Coursework 2: Image Matching

Laura Palacio Garcia (CID: 01322823)
Imperial College London
lp2816@ic.ac.uk

Jenna Luchak (CID: 01429938)
Imperial College London
jkl17@ic.ac.uk

1. Matching

1.1. Automatic

Our method used algorithms to detect interest points, create descriptors and find correspondences. Interest point detection implemented the Harris' corner detection algorithm, based on the findings of [4]. Our parameters for corner detection included $\sigma_1 = 5$, used in the Gaussian filter function and $threshold = 1,500,000$, used in non maximal suppression. The $threshold$ was chosen based on proper image coverage. We tested $threshold$ values ranging from 10,000 to 3,000,000. When $threshold$ was too low, the algorithm's non maximal suppression was too weak; recognized pixels as corners, when they were not. When $threshold$ was too large, not enough points were detected.

The descriptors were created by implementing a simple color histogram on a sample of pixels. The parameter for this algorithm was $radius = 5$, where every descriptor was comprised of 11 x 11 pixels, with the feature point in the center. The histograms of a couple patches can be seen in Figure 1.

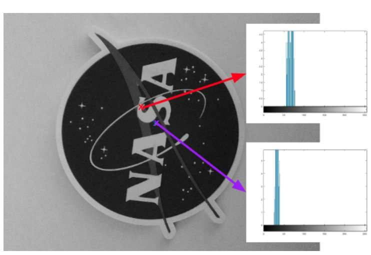

Figure 1. Histograms of two descriptors.

The matching algorithm implemented a nearest neighbours algorithm. Parameters $\sigma_1 = 5$ and $radius = 5$ were chosen. Various values of said parameters were tested, but these achieved the lowest average distance across all matched points. Additionally, a filtering parameter was used on the matched points so that only pairings whose distance had a value less than 11 (the length of each patch vector) were stored.

Our model's drawback is the simple colour histograms used to make descriptors. Our method uses gray scale, which may reduce matching accuracy because multiple colours will be mapped as similar shades of gray. However, a benefit of using gray scale is the reduction in stored memory (it does not store 3 intensity matrices per image like RGB) and computation time.

1.2. Transformation estimation

The homography matrix H maps a point in one image to a point in another image taken at the same camera position whereas the fundamental matrix F is used when a scene is viewed from two cameras and there is a point-to-line correspondence. In order to estimate both matrices, we use singular value decomposition (SVD) [8] on matrix A, built by solving the system of equations $x' = Hx$ for the homography matrix and $x'Fx = 0$ for the fundamental matrix [6]. At least 4 sets of matching points ($[x', y']$ in image A and $[x, y]$ in B, selected using *getpts*) are needed to create a $8x9$ matrix A and determine the 8 unknowns in H, whereas 8 points are needed to find F (eight-point algorithm [7]).

Then, we project some new points from image B to image A using $x' = Hx$. The homography accuracy (HA) is the average Euclidean distance between the true points in image A and the projected ones. After repeating the analysis 10 times, the average HA for our HG images was 9.82 pixels and $HA = 9.83$ for the Boat sequence [14].

Since all epipolar lines intersect at the epipole (last eigenvector after applying SVD on F [6]), we calculate the epipolar line of a point (Figure S1) knowing the point and epipole in that image [6]. The fundamental accuracy (FA) is the average point-to-line distance [15] between the new points in image B and their corresponding epipolar lines in image A (Figure S2), obtained using F. The average FA obtained with our images FD was 77.96 pixels and 32.39 with RANSAC, whereas $FA = 22.76$ with the Tsukuba sequence [12].

In conclusion, HA is quite small and the projected points almost match perfectly the original ones in our images (Figure S3). However, FA is bigger although the accuracy improves drastically with RANSAC.

2. Image Geometry

2.1. Homography (using images HG)

2.1.1 Investigate Interest Points

The interest points of an image undergoing three transformations: (i) No transformation, (ii) scaled 1/2 and (iii) scaled 1/3, were automatically detected. We used homography accuracy (HA) to compare their interest points (Figure S4). As the size of the image is reduced by 1/2 and 1/3, the number of features detected decreased proportionally. Case (i) detected the most points (14965), over 2 and 3 times as many detected in the scaled images, 5597 and 4259, respectively. This is expected. When an image is scaled, the pixel resolution of the image reduces. The less pixels available, the less pixels to detect.

The homography matrix was computed by selecting 4 interest points randomly. The matrix was used in conjunction with all of the interest points detected from each image to determine HA. The HA between the original image and the image scaled by 1/2 was 1.4144, while the HA between the original image and the image scaled by 1/3 was 1.4147.

This was an unexpected result. Reducing the image quality should reduce the precision of corner detection greatly; we assumed that scaling an image by 1/3 would produce a less accurate solution. A possible reason for the similar results may be that, since there are an abundance of feature points across all transformations, the HA was unaffected by the scaling transformation.

