code in assignments 2.

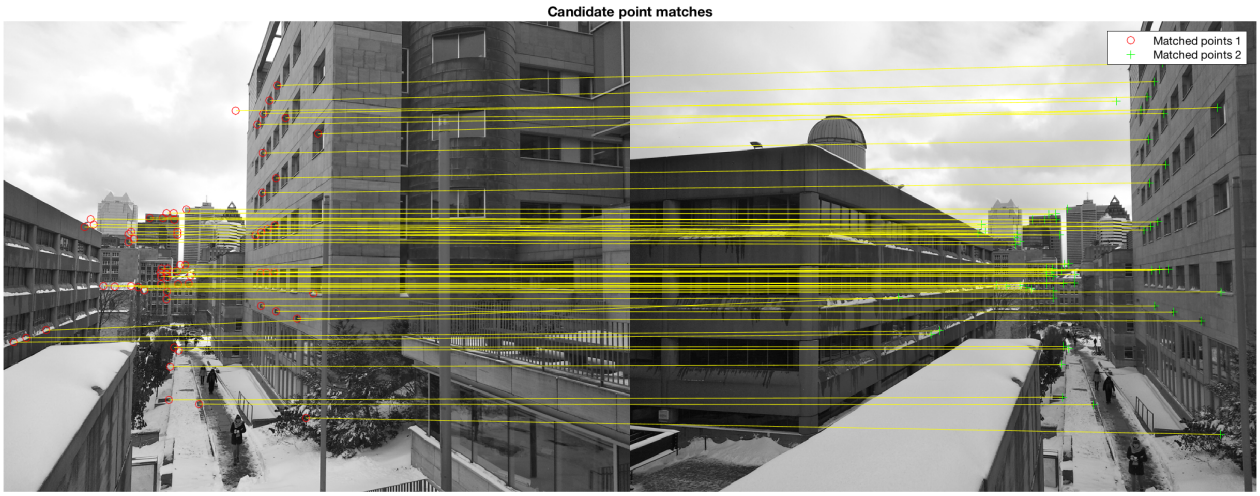
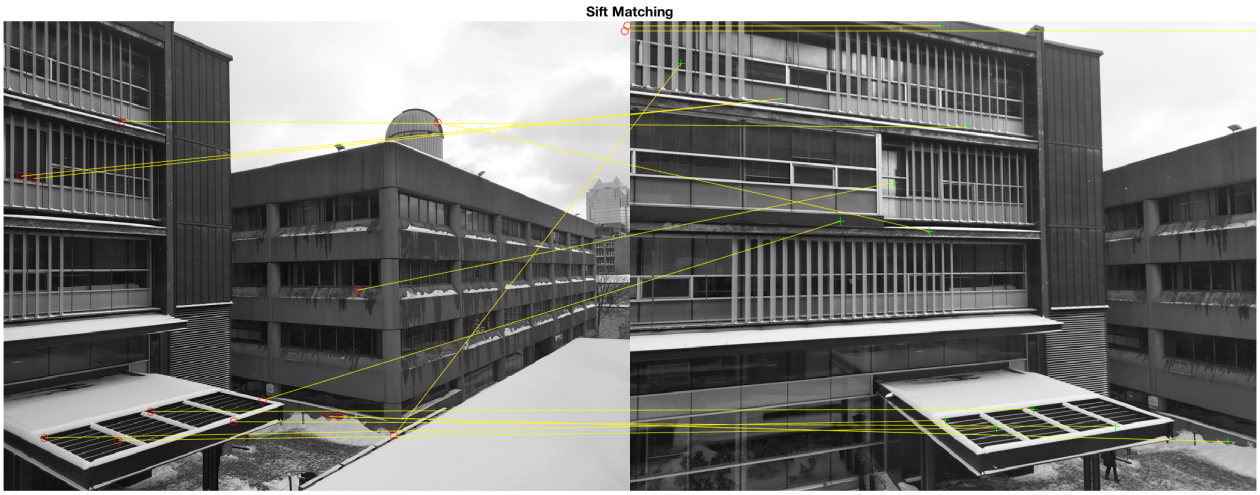


Figure 7: Show matched features by matlab function *matchFeatures*



Question 3

After matching feature points in input pictures, we will try to estimate homography using random sample consensus(RANSAC), which is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outlier. The outlier means that those feature points which cannot be matched to other image. RANSAC is a robust method to find reasonable results after certain iterations.

