MLCV Report II

Lizhang Lin (00840705) Tianci Ma (00804678)

1. Introduction

Feature detection and matching are important in many applications of computer vision such as motion tracking, 3D construction and image recognition. In this report, we aim to investigate on the subject of extracting and matching of the interest points and also calculating homography and fundemantal matrix between two images.

2. Preparing the tools

In order to match two images and obtaining their homography and fundamental matrix. We need to be able to extract interest points from the two images, extract the descriptor of the interest points and match the interest points through matching their descriptors. Once we have matched interest points we can calculate their homography and fundamental matrix.

A. Interest Point Detector

We implemented both manual and automatic interest point detection method. The manual interest point extraction allows users to manually select corresponding points on two images. The coordinate of these selected points were stored and used for further processing.

We implemented Harris Detector for automatic interest point extractions with some modification on the Harris detector algorithm. It should be noted that our detector implementation always selects the strongest 500 interest points since the computational cost of matching is superlinear in the number of interest points, it is desirable to restrict the maximum number of interest points extracted from each image.

One of the modifications we made is to set the threshold, which is used to pick suitable interest points, as a percentage of the maximum Harris response value (the default is 1%) so that the threshold becomes adaptive to varying images. So by selecting an appropriate threshold, for example 1%, we were able to obtain distinctive (the distinctiveness can be shown later) interest points as shown in figure 1 (left).

However, these interest points are distributed unevenly across the image. For example, it can be observed that points are denser in regions that exhibit high contrast. This property is not desired for matching correspondences because the feature descriptors, which are patches, of adjacent densely distributed interest points have strong similarity in terms of their features.

In order to mitigate this problem, Adaptive Non-Maximum Suppression (ANMS) method was proposed by Brown, Szeliski, and Winder (2005) [1], which only detects features that are both local maxima and whose response value is significantly (10%) greater than that of all its neighbors within a radius r. In our implementation, we implemented a simplified version of ANMS which filters out interest points that are not local maxima among their neighbours within a radius r, and the result can be observed in figure1 (right).

We noted that the Harris detector we implemented for experimental purpose is variant to scale changes because it only detects change of gradient in a given $7X7$ Gaussian window, and so are feature descriptors (section II.B) produced based on the Harris detector. One option we tried to counter this problem is to run detections at different scale spaces and obtain corresponding feature descriptors adjusted to the most optimal scale. However this method is very time consuming compared to SIFT (Scale-Invariant-Feature-Transform), after comparison we decided to use SIFT for interest point detection and descriptor extraction when the images have scaling transformation. Details can be found in section 3.

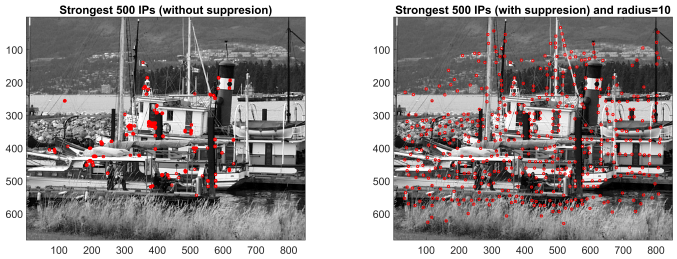

Fig. 1. Left: 500 strongest interest points obtained without ANMS; Right: 500 strongest interest points obtained after ANMS

B. Feature Descriptor

At first we took the $32X32$ squared pixel patch around the detected Harris interest points and for each of these patches, we obtained a color histogram with 127 bins (Note that there are 255 color level so 2 color levels in each bin) as the descriptor for that interest point. However we noticed that this type of descriptor is not very good for matching interest points with rotational transformation. This is due to the nature of the squared patch which, during rotation, causes some pixels that were in the squared patch to get rotated out and some new pixels outside the squared patch to get rotated in. An alternative to solve this problem is to estimate a dominant orientation at each detected interest point and use this orientation to rotate the square patch as the feature descriptor. For simplicity, in our implementation we used circle patch with a radius of 16 pixels around the interest points and obtain color histogram of the circle patches as the descriptors. This greatly improves the result as we can see, shown in figure 2, that more accurate and successful correspondences can be matched.

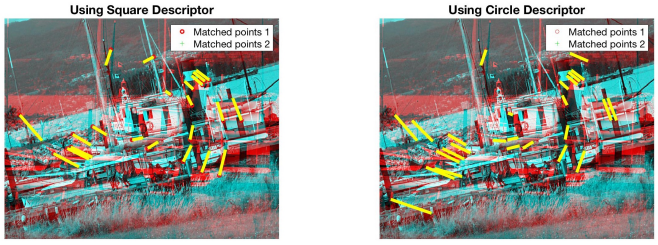

Fig. 2. Left: Matched correspondences using square patch descriptor. Right: Matched correspondences using circle patch descriptor

C. Feature Matching

Given feature descriptors extracted from two images, the goal of matching is to find consistent feature matches between two images. In order to do that, we find a set of correspondences using nearest neighbor (NN) with Manhattan distance metric. The result gives more false-positive matching than true-positive matching (figure 3 top).

We noticed that many correspondences are matched ambiguously due to the fact that their 1-NN (first nearest neighbour)

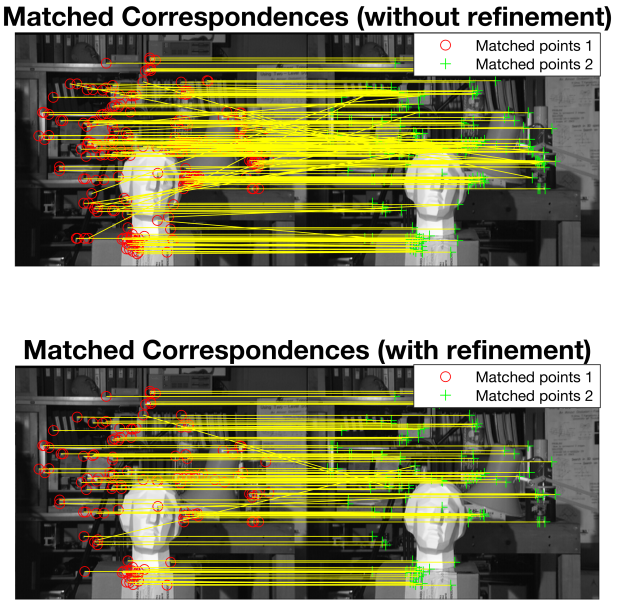

Fig. 3. Matched Correspondences before and after refinement techniques

and 2-NN (second nearest neighbour) have very close value. In order to solve the problems and increase the matching accuracy, we used 1-NN to 2-NN ratio as a threshold to reject ambiguous matching. It works by assuming that the 1-NN in some image is a potential match whilst the 2-NN in that same image is an incorrect match. Also, we followed the intuition that the correct matches always have substantially lower error than incorrect matches.

It should be noted that the technique we implemented here for rejecting ambiguously matched correspondences might inevitably reject some true-positive matches. However, the technique is still very powerful because, for example, by selecting a suitable 1-NN to 2-NN ratio it can eliminate 80% of the false matches at expense of losing less than 10% correct matches. Therefore, it significantly reduces the number of iterations required for RANSAC in the subsequent steps.

After that, there still exist interest points in the second image being matched to multiple points in the first image. This is not desirable because it can be observed that more than 50% of non-unique matches are incorrect. So we further refined the resulting matched correspondences by keeping only uniquely matched correspondences. The result can be found in figure 3 (bottom).

D. Obtaining Homography Matrix

We obtained a single homography matrix as taught in the slide 20 of the lecture Matching. Only 4 points are needed for computing a single homography matrix. However our task is to select the best 4 points that output an homography matrix, which can best describe the transformation between two images. We used the RANSAC method to achieve this task, this method requires as many available correspondences as possible and it is preferable that the set of correspondences is pre-processed at feature matching stage, such that only a small fraction of them are incorrect. RANSAC works by repeating many random iterations to find the best homography matrix for a set of available points. In order to find the best homography matrix, the RANSAC method randomly selects 4 points from the total available set in each iteration. It computed the homography matrix for that 4 points and it projected all the available points from one image to another using this homography matrix. Outlier were found by comparing the pixel distance between the expected projection and the actual projection with a pre-defined threshold measured in Manhatten distance (pixels). In the end, we only selected the homography matrix that has the least number of outliers.

Note that the pixel distance between the expected projection and the actual projection was found by first projection the point from image 2 to image 1 using the homography matrix. This is to simply multiply the matrix with the points in image 2 (appended with a 1 on the third component) and divide the obtained x and y component (the first and the second component in the result matrix) by the obtained z component (the third component in the result matrix). After projection, we used the Manhattan distance to calculate the distance between the expected coordinates and the projected coordinates. We obtained the sum of the absolute x and y difference between the two pairs of the coordinates, and we took the average of this number of all the projected points. This number that we obtained will be the Homography Accuracy Error, which will be denoted as HA.

E. Obtaining Fundamental Matrix

We obtained the Fundamental matrix using the Eight-point Algorithm as first proposed by H. Christopher Longuet-Higgins [2]. In this algorithm, a minimum of 8 points is needed for calculating a fundamental matrix. However, just as above, we wanted to find the best 8 points that describe the transformation. Therefore we also implemented a RANSAC method for obtaining the best fundamental matrix. The method is very similar as the RANSAC method for computing homography matrix. We ran large amount of iterations and in each iteration, we selected 8 points from the total available set. A fundamental matrix was calculated and we used this fundamental matrix to plot epipolar lines on one of the image. The distance between the corresponding points to the corresponding epipolar line was calculated and we found the outliers by comparing that distance with a pre-defined threshold. In the end, we only select the fundamental matrix that has the least amount of outliers.

In order to calculate the epipolar lines, we simply multiplied the matrix with the points in image 2 (appended with a 1 on the third component in the coordinate matrix) with the fundamental matrix. The obtained result was respectively a , b and c components of a line, in which $ax + by + c = 0$. From this equation we could draw the epipolar line on the image.

3. Obtaining the result

In order to apply the methods we have implemented above and to obtain results. We have prepared two sets of images, with image set HG which rotates and has scale changes and image set FD which has only horizontal translational displacement.