2.1.2 Investigating Correspondences

Two image transformations are investigated (Figure 3), a zoom between image 1 and image 2 (T_Z) and a rotation between image 1 and image 3 (T_R). Homography may be estimated with as few as 4 correspondences, up to and including the entire list (153 and 232 correspondences for T_Z and T_R respectively). In Figure 2, we will investigate the optimal number to use by comparing average HA over 100 trials to account for the randomness.

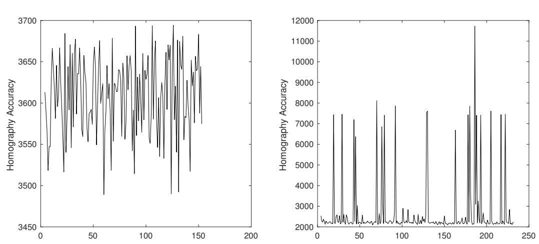

Figure 2. Homography accuracy (Y axis) with respect to number of correspondences (X axis) [Left] T_Z, and [Right] T_R.

The HA results range from 3,500 to 3,700 for T_Z and 2,000 to 12,000 for T_R. Both plots oscillate with no correlation in the data. Interestingly, T_R also spontaneously spikes upward; therefore, it can be inferred that our model is less robust for rotational transformations than it is for scaling transformations.

Unexpectedly, the trend in HA values is unrelated to the number of correspondences. We assumed that when a larger number of points is used to estimate homography, the system becomes over determined and not as accurate. Large error in our model may be either from the lack of complexity in our method to make descriptors, or our choice in image is not diverse enough.

We will now compare manual and automatic correspondences through analyzing their geometric transformations. Parameters will be derived from the elements of each homography matrix [1]. See results in Table 1.

Type	Zoomed in, T_Z			Rotation, T_R			
	scale	x	y	scale	θ	x	y
Auto	0	0	0	0	0	0	0
Manual	4.65	323	-311	5.65	26.7	-4081	-3363

Table 1. Geometric Transformation Parameters.

No parameters were estimated from automatic matrices, as their elements were approximately zero. From manual, pure scaling was estimated for T_Z, while rotation and scaling were estimated for T_R, since the rotational elements >1. Both transformations estimated a translation in x and y directions.

See Figure 3; T_Z parameters are better estimated than T_R, as the iPhone's maximum zoom capability is 5x. Our model is not accurate for all transformation types; it varies between paired points. The errors observed in Figure 3: green, purple and yellow are expected. The green pairing was mismatched with a similarly shaped letter edge, and yellow and purple mismatched with similarly shaped stars. The red pairing is an unexpected error (outlier); the matched element is spatially far and structurally dissimilar.

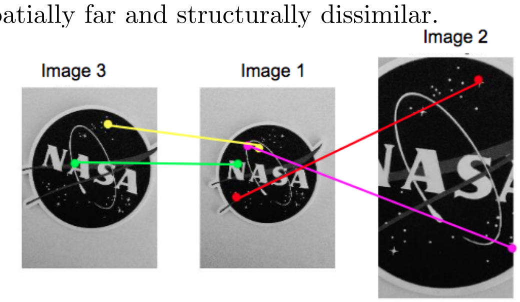

Figure 3. Automatically selected correspondences. [Center] Original image.[Left] T_R, and [Right] T_Z.

Our automatic model was inadequate at matching points and estimating transformation parameters. In future considerations, we would remove outliers from the list of correspondences by implementing a filter after nearest neighbours calculations.

2.2. Stereo Vision (using images FD)

The parameters used are focal length f (26mm on a Samsung Galaxy S7 camera) and the baseline b (image B is taken moving the camera 20cm to the right).

2.2.1 Fundamental

The minimum number of points to build a fundamental matrix is 8. The total number of correspondences between the images shown in Figure 4 was 126. After calculating the fundamental accuracy (FA) using a varying number of points between 8 and 126, we found that FA was lowest at 8 points (7.8), after which FA oscillated sporadically, reaching FA's as high as 700. FA is optimal at 8 points.

In Figure 4, the epipolar lines converge on the epipoles, located outside the limits of the images on the right. Although the images were taken only translating the camera, they are not perfectly stereo rectified. That might be the reason why the epipoles are not at infinity and the epipolar lines are not parallel along the horizontal axis, as they should be if they were rectified.

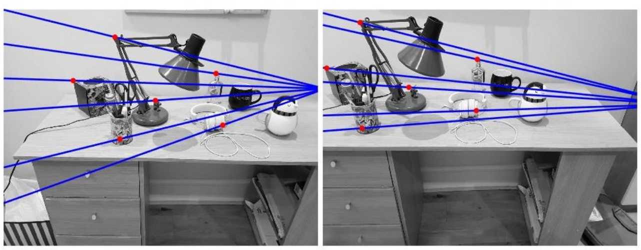

Figure 4. Epipolar lines (blue) containing epipoles and interest points (red) in [Left] image A, and [Right] image B.