To see that two images taken under these conditions are related by a homography, let (X, Y, Z) be a scene point written in the scene coordinate system. Then the image positions

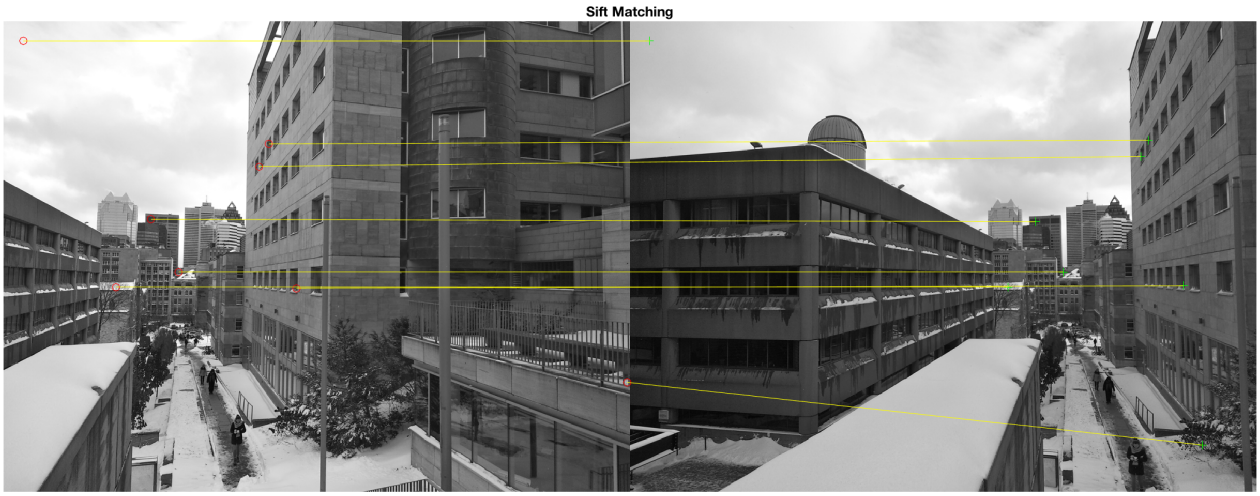


Figure 8: Show matched features by the method used in assignment 2

in the two images will be

$$\begin{bmatrix} w_1x_1 \\ w_1y_1 \\ w_1 \end{bmatrix} = KR_1[I|-c] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} w_2x_2 \\ w_2y_2 \\ w_2 \end{bmatrix} = KR_2[I|-c] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Where R1 and R2 are the two orientations of the camera.

$$(KR_1)^{-1} \begin{bmatrix} w_1x_1 \\ w_1y_1 \\ w_1 \end{bmatrix} = (KR_2)^{-1} \begin{bmatrix} w_2x_2 \\ w_2y_2 \\ w_2 \end{bmatrix}$$

$$\begin{bmatrix} w_2x_2 \\ w_2y_2 \\ w_2 \end{bmatrix} = KR_2 \left(KR_1\right)^{-1} \begin{bmatrix} w_1x_1 \\ w_1y_1 \\ w_1 \end{bmatrix}$$

Thus, we have a homography $KR_2(KR_1)^1$ that takes pixels (x_1, y_1) in camera orientation 1 to pixels (x_2, y_2) in camera orientation 2.

To solve for H for the corresponding points pairs between two images, we can rewrite the equation before.

The least squares solution for H is obtained by taking the $2N \times 9$ data matrix A and find the eigenvector of A^TA having smallest eigenvalue. We can use four corresponding points $(x_i, y_i, \widetilde{x_i}, \widetilde{y_i}), i = 1, ..., 4$, to get a system of 8 homogenous equations with 9 unknowns.

$$\begin{bmatrix} w\tilde{x}_i \\ w\tilde{y}_i \\ w \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -\tilde{x}_1 x_1 & -\tilde{x}_1 y_1 & -\tilde{x}_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -\tilde{y}_1 x_1 & -\tilde{y}_1 y_1 & -\tilde{y}_1 \\ : & : & : & : & : & : & : & : & : \\ x_N & y_N & 1 & 0 & 0 & 0 & -\tilde{x}_N x_N & -\tilde{x}_N y_N & -\tilde{x}_N \\ 0 & 0 & 0 & x_N & y_N & 1 & -\tilde{y}_N x_N & -\tilde{y}_N y_N & -\tilde{y}_N \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ : \\ 0 \\ 0 \end{bmatrix}$$

RANSAC approach There may be `outliers` on other surfaces in the scene that do not belongs to this plane, so we need to remove these points. Typically, we use RANSAN method to deal with this problem.