A. Reduction of image size

We first investigated how reducing the same image to half of its size would affect Harris interest point detection and matching. After detecting the interest points on both images, we obtained figure 4, we could observe that there are fewer interest points in the reduced size image. Some corners disappeared and some appeared. This was because our Harris implementation used a Gaussian window of fixed size for detection, which does not respond to changes in scale. HA was very close to zero, as it was the same image without any translation or rotation, and also that we only used inlier for obtaining the homography matrix.

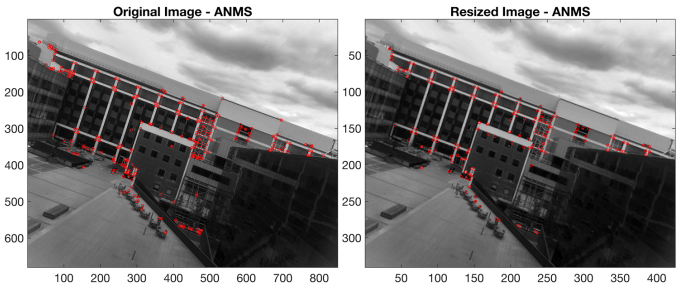

Fig. 4. Image pairs with extracted 500 strongest interest points highlighted in red circles. Note how Harris detector respond to scale change.

B. Automatic vs Manual interest point extraction

We wanted to compare the automatic vs manual interest point extractions. We first manually extracted 10 points from the image, and then we also randomly selected 10 points from the automatic interest point extraction for controling variables. After interest points were detected using both methods, we obtained the homography matrix for both methods and we projected the points from one image to another image using the both matrices. Note that the points that we used for ground truth were selected from the inlier of automatic interest point extraction. We obtained figure 5 for manual extraction (right) and automatic extraction(left). Looking at the expected and actual projection, we could observe that automatic interest point results in a homograpy that is more accurate than the manual extraction. This might be due to the error that is made during manual interest point selections. It is hard to select exactly the same pixel from two images using naked eyes while this is easier to be done automatically.

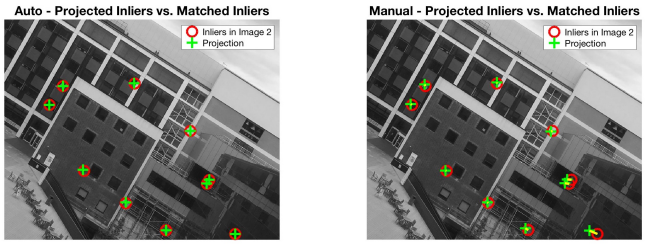

Fig. 5. Projection of Interest points using automatic interest points extraction vs manual interest points extraction. The projected points (marked in green) in the left and right images are projections of manually selected points using Homography matrix calculated with automatically and manully selected points respectively while the inliers (marked in red) are obtained from ground truth.

Looking at the homography matrix below (left) obtained from using the automatic interest points extraction method. We observe that there are inconsistency on the rotational and scaling component of the matrix. We interpolate between the two difference value. E.g. Interpolate 1 and 1.14 to obtain 1.07. Now by solving the simultaneous equation, we obtain scaling factor of 2.02 and rotation of 32.03 degree for image transformation that has its interest points extracted using the automatic method.

$$\begin{bmatrix} 1.71 & 1.14 & -713.5 \\ -1 & 1.71 & 16.36 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1.8333 & 1.3333 & -833.33 \\ -1.0833 & 2 & 1.25 \\ 0 & 0 & 1 \end{bmatrix}$$

Looking at the homography matrix above (right) for manual extraction, we interpolate between the inconsistent parameters and solve the simultaneous equations. We obtain scaling factor of 2.26 and rotation of 32.23 degree for image transformation that has its interest points extracted using the manual method.

We ran Ransac to obtain homography matrix again using all available points and set the projection error threshold to be only 1 pixel. This result was used as ground truth. The obtained scaling factor is 1.98 and rotation is 30.25 degree. The two obtained homography matrices implies similar scaling and rotational factor between automatic and manual method. However compared to ground truth result, we can see that using automatic interest point detection produces slightly better result. This can also be verified by figure 5, which shows that automatic method has better projection than the manual method.

C. Effect of the number of available corresponding points

This part of the paper focuses on investigating how the number of available corresponding points would affect the homography accuracy. In order to compare the HA of homography obtained using different numbers of corresponding points. We first selected 23 pairs of correct corresponding pairs (obtained from Ransac using 10000 iteration and an error threshold of 2) to be the reference points (ground truth). Homography matrices obtained from the experiments will be applied on these pairs of ground truth point for obtaining HA. To do the experiment, we first randomly selected 4 pair of points to compute homography matrix and obtain the HA by applying the obtained matrix to the reference points. Then we gradually increased this number to the maximum, which is 40 in our case. Each calculation will be repeated for 100 times to obtain a more reliable average HA. Note that due to relatively small number of available interest points, the reference points will not be separated from the homography matrix training data. Running the experiment and we obtained Figure 6 as shown below. It was observed that the HA quickly converge to zero as number of available points increased. This is understandable, since we were using Ransac, a correct homography matrix can be obtained as soon as four correct points were present in the available points. With a chance of a randomly selected point to be an outlier of around 15% (in this special case, 27 total correspondences and 4 outliers), and with only 4 available points, it is likely that not all 4 of them are within the inlier set of ground truth. However as number of available point increases, (say at 10 available points) it is less and less likely that there are fewer than 4 inlier, hence HA is converging to zero.

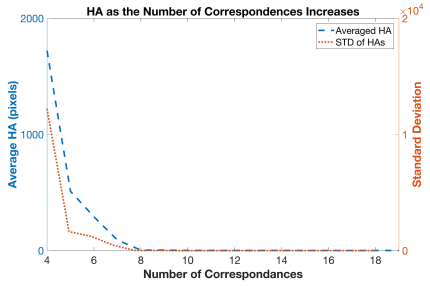

Fig. 6. Projection of Interest points using automatic interest points extraction vs manual interest points extraction

Among 27 total available points, there are 4 outliers. During each iteration of Ransac, four points were randomly selected to compute a homography matrix. The whole list of automatic correspondences were then projected using this homography matrix. If the distance between their actual projection and the expected projection was larger then a threshold (2 pixel in this case), the point was regarded as an outlier of that homography matrix. These 4 points were found to be the outlier of the best Homography matrix obtained from the Ransac result.

D. Fundamental Matrix and epipolar lines

Fundamental matrices obtained from the automatic correspondences are shown as the matrices below:

$$\begin{bmatrix} 0 & 0 & 0.077 \\ 0 & 0 & 0.695 \\ -0.08 & -0.701 & 0.117 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0.045 \\ 0 & 0 & 0.670 \\ -0.042 & -0.696 & 0.152 \end{bmatrix}$$

The obtained epipoles at the two images were respectively at (1.0e+03 *-9.0482, 0.5424) and (1.0e+03 *-3.5504 0.4069). Theoretically these epipoles should be at infinity on the X axis but they are respectively at 1.0e+03 *-9.0482 and 1.0e+03 *-3.5504 . This might be caused by that the two images have not only have horizontal translation but also some vertical translation, as shown in Figure 7 . The epipolar lines are displayed on the images and they demonstrate that feature matching and Ransac fundamental matrix estimation is able to recognise a relatively good horizontal translation between the two images. Note that the epipoles are not present on images as they both are supposed to be infinitely far from the origin of the images.

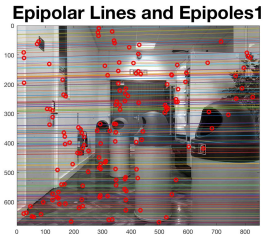

Fig. 7. Projection of interest points from image1 on image 2 and the epipolar lines

E. Disparity Map & 3D Depth Map

In order to minimise human error occurred when taking the two FD pictures, they were rectified before calculating disparity map. Figure 8 shows the result of rectified left and right FD images as well as the disparity map from one FD image to the other. We can observe from the disparity map that there are some errors, for example, the black dots scattered within objects at close range. This can be originated from various sources, such as the process of rectification (check the rectified wall and door on the left of the rectified image), the calibration of stereo camera and the algorithm used for disparity map calculation. However, despite these minor error, the distance of most objects, such as the bins, vending machines and the door, from the position of camera can be relatively correctly identified.

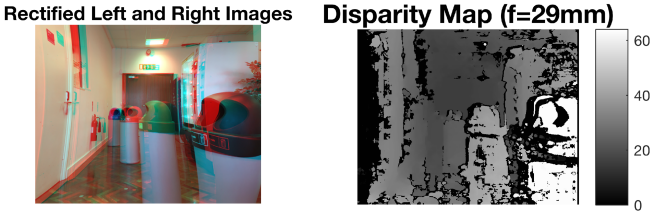

Fig. 8. Disparity Map between image 1 and 2. Note that the value of the colour bar on the right side of the map stands for the number of pixels that matched correspondences are shifted to the right from image 1 to 2.

Based on the disparity map, the 2D FD image pair can be reconstructed at 3D space as shown in figure 9 (top left). It is clear that accurate reconstruction can be obtained around areas where disparity map has little error. We then increased the focal length of the camera by 2 mm and added

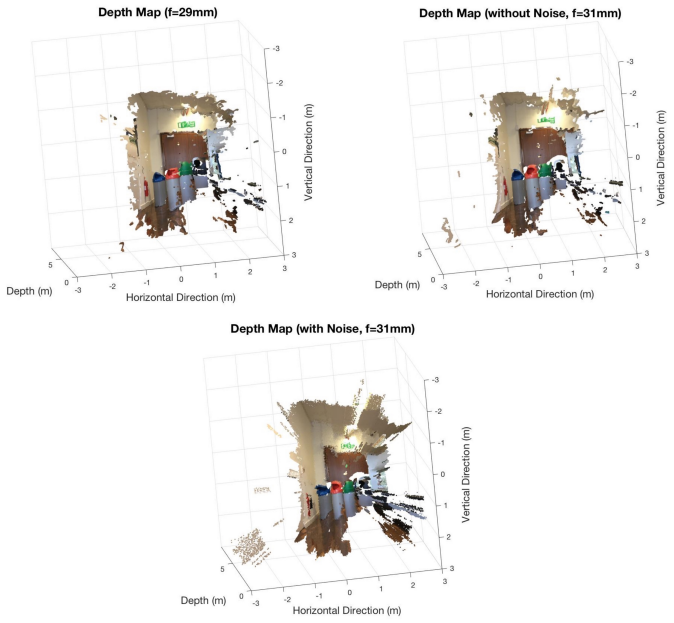

Fig. 9. 3D depth map obtained from disparity map.