2.2.2 Disparity map and depth map

Usually, first, images must be stereo rectified so that epipolar lines are parallel and disparities can be compared simply horizontally [9]. However, we assume our images are almost stereo rectified.

The disparity map is obtained using Sum of Squared Differences (SSD) [5], where we choose a disparity range and window size to compare intensities. Since image B was taken 20cm to the right of image A, the disparity range chosen are the positive values along the horizontal line (given $d = x_l - x_r$ [5]). In Figure 5, we see that larger windows produce less accurate and less detailed maps but are more robust to noise (Figure S5).

Depth is inversely proportional to the disparity ($z = f * b/d$) so the disparity will be larger when the object is closer. That is why the penguin on the right has large disparity (Figure 5) and small depth (Figure 6). Increasing the focal length, increases the depth (making the depth map brighter) and viceversa. However, an increase of only 2mm has a small effect (Figure 6) but 20mm really brightens the depth map (Figure S6).

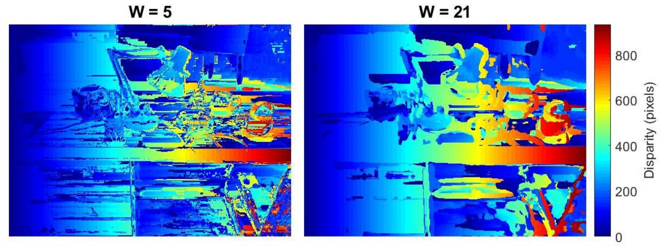

Figure 5. Disparity map between both images FD for different window sizes used to compare intensities.

Adding Gaussian noise of mean 1 and standard deviation 1 (a maximum of 2 pixels) to the disparity map has a blurring effect and little black noisy dots appear (Figure 6). When the standard deviation is increased to 10, the noisy effect is enhanced (Figure S6).

Given that our images are not completely stereo rectified, the results for the disparity map and depth map are not the best, as opposed to using the Tsukuba sequence (Figure S7).

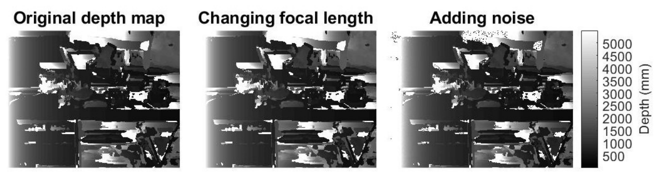

Figure 6. Depth maps of our images FD from the disparity map with $W = 21$: [Left] $f = 26mm$, [Center] $f = 28mm$, [Right] Gaussian noise of mean 1 and standard deviation 1.

2.2.3 Stereo image rectification

The rectification algorithm used comes from different sources [13], [3], [11], [2]. The two parameters known are the rotation R and translation T (or baseline b) applied to the camera when taking the new image B. R is a $3x3$ identity matrix since image B is taken translating the camera without rotation. Three orthogonal unit vectors (from the epipole) build a rectification matrix R_{rect} [2], which will create a left and right rotation matrix to get the left and right rectified images, respectively.

Stereo rectification ensures that the correspondence of a pixel in the left image is found by searching only the pixels along the horizontal epipolar line in the right image [3]. The obtained stereo rectified images (Figure 7) are not perfectly horizontally aligned but, not even a Matlab code example [10] worked on our images (but it did in the Tsukuba sequence in Figure S8).

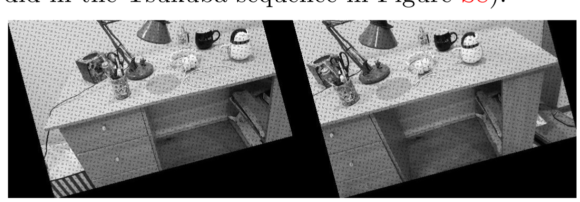

Figure 7. Stereo rectified pair of our images FD.

References

[1] Alan Zucconi. The Transformation Matrix. [Online]. Available: https://www.alanzucconi.com/2016/02/10/tranfsormation-matrix/, 2016. [Accessed April 11, 2018].

[2] Guido Gerig. Lecture on Image Rectification (Stereo) from "3D Computer Vision" (CS 6320). [Online]. Available: http://www.sci.utah.edu/~gerig/CS6320-S2012/Materials/CS6320-CV-F2012-Rectification.pdf, Spring 2012. [Accessed April 15, 2018].

[3] Hai Tao (Department of Computer Engineering in University of California at Santa Cruz). Lecture on Rectification and Depth Computation from "Image Analysis and Computer Vision" (CMPE 264). [Online]. Available: https://classes.soe.ucsc.edu/cmpe264/Fall06/Lec12.pdf, Fall 2006. [Accessed April 15, 2018].