1. Find many feature points in the two images (x_i, y_i) and (x_i', y_i')

2. For each feature point in the image, find the similar points in the second image(x_i', y_i') by some features of the points. Then we can get a set of 4-tuples (x_i, y_i, x_i', y_i')

3. Randomly choose four 4-tuples and solve the equation about homogenous H. Find the consensus set for that homogenous that the distance of the 4-tuple of the model is under some threshold. The new 4 tuples can be computed by homography H.

$$\begin{bmatrix} w\widetilde{x} \\ w\widetilde{y} \\ w \end{bmatrix} = H_i \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The distance can be defined by the following equation.

$$dist_{H_i}(x, y, \widetilde{x}, \widetilde{y}) = ||(\widetilde{x}, \widetilde{y}) - (x', y')||_2$$

4. After repeating the step 3 a certain number of times or until you find the number of the consensus higher than some threshold. Then we use the largest consensus set to re-estimate a homography H using least squares. All of the MatLab code are as follows.

Listing 4: Solve the equation about H

```
function H = TransformH(img1_points,img2_points)
    N = length(img1_points);
```

```
    % Setting the P matrix 2Nx9
    P = [img1_points(1:N,:),ones(N,1),zeros(N,3),...
        -1*img2_points(1:N,1).*img1_points(1:N,:),...
        -1*img2_points(1:N,1);...
        zeros(N,3),-1*img1_points(1:N,:),-1*ones(N,1),...
        img2_points(1:N,2).*img1_points(1:N,:),...
        img2_points(1:N,2)];
    % SVD Decomposition
    [~,S,V] = svd(P);
    sigmas = diag(S);

    % Detecting minimum diagonal element
    if length(sigmas) >= 9
        minSigma = min(sigmas);
        [~,minCol] = find(S==minSigma);
        q = double(vpa(V(:,minCol)));
    elseif length(sigmas)<9
        q = double(vpa(V(:,9)));
    end
    % reshape the smallest eigen value
    H = reshape(q,[3,3])';
end
```

Listing 5: The RANSAC method

```
function [A_ransac,im1InlierCorrPts,im2InlierCorrPts] = transformRANSAC(
    img1_points,img2_points)
    % Intialization
    N_pts = length(img1_points);  % total number of points
    k = 5;       %number of correspondences
    e = 0.5;     %outlier ratio = number of outliers/total number points
                 %              = 1- number of inliers/total number of points
    p = 0.99; % probability that a point is an inlier
    N_iter = round(log10(1-p)/log10(1-(1-e)^k)); % number of iterations
    %distThreshold = sqrt(5.99*sigma) %sigma = expected uncertainty
    %inlierThreshold = 0

    % Determination of Inliers
    im1InlierCorrPts = [];
    im2InlierCorrPts = [];
    nmatchedInliers = 0;
    maxMatchedInliers = 0;
    for i = 1:N_iter
        % Random sample
        randIndexes = randi(N_pts,5,1);
```

```
        im1pts = img1_points(randIndexes,:);
        im2pts = img2_points(randIndexes,:);

        % Homography Matrix estimation
        H = TransformH(im1pts,im2pts);
        % Forward Transformation Error
        im1ptsFor = transformFor(img1_points,H);
        errorForward = sum((im1ptsFor-img2_points).^2,2).^0.5;
        totalForwardError = sum(errorForward);
        % Backward Transformation Error
        im2ptsBack = transformBack(img2_points,H);
        errorBackward = sum((im2ptsBack-img1_points).^2,2).^0.5;
        totalBackwardError = sum(errorBackward);
        % Total Geometric Error
        totalError = totalForwardError + totalBackwardError;

        % Expected Error Distribution Std Dev
        sigma = sqrt(totalError/(2*N_pts));
%         distThreshold = sqrt(5.99)*sigma;
%         distThreshold = sqrt(sigma);
        distThreshold = 10;
        % Determining number of inliers
        logic1Inlier = errorForward<distThreshold;
        logic2Inlier = errorBackward<distThreshold;
        im1Inliers = logic1Inlier.*img1_points;
        im2Inliers = logic2Inlier.*img2_points;
        matchedInliers = im1Inliers(:,1)>0 & im2Inliers(:,1)>0;
        nMatchedInliers = nnz(matchedInliers);
        im1MatchedInliers = im1Inliers(matchedInliers,:);
        im2MatchedInliers = im2Inliers(matchedInliers,:);
        % Updating Prarameters
        if nMatchedInliers > maxMatchedInliers
            e = (1-nMatchedInliers/N_pts);
            N_iter = round(log10(1-p)/log10(1-(1-e)^k));
            im1InlierCorrPts = im1MatchedInliers;
            im2InlierCorrPts = im2MatchedInliers;
            maxMatchedInliers = nMatchedInliers;
            A_ransac_best = H;
        end
    end
    % Final Homography Estimation
    A_ransac = TransformH(im1InlierCorrPts,im2InlierCorrPts);
%     A_ransac = A_ransac/sum(A_ransac(:));
end
```