Gaussian noise (maximum 2 pixels) to the disparity map obtained after increasing the focal length. The resulting 3D depth maps are shown in figure 9 (top right and bottom respectively). Comparing the original depth map and the depth map obtained after increasing the focal length, it can be clearly observed that the latter seems to be a zoomed-in and magnified version of the former because the focal length is increased slightly. However, the physical size of the objects and their distance from the camera are still preserved, which is consistent with the fact that camera positions are not changed. Moreover, we can observe that from the depth map obtained after adding Gaussian noise (figure 9 bottom), it is clear that the 3D reconstruction of objects close to the camera are resilient to additive noise (maximum 2 pixels) whereas that of distant objects becomes less reliable (for example, the emergency exit sign) at the presence of noise.

4. Conclusion

To conclude, we implemented Harris detector and a scale-invariant circle patch as rotational-invariant descriptor. We also tested a set of HG images with SIFT, and found that SIFT outperforms Harris due to the fact that SIFT is scale-invariant. Furthermore, our experiment shows that it is important to select an appropriate 1-NN to 2-NN ratio to reject ambiguously matched correspondences and prepare the refined correspondences for RANSAC in the subsequent steps. Our RANSAC is able to produce an accurate result with as little as 1000 iterations and only 10 refined correspondences. We can also conclude that more available points increases the Homography Accuracy.

References

[1] Brown, Matthew, Richard Szeliski, and Simon Winder. "Multi-image matching using multi-scale oriented patches." Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 1. IEEE, 2005.

[2] H. Christopher Longuet-Higgins (September 1981). "A computer algorithm for reconstructing a scene from two projections". Nature. 293 (5828): 133âĂŞ135. doi:10.1038/293133a0.

5. Appendix

Listings

Listing 1. Q1.1.a - Get Manual Interest Points

```
function [points1_manual,points2_manual] = getManualIPs(img1, img2, num)
    figure;
    subplot(121);
    imshow(img1);
    points1_manual = fliplr(ginput(num));
    subplot(122);
    imshow(img2);
    points2_manual = fliplr(ginput(num));
end
```

Listing 2. Q1.2.a - Harris Detector

```
function coords = InterestPointDetector(img, t_percent, ANMS,radius)
%%
%Q1 Automatic
%2a) Harris Point Detector
maxheight = size(img, 1);
maxwidth = size(img, 2);

% ================= convert colour scale to gray scale ==================
if (size(size(img),2) == 3)
    img = rgb2gray(img);
end
img = im2double(img);

%% ======== find interest points based on the type of detector used ========

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% see http://db.cs.duke.edu/courses/cps196.1/spring04/handouts/Image%20Processing.pdf
% about why using a mask is better for generalising the approach of getting
% image derivatives
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ================= obtaining image derivatives ==================
dx = [-1 0 1;-1 0 1;-1 0 1]; % Derivative masks
dy = dx';
Ix = conv2(img, dx, 'same'); % Approximation of Image derivatives
Iy = conv2(img, dy, 'same');

% ====================== gaussian weighting ======================
g = [0.03 0.105 0.222 0.286 0.222 0.105 0.03];
fx = repmat(g, size(g,2), 1);
fy = fx';
g = fx.*fy;
```

```
Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed squared image derivatives
Iy2 = conv2(Iy.^2, g, 'same');
Ixy = conv2(Ix.*Iy, g, 'same');

% ================ auto-correlation matrix ===============
% ================ | Ix2 Ixy | ============================
% ================ | Ixy Iy2 | ============================


% ===== eigenvalues of the auto-correlation matrix =======
lambda1 = 0.5*(Ix2+Iy2)+sqrt(Ixy.^2+(0.5*(Ix2-Iy2)).^2);
lambda2 = 0.5*(Ix2+Iy2)-sqrt(Ixy.^2+(0.5*(Ix2-Iy2)).^2);

% ================ Harris feature function ===============
alpha = 0.06; % by convention
% cim = abs(lambda1.*lambda2-alpha*(lambda1+lambda2).^2);
k = 0.04; cim = (Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2;

% ================ extraing local maxima =================
if ANMS % adaptive non-maximal suppression method
    % Simply look for local maxima in the interest function
    % can lead to an uneven distribution of feature points across
    % the image, e.g., points will be denser in regions of higher
    % contrast. So we extract local maxima by performing a grey
    % scale morphological dilation and then finding points in the
    % corner strength image that match the dilated image and are
    % also greater than the threshold.


    sze = 2*radius+1;                   % Size of mask.
    mx = ordfilt2(cim,sze^2,ones(sze)); % the sum of response value of all neighbours

    % set threshold as % of the maximum value
    threshold = t_percent*max(max(mx));

    cim_ones = (cim==mx)&(cim>threshold);
    cim = cim_ones.*cim;
    % sort auto-correlation values in ascending order
    [r,c,value] = find(cim);       % Find row,col coords.
    [value,idx] = sort(value,'descend');
    coords = [r(idx),c(idx)];

else % directly extract local maxima
    threshold = t_percent*max(max(cim));
    cim = (cim>threshold);
    [r,c,value] = find(cim);
    [value,idx] = sort(value,'descend');
    coords = [r(idx),c(idx)];
end

% code from below removes all the corners that are within 32 pixels
% from the edges of the image, this is done so that
% in Q1,2b patches of 32x32 can be obtained with meaningful data
index = [];
for i = 1:size(coords, 1)
    if~(coords(i,1)>32 && coords(i,2)>32 && coords(i,1)<maxheight-32 && coords(i,2)<maxwidth-32)
        index(end+1) = i;
    end
end
coords(index,:)=[];

end
```

Listing 3. Q1.2.b - Feature Descriptor

```
function [patches] = getDescriptor(coords, img, d)
% INPUT
% ==========
% coords    : the coordinates of interest points
% img       : the image
% d         : the diameter of circle descriptor
%
% OUTPUT
% ==========
% patches   : the descriptor patch around the interest point

%Q1 Automatic
%2b) Obtain Descriptor
% this function takes pairs of coordinates and returns the 31x31 pixel
% circle-shaped patches around the given coordinates

% ========================================================================
% Alternatively, if we insist to use square-shaped patches around the
% interest points, what we can do is to estimate a dominant orientation.
% Then an oriented patch can be extracted and used to form a feature
% descriptor.

% The simplest orientation estimate is the average gradient within a region
% around the interest point.
% And in order to make this method more reliable, the aggregation window
% (circle-shaped) should be larger than the detecting window.

% Sometimes, however, the averaged (signed) gradient in a region can be
% very small and therefore an unreliable indicator of orientation. A more
% reliable method is to look at the histogram of orientations (e.g.36 bins)
% computed around the interest point.

% Note here, this alternative method is not implemented here.
% ========================================================================

if size(size(img)) == 3
    img = rgb2gray(img);
end
edges = 1:2:256;
radius = floor(d/2);
circle_mask  = fspecial('disk',radius)>0;

% ========== get all descriptor patches ===========
patches = [];
for i=1:size(coords, 1)
    patch = img(coords(i,1)-radius:coords(i,1)+radius,...
            coords(i,2)-radius:coords(i,2)+radius);
    patch = patch.*uint8(circle_mask); % circle patch
    N = histcounts(patch, edges);
    patches(i,:) = N(1,:);
end
```

Listing 4. Q1.2.c - Feature Matching

```
function [matchedPoints1, matchedPoints2] = myFeatureMatching(C1,C2,D1,D2,threshold)
% INPUT:
% ========================================================================
% C1        : a list of coordinates of detected interest points in image A
% C2        : a list of coordinates of detected interest points in image B
% D1        : feature descriptors of detected interest points in image A
% D2        : feature descriptors of detected interest points in image B
% threshold : the ratio between 1-NN (first NN) and 2-NN (second NN)
%
```

```
% OUTPUT:
% ==========================================================================
% matchedPoints1 : coordinates of matched interest points in image A
% matchedPoints2 : coordinates of matched interest points in image B

    x_size = size(D1,1);
    y_size = size(D2,1);
    for i = 1:x_size
        tmp_distance = [];
        for j = 1:y_size
            tmp_distance(j) = sum(abs(D1(i,:)-D2(j,:)))/y_size;
        end

        [value, idx] = sort(tmp_distance,'ascend');
        min_v = value(1);

        % 1-NN to 2-NN ratio threshold
        snd_min_v = value(2);
        if min_v > (snd_min_v/threshold)
            min_v = inf; %indicating corresponding descriptor not matched
        end

        distance(i) = min_v;
        index(i) = idx(1);
    end



    % Now filter out matches with distance inf
    [distance,idx] = sort(distance,'ascend');
    matchedIP_idx = idx(~isinf(distance)); % the index of matched interest points

    idx_matchedIP_img2_temp = index;
    idx_matchedIP_img1_temp = 1:1:size(index,2);
    idx_matchedIP_img1 = idx_matchedIP_img1_temp(matchedIP_idx);
    idx_matchedIP_img2 = idx_matchedIP_img2_temp(matchedIP_idx);

    % remove correspondance that are not matched uniquely (which are ambiguous)
    if threshold ~= 1
        arr1 = getNonRepeatableElementIdx(idx_matchedIP_img1);
        arr2 = getNonRepeatableElementIdx(idx_matchedIP_img2);
        idx = intersect(arr1,arr2);
        correspondance = [idx_matchedIP_img1(idx);idx_matchedIP_img2(idx)];
        if size(correspondance,2)==0
           error('user defined error - no interest point is matched under current parameters');
        end
    else
        correspondance = [idx_matchedIP_img1;idx_matchedIP_img2];
    end
    matchedPoints1=C1(correspondance(1,:),:);
    matchedPoints2=C2(correspondance(2,:),:);

end

function [idx] = getNonRepeatableElementIdx(input_array)
% this function returns the index of non-repeatable elements in the input
% array
[u,idx_u,~] = unique(input_array,'stable');
[u_sorted,idx_u_sorted] = sort(u,'ascend');
temp = histc(input_array,u_sorted);
[~, idx_nonRepeat] = find(temp==1);
idx = idx_u(idx_u_sorted(idx_nonRepeat));
idx = sort(idx,'ascend');
end
```