[4] Kavita Bala. CS4670: Computer Vision. [Online]. Available: http://www.cs.cornell.edu/courses/cs4670/2015sp/lectures/lec07_harris_web.pdf, 2015. [Accessed April 11, 2018].

[5] Krystian Mikolajczyk (Imperial College London). Lecture on Geometry from "Machine Learning for Computer Vision" (EE4-62), February 2018.

[6] Krystian Mikolajczyk (Imperial College London). Lecture on Matching from "Machine Learning for Computer Vision" (EE4-62), February 2018.

[7] Marc Pollefeys (University of North Carolina at Chapel Hill). Eight-point algorithm. [Online]. Available: http://www.cs.unc.edu/~marc/tutorial/node54.html, 2002. [Accessed April 10, 2018].

[8] Massachusetts Institute of Technology. Singular Value Decomposition (SVD) tutorial. [Online]. Available: http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm, 2017. [Accessed April 11, 2018].

[9] MathWorks. Disparity function in Matlab. [Online]. Available: https://uk.mathworks.com/help/vision/ref/disparity.html, 2018. [Accessed April 21, 2018].

[10] MathWorks. Uncalibrated Stereo Image Rectification in Matlab. [Online]. Available: https://uk.mathworks.com/help/vision/examples/uncalibrated-stereo-image-rectification.html, 2018. [Accessed April 25, 2018].

[11] National Instruments. Stereo Image Rectification In-Depth. [Online]. Available: http://zone.ni.com/reference/en-XX/help/372916T-01/nivisionconcepts/stereo_image_rectification_in-depth/, 2018. [Accessed April 15, 2018].

[12] The Middlebury Computer Vision Pages. Stereo datasets with ground truth. [Online]. Available: http://vision.middlebury.edu/stereo/data/scenes2001/data/tsukuba/, 2001. [Accessed March 29, 2018].

[13] University of Maryland Institute for Advanced Computer Studies. Lecture on Stereo Imaging from "Algorithms and systems for capture and playback of spatial audio" (CMSC828D). [Online]. Available: http://legacydirs.umiacs.umd.edu/~ramani/cmsc828d/lecture14_6pp.pdf, October 2000. [Accessed April 15, 2018].

[14] University of Oxford. Affine Covariant Features: Boat sequence. [Online]. Available: http://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/boat.tar.gz, 2007. [Accessed March 29, 2018].

[15] Wikipedia. Distance from a point to a line. [Online]. Available: https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line, 2018. [Accessed April 7, 2018].

Appendix

Supplementary figures

Figure S1. Epipolar line (in blue) in image A containing the epipole (located outside the image limits) and the interest point selected (in red).

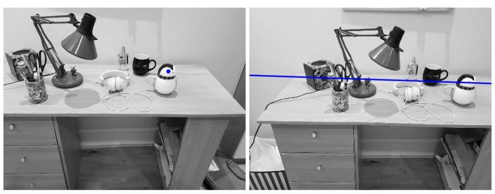

Figure S2. [Left] Selected point in image B, and [Right] its corresponding epipolar line in image A.

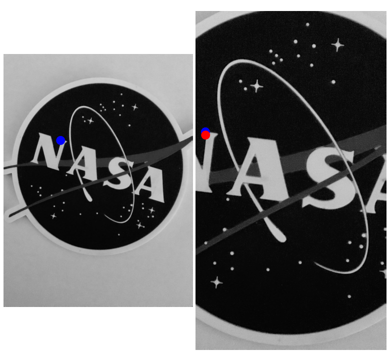

Figure S3. [Left] Selected point in image B, and [Right] its corresponding projected point (in red) and original true point (in blue) in image A.

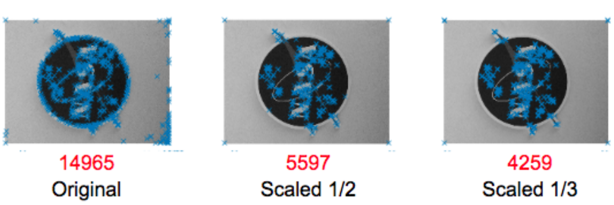

Figure S4. Number of automatically detected interest points (in red), when scaled by a given factor.

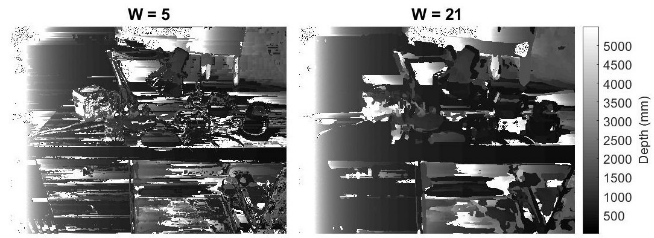

Figure S5. Depth maps of our images FD from the disparity map with different window sizes, with added Gaussian noise of mean 1 and standard deviation 1.