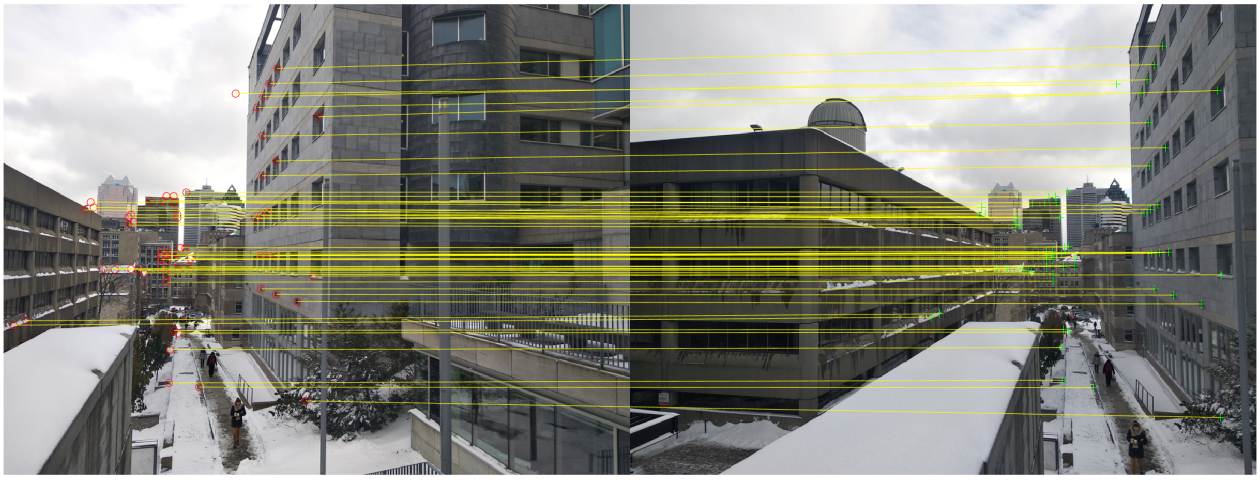


Figure 9: Result of RANSAC consensus set

Listing 6: Calculate the forward re-projection

```
function transformedPoints = transformFor(img_points,H)
    transformedPoints = [(H(1,1:2)*img_points'+H(1,3))./(H(3,1:2)*img_points'+H
        (3,3));...
        (H(2,1:2)*img_points'+H(2,3))./(H(3,1:2)*img_points'+H(3,3))]';
end
```

Listing 7: Calculate the backward re-projection error

```
function transformedPoints = transformBack(img_points,H)
    iH=inv(H);
    transformedPoints = [(iH(1,1:2)*img_points'+iH(1,3))./(iH(3,1:2)*img_points'+
        iH(3,3));...
        (iH(2,1:2)*img_points'+iH(2,3))./(iH(3,1:2)*img_points'+iH(3,3))]';
end
```

Question 4

To create a panoramic image from two overlapping photos we need to map one image plane to the other. From question 3 we can get the homography transformation matrix, in this question we need to warp the images and composite two images to make a panorama. The same as question 3, we use the stitching strategy iteratively to create a panorama for the dataset. Here we use Matlab function *imwarp*, *vision.AlphaBlender* to overlay on the top of the first one. Matlab function *vision.AlphaBlender* combines two images, overlays one image over another, or highlights selected pixels. Finally, the images have to be blended in order to get the panorama mosaic. The results of panorama and code are shown as follows.