Listing 5. Q1.3.a - Estimate Homography Matrix

```
function [besth, h, inliers] = RansacPredictH(m1, m2, numberoftrial, threshold)
% INPUT
% ================================================================================
% m1 : Coordinates of matched interest points in image 1. The columns are
%      coordinates and the rows are different interest points
%
% m2 : Coordinates of matched interest points in image 2. The columns are
%      coordinates and the rows are different interest points
%
% numberoftrail : The number of RANSAC iterations
%
% threshold     : RANSAC threshold. In our implementation, it stands for
%                 the Manhatten distance between the matched points in
%                 image 2 and the projected correspondence.
%
% OUTPUT
% ================================================================================
% besth   : The best homography matrix produced by RANSAC iterations
%
% inliers : Coordinates of points that are within the pre-specified
%           threshold when transformation is performed by the besth
%
% h       : Homography matrix obtained by using method in slide 20 with
%           inliers calculated from RANSAC

    if size(m1,1) < 4
        error('given fewer than 4 points');
    end
    newm1 = [];
    newm2 = [];
    [inliers(:), besth] = myRansac(m1, m2, numberoftrial, threshold);
    for i = 1:size(inliers, 2)
       if(inliers(i)==1)
        newm1(end+1,:) = m1(i,:);
        newm2(end+1,:) = m2(i,:);
       end
    end
    h = gethmatrix(newm1, newm2);
end

function [inliers, besth] = myRansac(m1, m2, numberoftrial, mythreshold)
    N_Trials = numberoftrial;
    threshold = mythreshold;
    Max_N_Inliers = -1;
    besth = zeros(3,3);
    for i = 1:N_Trials
        [distance, h] = myEstTFormDistance(m1, m2);
        [Inliers, N_Inliers] = findInliers(distance, threshold);

        if Max_N_Inliers < N_Inliers
            Max_N_Inliers = N_Inliers;
            inliers = Inliers;
            besth = h;
        end
    end
end

function [d, h] = myEstTFormDistance(m1, m2)
    sizeofindex = size(m1,1);
    indices = generateRandomIndices(sizeofindex);
    h = gethmatrix(m1(indices,:), m2(indices,:));
```

```
    d = computeDistance(m1, m2, h);
end



function [indices] = generateRandomIndices(sizeofindex)
    indices = zeros(4,1);
    for i = 1:4
        uniqueflag = false;
        while ~uniqueflag
            tmp = randi(sizeofindex);
            if(sum(find(indices == tmp))==0)
                uniqueflag = true;
            end
        end
        indices(i) = tmp;
    end
end

function d = computeDistance(m1, m2, h)
     projectedv2 = hmatrixproject(m1, h);
     d = sum(abs(projectedv2-m2),2);
end

function [inliers, N_Inliers] = findInliers(distance, threshold)

    inliers = [];
    N_Inliers = 0;

    for i = 1: size(distance, 1)
        if (distance(i) <= threshold)
            inliers(i) = true;
            N_Inliers = N_Inliers + 1;
        else
            inliers(i) = false;
        end
    end
end

function h = gethmatrix(m1, m2)
%3A) obtain homography matrix
n = size(m1, 1);
% Solve equations using SVD
x1 = m1(:,1)';
y1 = m1(:,2)';
x2 = m2(:,1)';
y2 = m2(:,2)';
rowofzero = zeros(3, n);
rowofxy = -[x1; y1; ones(1,n)];
hx = [rowofxy; rowofzero; x2.*x1; x2.*y1; x2];
hy = [rowofzero; rowofxy; y2.*x1; y2.*y1; y2];
h = [hx hy];
[U, D, V] = svd(h');
h = (reshape(V(:,9), 3, 3)).';
end
```

Listing 6. Q1.3.b - Estimate Fundamental Matrix

```
function [besth, h, inliers] = RansacPredictH(m1, m2, numberoftrial, threshold)
% INPUT
% ================================================================================
% m1 : Coordinates of matched interest points in image 1. The columns are
%      coordinates and the rows are different interest points
%
```

```
% m2 : Coordinates of matched interest points in image 2. The columns are
%      coordinates and the rows are different interest points
%
% numberoftrail : The number of RANSAC iterations
%
% threshold     : RANSAC threshold. In our implementation, it stands for
%                 the Manhatten distance between the matched points in
%                 image 2 and the projected correspondence.
%
% OUTPUT
% ============================================================================
% besth   : The best homography matrix produced by RANSAC iterations
%
% inliers : Coordinates of points that are within the pre-specified
%           threshold when transformation is performed by the besth
%
% h       : Homography matrix obtained by using method in slide 20 with
%           inliers calculated from RANSAC

    if size(m1,1) < 4
        error('given fewer than 4 points');
    end
    newm1 = [];
    newm2 = [];
    [inliers(:), besth] = myRansac(m1, m2, numberoftrial, threshold);
    for i = 1:size(inliers, 2)
       if(inliers(i)==1)
        newm1(end+1,:) = m1(i,:);
        newm2(end+1,:) = m2(i,:);
       end
    end
    h = gethmatrix(newm1, newm2);
end

function [inliers, besth] = myRansac(m1, m2, numberoftrial, mythreshold)
    N_Trials = numberoftrial;
    threshold = mythreshold;
    Max_N_Inliers = -1;
    besth = zeros(3,3);
    for i = 1:N_Trials
        [distance, h] = myEstTFormDistance(m1, m2);
        [Inliers, N_Inliers] = findInliers(distance, threshold);

        if Max_N_Inliers < N_Inliers
            Max_N_Inliers = N_Inliers;
            inliers = Inliers;
            besth = h;
        end
    end
end

function [d, h] = myEstTFormDistance(m1, m2)
    sizeofindex = size(m1,1);
    indices = generateRandomIndices(sizeofindex);
    h = gethmatrix(m1(indices,:), m2(indices,:));
    d = computeDistance(m1, m2, h);
end



function [indices] = generateRandomIndices(sizeofindex)
    indices = zeros(4,1);
    for i = 1:4
        uniqueflag = false;
```

```
        while ~uniqueflag
            tmp = randi(sizeofindex);
            if(sum(find(indices == tmp))==0)
                uniqueflag = true;
            end
        end
        indices(i) = tmp;
    end
end

function d = computeDistance(m1, m2, h)
     projectedv2 = hmatrixproject(m1, h);
     d = sum(abs(projectedv2-m2),2);
end

function [inliers, N_Inliers] = findInliers(distance, threshold)

    inliers = [];
    N_Inliers = 0;

    for i = 1: size(distance, 1)
        if (distance(i) <= threshold)
            inliers(i) = true;
            N_Inliers = N_Inliers + 1;
        else
            inliers(i) = false;
        end
    end
end

function h = gethmatrix(m1, m2)
%3A) obtain homography matrix
n = size(m1, 1);
% Solve equations using SVD
x1 = m1(:,1)';
y1 = m1(:,2)';
x2 = m2(:,1)';
y2 = m2(:,2)';
rowofzero = zeros(3, n);
rowofxy = -[x1; y1; ones(1,n)];
hx = [rowofxy; rowofzero; x2.*x1; x2.*y1; x2];
hy = [rowofzero; rowofxy; y2.*x1; y2.*y1; y2];
h = [hx hy];
[U, D, V] = svd(h');
h = (reshape(V(:,9), 3, 3)).';
end
```

Listing 7. Q1.3.c - Projection Using Homography

```
function projection = hmatrixproject(toBeProjected, besth)
% INPUT
% ========================================================================
% toBeProjected : The coordinates of points to be projected in image 1
% besth         : The best homography matrix onbtained from RansacPredictH
%
% OUTPUT
% ========================================================================
% projection : the coordinates of projected points
x_size = size(toBeProjected,1);
for i = 1:x_size
   tmp=besth*[toBeProjected(i,:),1]';
   q = tmp(3,:);
   projection(i,:) = [tmp(1,:)./q tmp(2,:)./q];
end
```

```
end
```

Listing 8. Q1.3.c - Calculate Homography Accuracy

```
function HA = calculateHA(v1, v2)
% INPUT
% ========================================================================
% v1 : a set of projections from image 1 to image 2
% v2 : the matched points in image 2, which corresponds to the points that
%      are to be projected from image 1 to image 2
% OUTPUT
% ========================================================================
% HA : Homography error (weighted sum of cityblock distance)

ydif = sum(abs(v1(:,1)-v2(:,1)))/size(v1,1);
xdif = sum(abs(v1(:,2)-v2(:,2)))/size(v1,1);
HA = ydif+xdif;
end
```

Listing 9. Q1.3.d - Calculate and Display Epipolar Lines

```
function [] = drawEpipolarline(f, v, img)
% INPUT
% ========================================================================
% f   : fundamental matrix
% v   : inliers of matched correspondences
%       row     -  different correspondences
%       column  -  x and y coordinates
% img : the image that the plot is based on
%
% OUTPUT
% ========================================================================
% HA : Homography error (weighted sum of cityblock distance)

imagesc(img), axis ([0 size(img,2) 0 size(img,1)]), colormap(gray), hold on
epiLines = epipolarLine(f,v);
points = lineToBorderPoints(epiLines,[size(img,2),size(img,1)]);
line(points(:,[2,4])',points(:,[1,3])');
plot(v(:,2),v(:,1),'ro','LineWidth',2);
truesize;
hold off
end
```