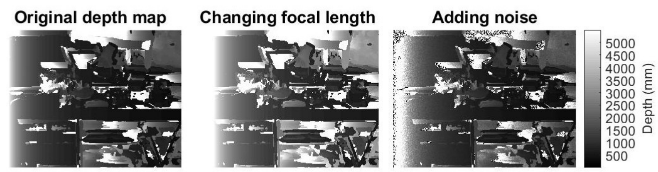

Figure S6. Depth maps of our images FD from the disparity map with $W = 21$: [Left] $f = 26mm$, [Center] $f = 46mm$, [Right] Gaussian noise of mean 1 and standard deviation 10.

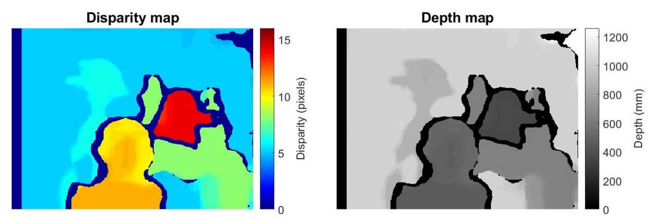

Figure S7. Disparity map and depth map of the Tsukuba sequence for $W = 21$.

Figure S8. Stereo rectified pair of the Tsukuba squence using a documented Matlab example [10].

Matlab code

Question 1: Matching

In this section, the implemented code is shown, which consists of a main script and some functions called from that script.

1.1 Manual

First, we load the images and call a function that obtains the coordinates of corresponding interest points in two images by clicking on them:

```
clear all
close all
clc
                %%%%%%%%%%%%%%%%%%%
                %% Q1: Matching %%
                %%%%%%%%%%%%%%%%%%%
%% 1) Manual
% Select at least 4 points for finding the homography matrix
folderName = 'sequences_images/myimages/FD';
imgList = dir(fullfile(folderName,'*.jpg'));
[x, y] = getPoints(folderName, imgList, 'both'); % select at least 4 points to find H
```

The function that gets the point coordinates is the following:

```

function [x, y] = getPoints(folderName, imgList,image)
% Obtain set of corresponding points by clicking on the same interest point
% in the different images.
x = [];
y = [];
if image == 'A'
    i = 1; % image index
elseif image == 'B'
    i = 2;
end

switch image
    case 'both'
        for i = 1:2 %size(imgList,1)
            I = rgb2gray(imread(fullfile(folderName,imgList(i).name)));
            figure(i);
            imshow(I);
            [X, Y] = getpts; % select points from image
            x(i,:) = round(X); % pixel value
            y(i,:) = round(Y);
            close(gcf) % close figure
        end
    otherwise
        I = rgb2gray(imread(fullfile(folderName,imgList(i).name)));
        figure;
        imshow(I);
        [X, Y] = getpts; % select points from image
        x = round(X'); % pixel value
        y = round(Y');
        close(gcf) % close figure
end
end
```

1.2 Automatic

Next, we implemented a series of functions to perform automatic matching correlation between at least 2 images. The code for this section was broken down into two parts. First, we use Harris corner detection and a simple color histogram to create descriptors, and then we perform nearest neighbour matching of those descriptors.

1.2(a) Create Descriptors

To create descriptors we initialize several functions to perform operations. Our main function, *CornerDetection*, prepares the image by converting it from RGB to gray scale as well as scales the image (if needed). The main function calls three sub functions, *HarrisCornerDetection*, *histogram* and *MakeDescriptors* to perform Harris corner detection, histograms and make the descriptors of the image, respectively. The code for *CornerDetection* is as follows.

```
function [featurePts, descriptors, intensity] = Corner_detection(folderName, factor, params)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function has the ability to scale an image based on size, detect feature points via Harris Corner detection and create descriptors via a simple color histogram algorithm.
%
% Outputs:
```

```
%       featurePts - The matrix coordinates of every interest point
%       descriptors - 11x11 pixel patches
%       intensity - matrix  containing the intensity of every image pixel
% Inputs:
%       folderName - location of images stored as 'JPG' files
%       factor - image size scaling factor
%       params - A 6 x 1 struc that contains input parameters
% Parameters:
%       params.threshold - defines corner response threshold value (500000)
%       params.sigma1 - The standard deviation of Gaussian filter
%       params.plotCorners - logical operator plots feature points
%       params.radius - square size of the patch window
%       params.plotHistogram - logical operator plots patch histograms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Redefine input data from params structure
threshold = params.threshold;
sigma1 = params.sigma1;
plotCorners = params.plotCorners;
radius = params.radius;
plotHistogram = params.plotHistogram;

%% Initialize for loop to read data and perform operations
imgList = dir(fullfile(folderName,'*.JPG')); % Locate list of images
stop_descr = 0; % Initialize timer
stop_corner = 0; % Initialize timer

intensity = {zeros(size(imgList,1),1)}; % Preallocate size
featurePts = {zeros(size(imgList,1),1)}; % Preallocate size
descriptors = {zeros(size(imgList,1),1)}; % Preallocate size

for i = 1:size(imgList,1)
    I = imread(fullfile(folderName,imgList(i).name)); % Read image
%% Prepare image to grayscale
    I = rgb2gray(I); % Convert to grayscale to make computation faster
    intensity{i} = I; % Store intensities of image

%% Scale the Image
     I = imresize(I,factor);

%% Harris Corner Detection
    tic; % Initiate the timer

    % Call the Harris corner detection function
    [featurePts{i},R{i}] = HarrisCornerDetection(I, threshold, sigma1, plotCorners);

    stop_corner = stop_corner + toc; % Update the timer

%% Create Descriptors
    tic; % Initiate the timer

    % Call the make descriptors function
    descriptors{i} = MakeDescriptors(radius, I, featurePts{i}, plotHistogram);

    stop_descr = stop_descr + toc; % Update the timer
end
fprintf('Corner Detection elapsed time: %g\n', stop_corner); % Print elapsed time
fprintf('Descriptor elapsed time: %g\n', stop_descr); % Print elapsed time
end
```