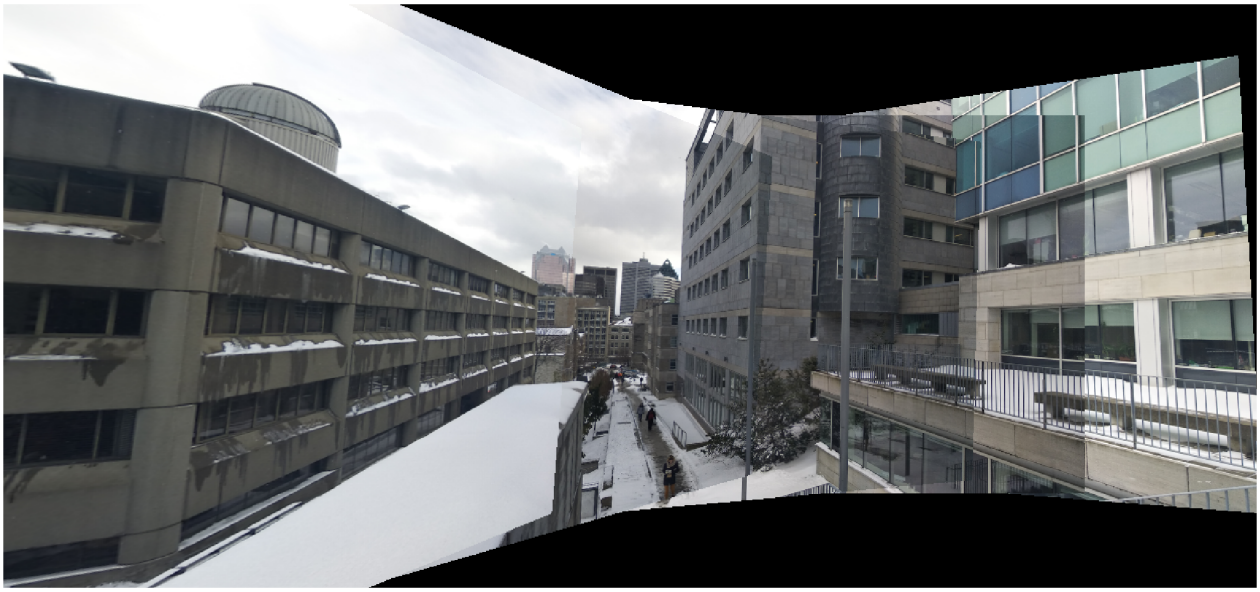


Figure 10: Result of panorama for horizontal images

Listing 8: Create a panorama

```
%% Q4 create a panorama
clc;
clear;
numImages = 4;
imageSize = zeros(numImages,2);
rgb0 = imread('assignment3/A3/assign3Datasets/horizontal/0.png');
image0 = single(rgb2gray(rgb0));
imageSize(1,:) = size(image0);
tforms(numImages+1) = projective2d(eye(3));
xlim = ones(numImages+1,2);
ylim = ones(numImages+1,2);
 for i=1:numImages
    imageName1=strcat(num2str(i-1),'.png');
    rgb1 = imread(strcat('assignment3/A3/assign3Datasets/horizontal/', imageName1)
        );
    image1 = single(rgb2gray(rgb1));
    [keypoints1,features1] = sift(image1,'Levels',4,'PeakThresh',5);

    imageName2=strcat(num2str(i),'.png');
    rgb2 = imread(strcat('assignment3/A3/assign3Datasets/horizontal/', imageName2)
        );
    image2 = single(rgb2gray(rgb2));
    [keypoints2,features2] = sift(image2,'Levels',4,'PeakThresh',5);
```