Listing 10. Q2.1.a (Harris)

```
% Experimental results
% Q2.1

% read images
factor = 2; % reduce the size of the second image by factor
img1 = imread('pictures/HG1.jpg');
img1 = imresize(img1,[680 850]);
img2 = imread('pictures/HG1.jpg');
img2 = imresize(img2,[680/factor 850/factor]);

% convert colour scale to gray scale
% img 1
if (size(size(img1),2) == 3)
    img1 = rgb2gray(img1);
end

% img 2
if (size(size(img2),2) == 3)
    img2 = rgb2gray(img2);
end
```

```
21
22
23
24 %% paramaters that can be adjusted
25 threshold = 0.01; % 1% of the maximum Harris response value
26 visualise = 1;
27 knn_threshold = 1.07;
28 num_strongest = 500; % number of strongest interest points
29 ANMS = true; % adaptive non-maximal suppression
30 radius = 2;
31 descriptor_type = 'CIRCLE';
32 descriptor_diameter = 32; % diameter of circle descriptor
33 num_manual_IPs = 10; % number of interest points get from manual selection
34
35 %% Get automatic interest points and features
36
37 % image 1 Harris points and feature descriptors
38 % ========================================================================
39 points1_raw = InterestPointDetector(img1, threshold, ANMS, radius);
40 if size(points1_raw,1)>num_strongest
41     points1 = points1_raw(1:num_strongest,:);
42 else
43     points1 = points1_raw;
44 end
45
46 if (descriptor_type == 'CIRCLE')
47     % Obtain circle patches as descriptors
48     descriptor1 = getDescriptor(points1(:,1:2), img1, descriptor_diameter);
49 elseif (descriptor_type == 'SIFT')
50     % Obtain SIFT descriptor
51     descriptor1 = [];
52     for i=1:size(points1,1)
53         img1_    = vl_imsmooth(im2double(img1), sqrt(points1(i,3)^2 - 0.5^2));
54         [Ix, Iy] = vl_grad(img1_) ;
55         mod      = sqrt(Ix.^2 + Iy.^2) ;
56         ang      = atan2(Iy,Ix) ;
57         grd      = shiftdim(cat(3,mod,ang),2) ;
58         grd      = single(grd) ;
59         descriptor1(i,:) = vl_siftdescriptor(grd, ...
60         [points1(i,2),points1(i,1),points1(i,3:4)]') ;
61     end
62 end
63
64 % image 2 Harris points and feature descriptors
65 % ========================================================================
66 points2_raw = InterestPointDetector(img2, threshold, ANMS, radius);
67 if size(points2_raw,1)>num_strongest
68     points2 = points2_raw(1:num_strongest,:);
69 else
70     points2 = points2_raw;
71 end
72
73 if (descriptor_type == 'CIRCLE')
74     % Obtain circle patches as descriptors
75     descriptor2 = factor*getDescriptor(points2(:,1:2), img2, descriptor_diameter/factor);
76 elseif (descriptor_type == 'SIFT')
77     % Obtain SIFT descriptor
78     descriptor2 = [];
79     for i=1:size(points2,1)
80         img2_    = vl_imsmooth(im2double(img2), sqrt(points2(i,3)^2 - 0.5^2));
81         [Ix, Iy] = vl_grad(img1_) ;
82         mod      = sqrt(Ix.^2 + Iy.^2) ;
83         ang      = atan2(Iy,Ix) ;
84         grd      = shiftdim(cat(3,mod,ang),2) ;
```

```
        grd      = single(grd) ;
        descriptor2(i,:) = vl_siftdescriptor(grd, ...
        [points2(i,2),points2(i,1),points2(i,3:4)]') ;
    end
end

%% visualise Detected Interest Points before matching
if visualise
    f1=figure;
    subplot(1,2,1);
    myvisualise(points1,img1,'Img1 - Interest Points');
    subplot(1,2,2);
    myvisualise(points2,img2,'Img2 - Interest Points')
    set(f1, 'Units', 'normalized', 'Position', [0,0,1,1]);
%     print(f1,'-dpng','-r600');
end

%% Matching correspondance
[matchedPoints1, matchedPoints2] = myFeatureMatching(points1(:,1:2),...
                points2(:,1:2), descriptor1, descriptor2, knn_threshold)

%% visualise matched correspondences
if visualise
figure; ax = axes;
showMatchedFeatures(img1,img2,fliplr(matchedPoints1),fliplr(matchedPoints2),'montage','Parent',ax);
title(ax, 'Matched Correspondances before RANSAC');
legend(ax, 'Matched points 1','Matched points 2');
end

%% Q2_1_a -
% Compare the interest points obtained in these two cases using HA error.
% ========================================================================
numberoftrial = 10000;
threshold = 4; % cityblock distance in pixels
[besth, ~, inliersIndex] = RansacPredictH(matchedPoints1, matchedPoints2, numberoftrial, threshold);

matchedPoints1_inliers = [];
matchedPoints2_inliers = [];
cnt = 1;
for i=1:size(matchedPoints1,1)
    if (inliersIndex(i) == 1)
        matchedPoints1_inliers(cnt,:) = matchedPoints1(i,:);
        matchedPoints2_inliers(cnt,:) = matchedPoints2(i,:);
        cnt = cnt +1;
    end
end

projectedv2 = hmatrixproject(matchedPoints1_inliers, besth);

% Compute HA error
HA = calculateHA(projectedv2, matchedPoints2_inliers);

%%  visualise matched correspondences after RANSAC
if visualise
f_auto=figure; ax = axes;
showMatchedFeatures(img1,img2,fliplr(matchedPoints1_inliers),...
    fliplr(matchedPoints2_inliers),'montage','Parent',ax);
title(ax, 'Matched Correspondences after RANSAC - Auto');
legend(ax, 'Matched points 1','Matched points 2');
% print(f_auto,'-dpng','-r600');
end
```

Listing 11. Q2.1.b

```
%% Q2_1_b using SIFT
%% parameters to be adjusted
knn_threshold = 1.4;
num_strongest = 500; % number of strongest interest points
visualise = 1;
num_manual_IPs = 10; % number of manually selected interest points
%% read image
% img1 = imread('pictures/img1.pgm');
% img2 = imread('pictures/img2.pgm');
img1 = imread('pictures/HG1.JPG');
img2 = imread('pictures/HG2.JPG');

img1 = imresize(img1,[680 850]);
img2 = imresize(img2,[680 850]);

% initialisation
addpath('external/vlfeat-0.9.18');
addpath('external/libsvm-3.18/matlab');
run('external/vlfeat-0.9.18/toolbox/vl_setup.m'); % vlfeat library
cd('external/libsvm-3.18/matlab'); % libsvm library
run('make');
cd('../../..');

% ================= convert colour scale to gray scale =================
% img 1
if (size(size(img1),2) == 3)
    img1 = rgb2gray(img1);
end
img1 = im2double(img1);
% img 2
if (size(size(img2),2) == 3)
    img2 = rgb2gray(img2);
end
img2 = im2double(img2);


%% Get Manual interest points and features
% [points1_manual, points2_manual] = getManualIPs(img1,img2,num_manual_IPs);
load lizhang_manual_points.mat

if visualise
    f_manual = figure; ax = axes;
    showMatchedFeatures(img1,img2,fliplr(points1_manual),...
        fliplr(points2_manual),'montage','Parent',ax);
    h = title('Manually Matched Points');
    set(h,'FontSize',22);
    set(f_manual, 'Units', 'normalized', 'Position', [0,0,0.6,0.6]);
    % print(f_manual,'-dpng','-r600');
end

% estimate homography matrix and HA
numberoftrial = 1000;
threshold = 6.7; % pixels - 6.7 is chosen such that all manually selected
                 % points are inliers
[besth_manual, f2, inliersIndex] = RansacPredictH(points1_manual, points2_manual, numberoftrial,
    threshold);

points1_manual_inliers = [];
points2_manual_inliers = [];
cnt = 1;
for i=1:size(points1_manual,1)
    if (inliersIndex(i) == 1)
        points1_manual_inliers(cnt,:) = points1_manual(i,:);
        points2_manual_inliers(cnt,:) = points2_manual(i,:);
```

```
        cnt = cnt +1;
    end
end

projectedv2 = hmatrixproject(points1_manual, besth_manual);

% Compute HA error
HA = calculateHA(projectedv2, points2_manual);


%%  visualise matched correspondences after RANSAC (Manual)
if visualise
    f_auto_manual=figure; ax = axes;
    showMatchedFeatures(img1,img2,fliplr(points1_manual_inliers),...
        fliplr(points2_manual_inliers),'montage','Parent',ax);
    title(ax, 'Matched Correspondences after RANSAC – Manual');
    legend(ax, 'Matched points 1','Matched points 2');
    % print(f_auto,'-dpng','-r600');
end









% ================================================================================
% ================================================================================
% ================================================================================
% ================================================================================
% ========================Here Goes Automatic=============================
% ========================Here Goes Automatic=============================
% ========================Here Goes Automatic=============================
% ========================Here Goes Automatic=============================
% ================================================================================
% ================================================================================
% ================================================================================
% ================================================================================
% ================================================================================

%% extract interest points & their descriptors
[F1_raw,D1_raw]=vl_sift(single(img1));
[F2_raw,D2_raw]=vl_sift(single(img2));
coords1_sift_raw = horzcat(F1_raw(2,:)',F1_raw(1,:)');
coords2_sift_raw = horzcat(F2_raw(2,:)',F2_raw(1,:)');

%% get strong interest points
[~,idx1] = sort(F1_raw(4,:),'descend');
[~,idx2] = sort(F2_raw(4,:),'descend');

%% get the strongest interest points and their descriptors
F1 = F1_raw(:,idx1(1:num_strongest));
F2 = F2_raw(:,idx2(1:num_strongest));
D1 = D1_raw(:,idx1(1:num_strongest));
D2 = D2_raw(:,idx2(1:num_strongest));
coords1_sift = coords1_sift_raw(idx1(1:num_strongest),:);
coords2_sift = coords2_sift_raw(idx2(1:num_strongest),:);
points1_sift = horzcat(coords1_sift,F1(3:4,:)');
points2_sift = horzcat(coords2_sift,F2(3:4,:)');

%% visualise the strongest N points in both images
if visualise
    figure;
    subplot(1,2,1);
    imshow(img1);
    hold on ;
```

```
    vl_plotframe(F1);
    title('Interest Points (SIFT)');
    subplot(1,2,2);
    imshow(img2);
    hold on ;
    vl_plotframe(F2);
    title('Interest Points (SIFT)');
end

%% Matching correspondence
[matchedPoints1_sift, matchedPoints2_sift] = myFeatureMatching(coords1_sift,...
                coords2_sift, D1', D2', knn_threshold)

%% visualise matched correspondences
if visualise
figure;
showMatchedFeatures(img1,img2,fliplr(matchedPoints1_sift),fliplr(matchedPoints2_sift),'montage');
title('Matched Correspondences');
legend('Matched points 1','Matched points 2');
end

%% Compute homography matrix
% Using 10 matched correspondences, in order to be consistent with the
% number manual points. The result is averaged

numberoftrial = 10000;
threshold = 6.7; % average pixels
num_loop = 10;

% init
HA = zeros(num_loop,1);

for iter = 1:num_loop
[rand_points1,idx1] = datasample(matchedPoints1_sift,size(points1_manual,1),1,'Replace',false);
rand_points2 = matchedPoints2_sift(idx1,:);

[besth_auto, f2, inliersIndex] = RansacPredictH(rand_points1, rand_points2, numberoftrial, threshold);

matchedPoints1_inliers = [];
matchedPoints2_inliers = [];
cnt = 1;
for i=1:size(points1_manual,1)
    if (inliersIndex(i) == 1)
        matchedPoints1_inliers(cnt,:) = rand_points1(i,:);
        matchedPoints2_inliers(cnt,:) = rand_points2(i,:);
        cnt = cnt +1;
    end
end

projectedv2 = hmatrixproject(matchedPoints1_inliers, besth_auto);

% Compute HA error
HA(iter,1) = calculateHA(projectedv2, matchedPoints2_inliers);
end
HA_ave_auto = mean(HA);

%%  visualise matched correspondences after RANSAC
if visualise
f_auto_manual=figure;
subplot(121);
showMatchedFeatures(img2,img2,fliplr(matchedPoints2_inliers),...
    fliplr(projectedv2),'PlotOptions',{'ro','g+','y-'});
title('Auto - Projected Inliers vs. Matched Inliers','FontSize',22);
l1 = legend('Inliers in Image 2', 'Projection');
```

```
set(l1,'FontSize',16);

subplot(122);
projectedv2_manual = hmatrixproject(matchedPoints1_inliers, besth_manual);
showMatchedFeatures(img2,img2,fliplr(matchedPoints2_inliers),...
    fliplr(projectedv2_manual),'PlotOptions',{'ro','g+','y-'});
title('Manual - Projected Inliers vs. Matched Inliers','FontSize',22);
l2=legend('Inliers in Image 2','Projection');
set(l2,'FontSize',16);
set(f_auto_manual, 'Units', 'normalized', 'Position', [0,0,1,1]);
% print(f_auto_manual,'-dpng','-r600');
end
```