The sub function that implements Harris corner detection is as follow.

```
function [featurePts] = HarrisCornerDetection(I, threshold, sigma1, plotCorners)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```matlab
% This function performs Harris Corner Detection on an image.
% Outputs:
%       featurePts - provides the x and y coordinates of the detected corners
% Inputs:
%       I - contains the intensity matrix for an image in grayscale
%       threshold - the corner response threshold value
%       sigma1 - the standard deviation for the gaussian filter
%       plotCorners - logical oeprator for plotting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 1. Image Derivatives
% i.e. Calculate derivatives in x and y direction for every pixel and compute Ix^2, Ixy, and Iy^2

    [dx,dy] = meshgrid(-1:1,-1:1); % Initialize x and y direction filters
    Ix = convn(I,dx,'same'); % derivative in x direction
    Iy = convn(I,dy,'same'); % derivative in y direction

    % Squared derivative matrices
    Ix2 = Ix.^2;
    Iy2 = Iy.^2;
    Ixy = Ix.*Iy;

%% 2. Window function - Gaussian filter
% i.e. Compute w(x,y)
    % Calculate gaussian function
    [x,y] = meshgrid(round(-sigma1/2):round(sigma1/2), round(-sigma1/2):round(sigma1/2));
    gauss = (2*pi*sigma1^2)^-1 * exp((x.^2+y.^2)/(-2*sigma1^2)); % gaussian function
    [a,b] = size(gauss); % dimensions of the resultant window function

    % Add all of the gaussian function terms together
    sum = 0; % Initialize counter
    for j = 1:a
        for k = 1:b
            sum = sum+gauss(j,k);
        end
    end
    % gauss normalization
    g = gauss./sum;

%% 3. Compute second moment matrix
% i.e. calculate the elements of matrix, M = w(x,y)*[Ix^2 Ixy;Iyx Iy^2])
    M11 = convn(Ix2,g,'same'); % g(Ix^2)
    M22 = convn(Iy2,g,'same'); % g(Iy^2)
    M12 = convn(Ixy,g,'same'); % g(Ixy)

%% 4. Measure of corner response
    k = 0.05; % empirical constant between 0.04 to 0.06
    R = (M11.*M22) - (M12.^2) - k*(M11+M22).^2;
    [a,b] = size(R);
    R_pad = padarray(R,[1,1]); % Outine matrix R with zeros

%% 5. Non maximum suppression
    N = 0; % center of matrix
    N = padarray(N,[1,1],1);
    neighbours = numel(N)-1;

    % Reconstruct matrix
    local_max = ordfilt2(R_pad, neighbours,N);
    local_max = local_max(1:a,1:b); % only use these elements

    % Detect feature points
    harris_points = (R == local_max) & (R > threshold);
    [rows,columns] = find(harris_points);
    x = columns;
    y = rows;
    featurePts = [rows,columns];

%% Plot Image and feature points
    if plotCorners == 1
        figure,
        imshow(I)
        hold on
        plot(x,y,'x');
    end
end
```