Figure 11: Result of panorama for vertical images

```
22
23      imageSize(i+1,:) = size(image2);
24
25      [indexPairs,matchmetric] = matchFeatures(features1',features2');
26      img1_points = keypoints1(1:2, indexPairs(:, 1))';
27      img2_points = keypoints2(1:2, indexPairs(:, 2))';
28
29      [H_ransac,im1InlierCorrPts,im2InlierCorrPts] = transformRANSAC(img1_points,
            img2_points);
30      tforms(i+1) = projective2d(inv(H_ransac)');
31      tforms(i+1).T = tforms(i+1).T * tforms(i).T;
32  %     [xlim(i+1,:),ylim(i+1,:)] = outputLimits(tforms(i+1),[1 imageSize(i+1,2)],[1
```

```
    imageSize(i+1,1)]);
 end

 % Compute the output limits  for each transform
for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), [1 imageSize(i,2)], [1
        imageSize(i,1)]);
end

avgXLim = mean(xlim, 2);
[~, idx] = sort(avgXLim);
centerIdx = floor((numel(tforms)+1)/2);
centerImageIdx = idx(centerIdx);
Tinv = invert(tforms(centerImageIdx));
for i = 1:numel(tforms)
    tforms(i).T = tforms(i).T * Tinv.T;
end

%%% Initialize the Panorama
for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), [1 imageSize(i,2)], [1
        imageSize(i,1)]);
end
maxImageSize = max(imageSize);

% Find the minimum and maximum output limits
xMin = min([1; xlim(:)]);
xMax = max([maxImageSize(2); xlim(:)]);
yMin = min([1; ylim(:)]);
yMax = max([maxImageSize(1); ylim(:)]);

% Width and height of panorama.
width  = round((xMax - xMin)/2);
height = round((yMax - yMin)/2);

% Initialize the empty panorama.
panorama = zeros([height width 3], 'like', rgb0);
% panorama = zeros([height width 3]);

blender = vision.AlphaBlender('Operation', 'Binary mask', ...
    'MaskSource', 'Input port');

% Create a 2-D spatial reference object defining the size of the panorama.
xLimits = [xMin xMax];
```

Figure 12: Question 5 left image

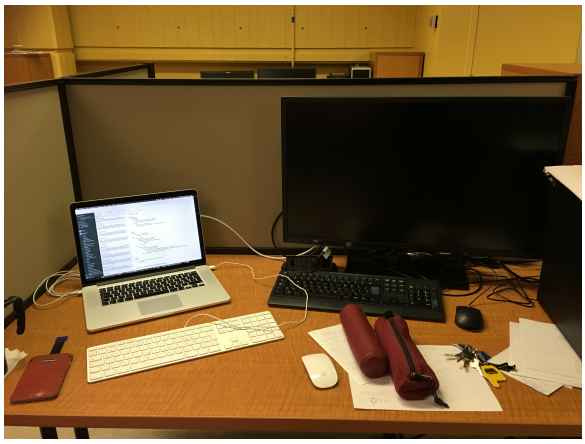


Figure 13: Question 5 centre image

```
74 yLimits = [yMin yMax];
75 panoramaView = imref2d([height width], xLimits, yLimits);
76
77 % Create the panorama.
78 for i = 1:numImages+1
79     imageNameI=strcat(num2str(i-1),'.png');
80     I = imread(strcat('assignment3/A3/assign3Datasets/horizontal/', imageNameI));
81     % Transform I into the panorama.
82     warpedImage = imwarp(I, tforms(i), 'OutputView', panoramaView);
83     % Generate a binary mask.
84     mask = imwarp(true(size(I,1),size(I,2)), tforms(i), 'OutputView', panoramaView
           );
85     % Overlay the warpedImage onto the panorama.
86     panorama = step(blender, panorama, warpedImage, mask);
87 end
88
89 figure(8)
90 imshow(panorama)
```

Question 5

The panorama reveals some distinct discontinuities. In Figure 15, blending of right and center image only leads to a much better result.