Listing 12. Q2.1.c

```
%% Q2_1_C using SIFT
% Estimate homography from different number of correspondences from Q1.2
% starting from the minimum number up to the maximum number of available
% pairs. Report and discuss HA for different number of correspondences.
% Find the number of outliers in your list of automatic correspondences
% and explain your approach to that.

%% parameters to be adjusted
knn_threshold = 1.4;
num_strongest = 500; % number of strongest interest points
visualise = 1;

%% read image
% img1 = imread('pictures/scene1.ppm');
% img2 = imread('pictures/scene2.ppm');
% img1 = imread('pictures/img1.pgm');
% img2 = imread('pictures/img6.pgm');
img1 = imread('pictures/HG1.JPG');
img1 = imresize(img1,[680 850]);
img2 = imread('pictures/HG2.JPG');
img2 = imresize(img2,[680 850]);

% initialisation
addpath('external/vlfeat-0.9.18');
addpath('external/libsvm-3.18/matlab');
run('external/vlfeat-0.9.18/toolbox/vl_setup.m'); % vlfeat library
cd('external/libsvm-3.18/matlab'); % libsvm library
run('make');
cd('../../..');

% ================= convert colour scale to gray scale =================
% img 1
if (size(size(img1),2) == 3)
    img1 = rgb2gray(img1);
end
img1 = im2double(img1);
% img 2
if (size(size(img2),2) == 3)
    img2 = rgb2gray(img2);
end
img2 = im2double(img2);


%% extract interest points & their descriptors
[F1_raw,D1_raw]=vl_sift(single(img1));
[F2_raw,D2_raw]=vl_sift(single(img2));
coords1_sift_raw = horzcat(F1_raw(2,:)',F1_raw(1,:)');
coords2_sift_raw = horzcat(F2_raw(2,:)',F2_raw(1,:)');

```

```
%% sort by confidence in descending order
[~,idx1] = sort(F1_raw(4,:),'descend');
[~,idx2] = sort(F2_raw(4,:),'descend');

%% get the strongest interest points and their descriptors
F1 = F1_raw(:,idx1(1:num_strongest));
F2 = F2_raw(:,idx2(1:num_strongest));
D1 = D1_raw(:,idx1(1:num_strongest));
D2 = D2_raw(:,idx2(1:num_strongest));
coords1_sift = coords1_sift_raw(idx1(1:num_strongest),:);
coords2_sift = coords2_sift_raw(idx2(1:num_strongest),:);
points1_sift = horzcat(coords1_sift,F1(3:4,:)');
points2_sift = horzcat(coords2_sift,F2(3:4,:)');


%% visualise the strongest N points in both images
if visualise
    figure;
    subplot(121);
    myvisualise(points1_sift, img1, 'very original corner image 1');
    subplot(122);
    myvisualise(points2_sift, img2, 'very original corner image 2');
end

%% Matching correspondence
[matchedPoints1_sift, matchedPoints2_sift] = myFeatureMatching(coords1_sift,...
                coords2_sift, D1', D2', knn_threshold);
%% visualise matched correspondences before RANSAC
if visualise
    figure; ax = axes;
    showMatchedFeatures(img1,img2,fliplr(matchedPoints1_sift),fliplr(matchedPoints2_sift),'montage','
        Parent',ax);
    title(ax, 'our own implementation');
    legend(ax, 'Matched points 1','Matched points 2');
end

%% Compute ground truth inliers and outliers
numberoftrial = 10000;
threshold = 2; % average pixels

[besth, f2, inliersIndex] = RansacPredictH(matchedPoints1_sift, matchedPoints2_sift, numberoftrial,
    threshold);
matchedPoints1_inliers_ground_truth = [];
matchedPoints2_inliers_ground_truth = [];
cnt = 1;
cnt_out = 1;
outliers1 = [];
outliers2 = [];
% obtain coordinates of inliers and outliers
for i=1:size(matchedPoints1_sift,1)
    if (inliersIndex(i) == 1)
        matchedPoints1_inliers_ground_truth(cnt,:) = matchedPoints1_sift(i,:);
        matchedPoints2_inliers_ground_truth(cnt,:) = matchedPoints2_sift(i,:);
        cnt = cnt +1;
    else
        outliers1(cnt_out,:) = matchedPoints1_sift(i,:);
        outliers2(cnt_out,:) = matchedPoints2_sift(i,:);
        cnt_out = cnt_out + 1;
    end
end
projectedv2 = hmatrixproject(matchedPoints1_inliers_ground_truth, besth);

%% visualise ground truth
if visualise
```

```
    f_auto=figure; ax = axes;
    showMatchedFeatures(img1,img2,fliplr(matchedPoints1_inliers_ground_truth),...
        fliplr(matchedPoints2_inliers_ground_truth),'montage','Parent',ax);
    h = title(ax, 'Ground Truth');
    set(h,'FontSize',22);
    legend(ax, 'Matched points 1','Matched points 2');
    % print(f_auto,'-dpng','-r600');
end

%% visualise Outliers obtained when computing ground truth
if visualise
    f_auto=figure; ax = axes;
    showMatchedFeatures(img1,img2,fliplr(outliers1),...
        fliplr(outliers2),'montage','Parent',ax);
    h = title(ax, 'Ground Truth');
    set(h,'FontSize',22);
    legend(ax, 'Matched points 1','Matched points 2');
    % print(f_auto,'-dpng','-r600');
end


%% Find the relationship between the number of correspondence and HA
% params
numberoftrial = 1000;
threshold = 6.7; % average pixels
num_loop = 100;
step = 1;
num_points = 4:step:size(matchedPoints1_sift,1);
% num_points = 4:step:25;
num_step = size(num_points,2);

% init
HA = zeros(num_loop,num_step);

for n_step = 1:num_step
for iter = 1:num_loop
[rand_points1,idx1] = datasample(matchedPoints1_sift,num_points(n_step),1,'Replace',false);
rand_points2 = matchedPoints2_sift(idx1,:);

[testh, f2, inliersIndex] = RansacPredictH(rand_points1, rand_points2, numberoftrial, threshold);

matchedPoints1_inliers = [];
matchedPoints2_inliers = [];
cnt = 1;
for i=1:size(rand_points1,1)
    if (inliersIndex(i) == 1)
        matchedPoints1_inliers(cnt,:) = rand_points1(i,:);
        matchedPoints2_inliers(cnt,:) = rand_points2(i,:);
        cnt = cnt +1;
    end
end


projectedv2 = hmatrixproject(matchedPoints1_inliers_ground_truth, testh);



% Compute HA error
HA(iter,n_step) = calculateHA(projectedv2, matchedPoints2_inliers_ground_truth);
num_outliers(iter,n_step) = num_points(n_step) - size(matchedPoints1_inliers,1);
end
end
HA_ave_auto = mean(HA,1);
HA_std_auto = std(HA,0,1);
```

```
176
177
178 %% visualise average HA error against number of correspondences
179 if visualise
180     f = figure;
181     [a,h1,h2]=plotyy(num_points, HA_ave_auto, num_points, HA_std_auto);
182     xlim(a,[num_points(1) num_points(end)]);
183     xlabel('Number of Correspondances',...
184         'FontWeight','Bold','FontSize',24);
185     ylabel(a(1),'Average HA (pixels)',...
186         'FontWeight','Bold','FontSize',24);
187     ylabel(a(2),'Standard Deviation',...
188         'FontWeight','Bold','FontSize',24);
189     set(a(1),'FontSize',20);
190     set(a(2),'FontSize',20);
191     set(a, 'XTick', num_points(1):2:num_points(end));
192     y1 = ylim(a(1)); y2 = ylim(a(2));
193     set(a(1), 'YTick', linspace(y1(1),y1(2),5));
194     set(a(2), 'YTick', linspace(y2(1),y2(2),5));
195     set(h1,'LineWidth',3,'LineStyle','--');
196     set(h2,'LineWidth',3,'LineStyle',':');
197     set(f, 'Units', 'normalized', 'Position', [0.2,0.2,0.8,0.8]);
198     title('HA against the NO. of Correspondences');
199     legend('Averaged HA', 'Standard Deviation');
200     % print(f,'-dpng','-r600');
201 end
```