The function that implements descriptors is as follows.

```
function descriptors = MakeDescriptors(radius, I, featurePts, plotHistogram)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function makes descriptors out of feature points for an image
% Outputs:
%         descriptors - a list of descriptors stored for each image
% Inputs:
%         radius - defines the square size of the patch window
%         I - contains the intensity matrix for an image in grayscale
%         featurePts - the x and y coordinates of the detected corners
%         plot_histogram - Change to 1 to see histogram plots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 1. Create a window around every feature point
% i.e. Pad the image intensity matrix with zeros and then create a window around every feature point using all of the available pixels
    padded_I = padarray(I,[radius, radius],0); % Pad the I matrix with 'radius' loops of zeros
    num_featurePts = size(featurePts,1); % Number of feature points
    window = {zeros(1,num_featurePts)}; % Preallocate for speed

    % Collect pixels around every feature point
    for FP = 1:num_featurePts % loop through all feature points
        R_FP = featurePts(FP,1)+radius;
        C_FP = featurePts(FP,2)+radius;

        % Define window
        window_withZeros = padded_I(R_FP-radius:R_FP+radius,C_FP-radius:C_FP+radius);

        % Remove the padded zeros from each window
        rowsWithZeros = all(window_withZeros==0,1);
        colsWithZeros = all(window_withZeros==0,2);
        window{FP} = window_withZeros(~rowsWithZeros, ~colsWithZeros);
    end
%% 2. Create a Color histogram for every window
    numBins = 256; % Maximum number of colour intensities in an image
    frequencies = zeros(numBins,length(window)); % Preallocate for speed
    for w = 1:length(window) % Iterate through every window patch
        frequencies(:,w) = histogram(window{w},numBins,plotHistogram);
    end
    descriptors = frequencies; % A list of descriptors stored for each image
end
```

The sub function that implements creating histograms is as follows.

```
function frequencies = histogram(intensity,numBins,plotHistogram)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function determines the frequencies in every bin for a group of pixels
% Outputs:
%         frequencies - number of pixels in a given bin
% Inputs:
%         numBins - number of bins
%         intensity - intensity matrix for image in grayscale
%         plot_histogram - Change to 1 to see histogram plots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    frequencies = zeros(1,numBins); % pre allocate size
    for B=1:numBins
        frequencies(B) = size(intensity(intensity == B),1);
    end

    % Plot the histogram
    if plotHistogram == 1
    figure,
    x = 1:numBins;
    bar(x, frequencies, 'k');
```

```
    title('Grayscale Histogram');
    else
    end
end
```

1.2(b) Match Descriptors

To match descriptors, we created a function that uses the outputs from our make descriptors function as inputs. The matched pairs are determined through an algorithm that calculates the distances between one patches descriptors and all of the descriptors from at least one other image. Each patches match was the minimum distance found for every corresponding images descriptors. The function for this operation is as follows.

```
function [OriginalPoints,MatchedPoints] = FindingCorrespondences(featurePts, descriptors, MatchingImage)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function takes manually selected points from an image, locates the closest feature point relative to a list of automatically obtained feature points for an image and then uses a nearest neighbour algorithm to best match each feature point to another image that has been transformed (i.e. rotated and zoomed in).
%
% Outputs:
%       OriginalPoints - automatically selected interest points
%       MatchedPoints - estimated points on image 2 relative to image 1
%
% Inputs:
%       featurePts - The matrix coordinates of every interest point
%       descriptors - a list of descriptors stored for each image
%       MatchingImage - number of image in folder to match
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Nearest Neighbour Algorithm
tic; % Initiate timer

O = 1; % Original image
relative_descr = descriptors{O}; % Define the original images descriptors
count = 0; % initialize

for im = MatchingImage % Define the matching image
    matching_descr = descriptors{im}; % Define the current matching image descriptors

% Iterate through selected patch vectors
    for i = 1:1:size(relative_descr,2)
        % Calculate distances
        distances = vecnorm((relative_descr(:,i) - matching_descr),2);

        % Coordinates of the matching patches, corresponding to
        % the minimum distance
        [v,index] = min(distances);
        value(i) = v;
        if v <=11
            count = count + 1; % update counter
            match_x(count) = featurePts{im}(index,2); % columns = x
            match_y(count) = featurePts{im}(index,1); % rows = y
            original_x(count) = featurePts{O}(index,2);
            original_y(count) = featurePts{O}(index,1);
        end
    end
    %
    MatchedPoints = [match_x ; match_y]';
    OriginalPoints = [original_x ; original_y]';
end
stop_NN = toc; % Stop the timer
fprintf('Nearest neighbour elapsed time: %g\n', stop_NN);
end
```