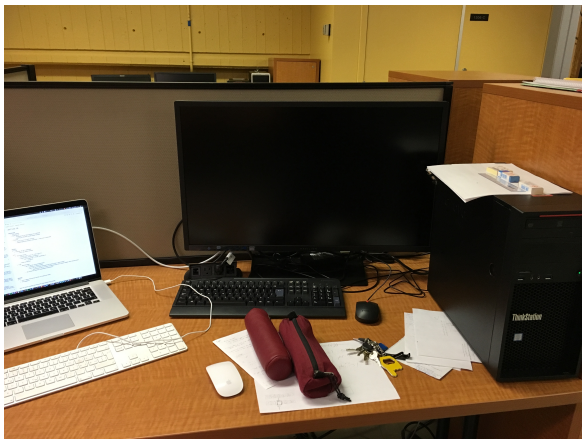


Figure 14: Question 5 right image

Figure 15: Question 5 result of panorama

Question 6

When we have calculate the homography H between closest pair of the images, we are required to decide how to produce the final panorama image. This not only involves a compositing surface and view, but also involves selecting which pixels contribute to the final composite and how to optimally blend these pixels to minimize the blur and ghost edges. From the literature search, these problems can be addressed by these techniques, namely pixel selection, blending, compositing surface, and exposure compensation.

To deal with the ghost edges, we need to decide which pixels to use and how to weight or blend them. The simplest way to create the final panorama is to take an average value

Figure 16: image 1

Figure 17: image 2

Figure 18: image 3

Figure 19: image 4

Figure 20: image 5

Figure 21: image 6

Figure 22: Question 5 result of panorama

of each pixel. However this method usually does not work very well, because of the mis-registrations and exposure differences. A better approach to the simple method is to weight pixels near the center of the image more and down-weight pixels near the edges. This can be implemented by using a $distancemap$ or $grassfiretransform$. The weighted averaging with a distance map is defined as $feathering$[1][2]. Later some researchers[3] put forward one way to improve the feathering to relieve the ghost edges is to raise the difference map values to some large power. Then the weight averages become dominated by the larger values. This approach can provide a tradeoff between visible exposure differences and blur.

Another method to deal with the ghost edges is optimal seam selection. Uyttendaele et al. [4] observe translucent looking ghosts which is the most visible artifacts in the moving objects. The idea of their method is to decided to keep one and erase one. They compare the overlapping between the input image pairs to find the regions of difference. Next they build a graph to represent the regions of difference by vertices and edges that overlap in the final composite. And the vertices should be removed from the final composite until no edge spans a pair of remaining vertices. A vertex cover algorithm can be used to compute the smallest set. So this algorithm prefers to moving regions that are near the edge of the images, which will relieve the ghost edges.

In addition, in order to deal the ghost edges at the locations, we need to blend the images to compensate for exposure difference and other mis-alignments. It is difficult to come up with a pleasing balance between smoothing out low-frequency exposure variations and retaining sharp enough transitions. Burt and Adelson [5] created an algorithm to solve this problem called Laplacian pyramid blending. They use a frequency adaptive width created by Laplacian pyramid and make the transition width a function of pyramid level. The first step is to convert each image to Laplacian pyramid. This create a special representation of

image signal. Then the valid pixel image associated with each source image is converted in Gaussian pyramid. Finally, the composite image are reconstructed by interpolating all the pyramid levels.

Last, we talk about a problem faced in the above questions, which is the larger view of the images. A natural approach is to select one of the images as the $reference$ and then to warp of the other images into the reference coordinate system. For larger fields of view, we cannot use a flat representation without excessively stretching pixels out of the image border. The usual method for larger view is to use a cylindrical or spherical projection.

References

[1] Chen, C.-Y. and Klette, R. (1999). Image stitching - comparisons and new techniques. In Com- puter Analysis of Images and Patterns (CAIPâĂŹ99), pages 615âĂŞ622, Springer-Verlag, Ljubljana.

[2] Szeliski, R. and Shum, H.-Y. (1997). Creating full view panoramic image mosaics and texture- mapped models. Computer Graphics (SIGGRAPHâĂŹ97 Proceedings), , 251âĂŞ258.

[3] Wood, D. N. et al.. (1997). Multiperspective panoramas for cel animation. In Computer Graphics Proceedings, Annual Conference Series, pages 243âĂŞ250, ACM SIGGRAPH, Proc. SIG- GRAPHâĂŹ97 (Los Angeles).

[4] Uyttendaele, M., Eden, A., and Szeliski, R. (2001). Eliminating ghosting and exposure arti- facts in image mosaics. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPRâĂŹ2001), pages 509âĂŞ516, Kauai, Hawaii.

[5] Burt, P. J. and Adelson, E. H. (1983). A multiresolution spline with applications to image mosaics. ACM Transactions on Graphics, 2(4), 217âĂŞ236.