Listing 13. Q2.2.a & b (HARRIS)

```
1  %% Q2_2_a_b HARRIS
2  % a) Estimate fundamental matrix using list of correspondences from Q1.1 or
3  % Q1.2.a.
4  % b) Calculate the epipoles for images A and B. Show epipolar lines and
5  % epipoles on the images if possible.
6
7  %% parameters to be adjusted
8  threshold = 0.01;  % 1 percent of maximum response value
9  visualise = 1;
10 knn_threshold = 1.11;
11 num_strongest = 500; % number of strongest interest points
12 ANMS = true; % adaptive non-maximal suppression
13 radius = 5;
14 descriptor_type = 'CIRCLE';
15 descriptor_diameter = 32; % diameter of circle descriptor
16 num_manual_IPs = 10; % number of interest points get from manual selection
17
18 %% read image
19 img1 = imread('pictures/FD1.JPG');
20 img1 = imresize(img1,[680 850]);
21 img2 = imread('pictures/FD2.JPG');
22 img2 = imresize(img2,[680 850]);
23
24 if (descriptor_type ~= 'CIRCLE')
25     % initialisation
26     addpath('external/vlfeat-0.9.18');
27     addpath('external/libsvm-3.18/matlab');
28     run('external/vlfeat-0.9.18/toolbox/vl_setup.m'); % vlfeat library
29     cd('external/libsvm-3.18/matlab'); % libsvm library
30     run('make');
31     cd('../../..');
32 end
33
34 % ================= convert colour scale to gray scale =================
35 % img 1
```

```
if (size(size(img1),2) == 3)
    img1 = rgb2gray(img1);
end

% img 2
if (size(size(img2),2) == 3)
    img2 = rgb2gray(img2);
end


%% extract interest points & their descriptors
% image 1 Harris points and feature descriptors
% ================================================================================
points1_raw = InterestPointDetector(img1, threshold, ANMS, radius);
if size(points1_raw,1)>num_strongest
    points1 = points1_raw(1:num_strongest,:);
else
    points1 = points1_raw;
end

if (descriptor_type == 'CIRCLE')
    % Obtain circle patches as descriptors
    descriptor1 = getDescriptor(points1(:,1:2), img1, descriptor_diameter);
elseif (descriptor_type == 'SIFT')
    % Obtain SIFT descriptor
    descriptor1 = [];
    for i=1:size(points1,1)
        img1_    = vl_imsmooth(im2double(img1), sqrt(points1(i,3)^2 - 0.5^2));
        [Ix, Iy] = vl_grad(img1_) ;
        mod      = sqrt(Ix.^2 + Iy.^2) ;
        ang      = atan2(Iy,Ix) ;
        grd      = shiftdim(cat(3,mod,ang),2) ;
        grd      = single(grd) ;
        descriptor1(i,:) = vl_siftdescriptor(grd, ...
        [points1(i,2),points1(i,1),points1(i,3:4)]') ;
    end
end

% image 2 Harris points and feature descriptors
% ================================================================================
points2_raw = InterestPointDetector(img2, threshold, ANMS, radius);
if size(points2_raw,1)>num_strongest
    points2 = points2_raw(1:num_strongest,:);
else
    points2 = points2_raw;
end

if (descriptor_type == 'CIRCLE')
    % Obtain circle patches as descriptors
    descriptor2 = factor*getDescriptor(points2(:,1:2), img2, descriptor_diameter/factor);
elseif (descriptor_type == 'SIFT')
    % Obtain SIFT descriptor
    descriptor2 = [];
    for i=1:size(points2,1)
        img2_    = vl_imsmooth(im2double(img2), sqrt(points2(i,3)^2 - 0.5^2));
        [Ix, Iy] = vl_grad(img1_) ;
        mod      = sqrt(Ix.^2 + Iy.^2) ;
        ang      = atan2(Iy,Ix) ;
        grd      = shiftdim(cat(3,mod,ang),2) ;
        grd      = single(grd) ;
        descriptor2(i,:) = vl_siftdescriptor(grd, ...
        [points2(i,2),points2(i,1),points2(i,3:4)]') ;
    end
end
```

```
100
101 %% Visualise Detected Interest Points
102 if visualise
103     f1=figure;
104     subplot(1,2,1);
105     myvisualise(points1,img1,'Img1 – interest points');
106     subplot(1,2,2);
107     myvisualise(points2,img2,'Img2 – interest points')
108     set(f1, 'Units', 'normalized', 'Position', [0,0,1,1]);
109 %     print(f1,'-dpng','-r600');
110 end
111
112 %% Matching correspondance
113 [matchedPoints1, matchedPoints2] = myFeatureMatching(points1(:,1:2),...
114                 points2(:,1:2), descriptor1, descriptor2, knn_threshold);
115
116 %% Visualise matched correspondences
117 if visualise
118     figure; ax = axes;
119     showMatchedFeatures(img1,img2,fliplr(matchedPoints1),fliplr(matchedPoints2),'montage','Parent',ax)
        ;
120     title(ax, 'Matched Correspondances before RANSAC');
121     legend(ax, 'Matched points 1','Matched points 2');
122 end
123
124 %% Compute Homography Matrix using RANSAC
125 numberoftrial = 10000;
126 threshold = 20; % average cityblock distance in pixels
127
128 [besth, f2, inliersIndex] = RansacPredictH(matchedPoints1, matchedPoints2, numberoftrial, threshold);
129 matchedPoints1_inliers_ground_truth = [];
130 matchedPoints2_inliers_ground_truth = [];
131 cnt = 1;
132 for i=1:size(matchedPoints1,1)
133     if (inliersIndex(i) == 1)
134         matchedPoints1_inliers_ground_truth(cnt,:) = matchedPoints1(i,:);
135         matchedPoints2_inliers_ground_truth(cnt,:) = matchedPoints2(i,:);
136         cnt = cnt +1;
137     end
138 end
139 projectedv2 = hmatrixproject(matchedPoints1_inliers_ground_truth, besth);
140
141 %% visualise projected points after RANSAC
142 if visualise
143     figure; ax = axes;
144     showMatchedFeatures(img1,img2,fliplr(matchedPoints1_inliers_ground_truth),fliplr(
        matchedPoints2_inliers_ground_truth),'montage','Parent',ax);
145     title(ax, 'Inliers after RANSAC');
146     legend(ax, 'Matched points 1','Matched points 2');
147 end
148
149 %% Compute Fundamental Matrix using RANSAC
150 numOfTrials = 10000;
151 thr = 0.01;
152 [bestf, f, inliers] = RansacPredictF(matchedPoints1_inliers_ground_truth,...
153                       matchedPoints2_inliers_ground_truth,numOfTrials,thr);
154 matchedPoints1_inliers = matchedPoints1_inliers_ground_truth.*[inliers',inliers'];
155 matchedPoints2_inliers = matchedPoints2_inliers_ground_truth.*[inliers',inliers'];
156 % f1 = estimateFundamentalMatrix(v1, v2,'Method','RANSAC','NumTrials',500);
157 figure;
158 drawEpipolarline(bestf, matchedPoints1_inliers, img2);
159
160 %% visualise projected points after RANSAC
161 if visualise
```

```
    figure; ax = axes;
    showMatchedFeatures(img1,img2,fliplr(matchedPoints1_inliers),fliplr(matchedPoints2_inliers),'
        montage','Parent',ax);
    title(ax, 'Inliers after Fundamental Matrix RANSAC');
    legend(ax, 'Matched points 1','Matched points 2');
end
```

Listing 14. Q2.2.a & b (SIFT)

```
%% Q2_2_a SIFT
% Estimate fundamental matrix using list of correspondences from Q1.1 or
% Q1.2.a.

%% parameters to be adjusted
knn_threshold = 1.4;
num_strongest = 500; % number of strongest interest points
visualise = 1;

%% read image
img1 = imread('pictures/FD7.JPG');
% img1 = imread('pictures/scene1.ppm');
img1 = imresize(img1,[680 850]);
img2 = imread('pictures/FD8.JPG');
% img2 = imread('pictures/scene5.ppm');
img2 = imresize(img2,[680 850]);

% initialisation
addpath('external/vlfeat-0.9.18');
addpath('external/libsvm-3.18/matlab');
run('external/vlfeat-0.9.18/toolbox/vl_setup.m'); % vlfeat library
cd('external/libsvm-3.18/matlab'); % libsvm library
run('make');
cd('../../..');

% ================= convert colour scale to gray scale =================
% img 1
if (size(size(img1),2) == 3)
    img1 = rgb2gray(img1);
end

% img 2
if (size(size(img2),2) == 3)
    img2 = rgb2gray(img2);
end


%% extract interest points & their descriptors
[F1_raw,D1_raw]=vl_sift(single(img1));
[F2_raw,D2_raw]=vl_sift(single(img2));
coords1_sift_raw = horzcat(F1_raw(2,:)',F1_raw(1,:)');
coords2_sift_raw = horzcat(F2_raw(2,:)',F2_raw(1,:)');

%% sort by confidence in descending order
[~,idx1] = sort(F1_raw(4,:),'descend');
[~,idx2] = sort(F2_raw(4,:),'descend');

%% get the strongest interest points and their descriptors
if size(F1_raw,2) > num_strongest
    F1 = F1_raw(:,idx1(1:num_strongest));
    F2 = F2_raw(:,idx2(1:num_strongest));
    D1 = D1_raw(:,idx1(1:num_strongest));
    D2 = D2_raw(:,idx2(1:num_strongest));
    coords1_sift = coords1_sift_raw(idx1(1:num_strongest),:);
    coords2_sift = coords2_sift_raw(idx2(1:num_strongest),:);
```

```
    points1_sift = horzcat(coords1_sift,F1(3:4,:)');
    points2_sift = horzcat(coords2_sift,F2(3:4,:)');
else
   F1 = F1_raw;
   F2 = F2_raw;
   D1 = D1_raw;
   D2 = D2_raw;
   coords1_sift = coords1_sift_raw;
   coords2_sift = coords2_sift_raw;
   points1_sift = horzcat(coords1_sift,F1(3:4,:)');
   points2_sift = horzcat(coords2_sift,F2(3:4,:)');
end


%% visualise the strongest N points in both images
if visualise
    figure;
    subplot(2,2,1);
    myvisualise(points1_sift, img1, 'very original corner image 1');
    subplot(2,2,2);
    myvisualise(points2_sift, img2, 'very original corner image 2');
end

%% Matching correspondence (our own implementation)
[matchedPoints1_sift, matchedPoints2_sift] = myFeatureMatching(...
            coords1_sift, coords2_sift, D1', D2', knn_threshold);

%% Visualise matched correspondences before RANSAC
if visualise
    figure; ax = axes;
    showMatchedFeatures(img1,img2,fliplr(matchedPoints1_sift),fliplr(matchedPoints2_sift),'montage','
        Parent',ax);
    h=title(ax, 'Matched Correspondences before RANSAC');
    set(h,'FontSize',16);
    legend(ax, 'Matched points 1','Matched points 2');
end


%% Compute Fundamental Matrix using RANSAC
numOfTrials = 100000;
thr = 0.01;
[bestf1, f, inliers1] = RansacPredictF(matchedPoints1_sift,...
                      matchedPoints2_sift,numOfTrials,thr);

[bestf2, f, inliers2] = RansacPredictF(matchedPoints2_sift,...
                      matchedPoints1_sift,numOfTrials,thr);

matchedPoints1_inliers = [];
matchedPoints2_inliers = [];
cnt = 1;
for i=1:size(matchedPoints1_sift,1)
    if (inliers1(i) == 1)
        matchedPoints1_inliers(cnt,:) = matchedPoints1_sift(i,:);
        matchedPoints2_inliers(cnt,:) = matchedPoints2_sift(i,:);
        cnt = cnt +1;
    end
end
if visualise
    f=figure;
    drawEpipolarline(bestf1, matchedPoints1_sift, img2);
    h=title('Img1 – Epipolar Lines and Epipoles');
    set(h,'FontSize',30);
end

```

```
% Calculate epipole
[isIn,epipole1] = isEpipoleInImage(bestf1,size(img1))
[isIn,epipole2] = isEpipoleInImage(bestf2,size(img2))
```

Listing 15. Q2.2.f & c & d

```
% Q_2_2_f_c_d
% f) rectify images
% c) Calculate disparity map between Image A and B
% d) Plot 3D depth map

%% Load the Parameters of the Stereo Camera
% Note that the parameters used here are pre-calculated using Matlab
% Stereo Camera Calibration APP
load('stereoParams.mat');

%% Read and Rectify Frames
img1 = imread('pictures/FD8.JPG');
img2 = imread('pictures/FD7.JPG');

img2 = imresize(img2,[680 850]);
img1 = imresize(img1,[680 850]);

figure;
subplot(121);
imshow(img1);
subplot(122);
imshow(img2);

%% Rectification (f)
[img1Rect, img2Rect] = ...
    rectifyStereoImages(img1, img2, stereoParams);

f = figure;
subplot(121);
imshow(stereoAnaglyph(img1Rect, img2Rect));
title('Rectified Left and Right Images','FontSize',16);

%% Compute Disparity (c)
% In rectified stereo images any pair of corresponding points are located
% on the same pixel row. For each pixel in the left image compute the
% distance to the corresponding pixel in the right image. This distance is
% called the disparity, and it is proportional to the distance of the
% corresponding world point from the camera.
img1Gray  = rgb2gray(img1Rect);
img2Gray = rgb2gray(img2Rect);

disparity_range = [0 64];
disparityMap = disparity(img1Gray, img2Gray,'DisparityRange',disparity_range);

subplot(122);
imshow(disparityMap, disparity_range);
title('Disparity Map (f=29mm)','FontSize',16);
colormap jet
colorbar

%% Reconstruct the 3-D Scene (d)
% Reconstruct the 3-D world coordinates of points corresponding to each
% pixel from the disparity map.
points3D = reconstructScene(disparityMap, stereoParams);

% Convert to meters and create a pointCloud object
points3D = points3D ./ 1000;
ptCloud = pointCloud(points3D, 'Color', img1Rect);
```

```
59
60 % Create a streaming point cloud viewer
61 player3D = pcplayer([-3, 3], [-3, 3], [0, 8], 'VerticalAxis', 'y', ...
62     'VerticalAxisDir', 'down');
63 xlabel(player3D.Axes, 'Horizontal Direction (m)','FontSize',14);
64 ylabel(player3D.Axes, 'Vertical Direction (m)','FontSize',14);
65 zlabel(player3D.Axes, 'Depth (m)','FontSize',14);
66 title(player3D.Axes, 'Depth Map (f=29mm)','FontSize',16);
67
68 % Visualize the point cloud
69 view(player3D, ptCloud);
```

Listing 16. Q2.2.e

```
1  % Q_2_2_e
2  % e) Change the focal length by 2mm, repeat Q2.2.d and compare.
3  % Add small random noise (e.g. Gaussian with max 2 pixel) to the disparity
4  % map, repeat Q2.2.d and compare.
5
6  % Field of view, Depth of field and Perspective
7
8  %% Load the Parameters of the Stereo Camera
9  % Note that the parameters used here are pre-calculated using Matlab
10 % Stereo Camera Calibration APP
11 load('stereoParams.mat');
12
13 %% Read and Rectify Frames
14 img1 = imread('pictures/FD8.JPG');
15 % img1 = imread('pictures/scene1.ppm');
16 img2 = imread('pictures/FD7.JPG');
17 % img2 = imread('pictures/scene2.ppm');
18
19
20 % zoom the images by 2mm focal length (from 29mm to 31mm)
21 img_height = size(img1,1);
22 img_length = size(img1,2);
23 delta_h = img_height*2/31;
24 delta_l = img_length*2/31;
25
26 img1 = img1(1+delta_h:end-delta_h,1+delta_l:end-delta_l,:);
27 img2 = img2(1+delta_h:end-delta_h,1+delta_l:end-delta_l,:);
28 img1 = imresize(img1,[680 850]);
29 img2 = imresize(img2,[680 850]);
30
31
32 [img1, img2] = ...
33     rectifyStereoImages(img1, img2, stereoParams);
34
35 figure;
36 imshow(stereoAnaglyph(img1, img2));
37 title('Rectified Image','FontSize',16);
38
39 %% Compute Disparity
40 % In rectified stereo images any pair of corresponding points are located
41 % on the same pixel row. For each pixel in the left image compute the
42 % distance to the corresponding pixel in the right image. This distance is
43 % called the disparity, and it is proportional to the distance of the
44 % corresponding world point from the camera.
45 frameLeftGray  = rgb2gray(img1);
46 frameRightGray = rgb2gray(img2);
47 disparity_range = [0 64];
48 disparityMap = disparity(frameLeftGray, frameRightGray,...
49                     'DisparityRange',disparity_range);
50
```

```
figure;
imshow(disparityMap,disparity_range);
title('Disparity Map (without noise, f=31mm)','FontSize',16);
colormap jet
colorbar

%% Reconstruct the 3-D Scene (depth map)
% Reconstruct the 3-D world coordinates of points corresponding to each
% pixel from the disparity map.
points3D = reconstructScene(disparityMap, stereoParams);

% Convert to meters and create a pointCloud object
points3D = points3D ./ 1000;
ptCloud = pointCloud(points3D, 'Color', img1);

% Create a streaming point cloud viewer
player3D = pcplayer([-3, 3], [-3, 3], [0, 8], 'VerticalAxis', 'y', ...
    'VerticalAxisDir', 'down');
xlabel(player3D.Axes, 'Horizontal Direction (m)','FontSize',14);
ylabel(player3D.Axes, 'Vertical Direction (m)','FontSize',14);
zlabel(player3D.Axes, 'Depth (m)','FontSize',14);
title(player3D.Axes, 'Depth Map (without Noise, f=31mm)','FontSize',16);

% Visualize the point cloud
view(player3D, ptCloud);
% title('Depth Map (Larger Focal Length)');

%% Add small random noise to disparity map and repeat plotting depth map
disparityMap_noisy = disparityMap + (rand(size(disparityMap,1),size(disparityMap,2))-0.5).*4;
figure;
imshow(disparityMap_noisy, disparity_range);
title('Disparity Map (with noise, f=31mm)','FontSize',16);
colormap jet
colorbar

%% depth map
points3D_noisy = reconstructScene(disparityMap_noisy, stereoParams);

% Convert to meters and create a pointCloud object
points3D_noisy = points3D_noisy ./ 1000;
ptCloud_noisy = pointCloud(points3D_noisy, 'Color', img1Rect);

% Create a streaming point cloud viewer
player3D_noisy = pcplayer([-3, 3], [-3, 3], [0, 8], 'VerticalAxis', 'y', ...
    'VerticalAxisDir', 'down');
xlabel(player3D_noisy.Axes, 'Horizontal Direction (m)','FontSize',14);
ylabel(player3D_noisy.Axes, 'Vertical Direction (m)','FontSize',14);
zlabel(player3D_noisy.Axes, 'Depth (m)','FontSize',14);
title(player3D_noisy.Axes, 'Depth Map (with Noise, f=31mm)','FontSize',16);

% Visualize the point cloud
view(player3D_noisy, ptCloud_noisy);
```