1.3 Transformation estimation

The code used to estimate the homography matrix and the fundamental matrix is:

```
%% 3) Transformation estimation

%% Q1.3a) Homography matrix %%%
A = buildMatrixA(x,y,'homography'); % find matrix A
[U,S,V] = svd(A); % singular value decomposition
h = V(:,end)/V(end,end); % homography transfo'rmation
H = [h(1) h(2) h(3);
     h(4) h(5) h(6);
     h(7) h(8) h(9)];

%% Q1.3b) Fundamental matrix %%%
[x, y] = getPoints(folderName, imgList, 'both'); % select at least 8 points to find F
A = buildMatrixA(x,y,'fundamental');
[U,S,V] = svd(A); % singular value decomposition
f = V(:,end)/V(end,end);
F = [f(1) f(2) f(3);
     f(4) f(5) f(6);
     f(7) f(8) f(9)];

% Using Matlab functions...
matchedPoints1 = [];
matchedPoints2 = [];
for i = 1:size(x,2)
    % Points of image A and B
    matchedPoints1 = [matchedPoints1; x(1,i), y(1,i)];
    matchedPoints2 = [matchedPoints2; x(2,i), y(2,i)];
end
f8norm = estimateFundamentalMatrix(matchedPoints2,matchedPoints1,'Method','Norm8Point');
fRANSAC = estimateFundamentalMatrix(matchedPoints1,matchedPoints2, 'Method', 'RANSAC', 'NumTrials', 2000, 'DistanceThreshold', 1e-4);
[isIn,epipole] = isEpipoleInImage(fRANSAC, size(I_l));
```

This is the function that creates matrix A depending on the type of transformation to be performed:

```
function A = buildMatrixA(x,y,type)
% Create matrix A used to solve Ah = 0 and Af = 0
A = [];
for i = 1:size(x,2)
    % Points of image A and B
    points_a = [x(1,i); y(1,i)];
    points_b = [x(2,i); y(2,i)];

    switch type
        case 'homography'
            % Add all set of points in a matrix
            A = [A;
                 0 0 0 -points_b(1) -points_b(2) -1 points_a(2)*points_b(1) points_a(2)*points_b(2) points_a(2);
                 -points_b(1) -points_b(2) -1 0 0 0 points_a(1)*points_b(1) points_a(1)*points_b(2) points_a(1)];
        case 'fundamental'
            A = [A;
                 points_a(1)*points_b(1) points_a(1)*points_b(2) points_a(1) points_a(2)*points_b(1) points_a(2)*points_b(2) points_a(2) points_b(1) points_b(2) 1];
    end
end

end
```

Then, the homography accuracy is computed like so:

```
%% Q1.3c) Homography accuracy %%%
% Take new set of points (>= 1 pair) from the images
[x, y] = getPoints(folderName, imgList, 'both');
HA = computeHomographyAccuracy(x,y,H, folderName,imgList);
```

The function being called to find the homography accuracy is:

```
function HA = computeHomographyAccuracy(x,y,H,folderName,imgList)
% Find the accuracy by projecting the points and compare with real ones
for i = 1:size(x,2)
    points_A = [x(1,i); y(1,i)];
    points_B = [x(2,i); y(2,i)];

    points_A_hom_coord = H*[points_B;1]; % estimated points A
    points_A_proj = points_A_hom_coord/points_A_hom_coord(3); % obtain projected values
    points_A_proj = points_A_proj(1:2); % only keep x and y
    %xa = (H(1,1)*points_B(1) + H(1,2)*points_B(2) + H(1,3))/(H(3,1)*points_B(1) + H(3,2)*points_B(2) + 1);
    %ya = (H(2,1)*points_B(1) + H(2,2)*points_B(2) + H(2,3))/(H(3,1)*points_B(1) + H(3,2)*points_B(2) + 1);

    % Calculate the distance between correct and estimated points
    distance_points(i) = pdist([points_A_proj(1),points_A_proj(2); points_A(1),points_A(2)],'Euclidean');

    % Plot one image of B and the corresponding projected point in A
    if i == 1
        % Point in image B
        I = rgb2gray(imread(fullfile(folderName,imgList(2).name))); % image B
        figure;
        subplot(1,2,1)
        imshow(I);
        hold on
        plot(points_B(1), points_B(2), 'b.', 'MarkerSize', 20)
        %title('Selected point in image B','FontSize',15)
        %title('(a)', 'FontSize', 20)

        % Corresponding projected point and original point in image A
        I = rgb2gray(imread(fullfile(folderName,imgList(1).name))); % image A
        subplot(1,2,2)
        imshow(I);
        hold on
        plot(points_A(1), points_A(2), 'b.', 'MarkerSize', 20)
        plot(points_A_proj(1), points_A_proj(2), 'r.', 'MarkerSize', 20)
        %title('(b)', 'FontSize', 20)
        %title('Corresponding \color{red}projected \color{black} and \color{blue}original \color{black}point in image A','FontSize',15)
        pos = get(gca, 'Position');
        pos(1) = 0.47; % x
        set(gca, 'Position', pos)
    end
end
HA = mean(distance_points); % homography accuracy

end
```

In the main script, the epipoles, epipolar lines and fundamental accuracy are calculated:

```
%% Q1.3d) Fundamental matrix accuracy %%%

% Calculate coordinates of EPIPOLES (epipole can't be 0)
image = 'A';
epipole = getEpipoles(folderName, imgList, image, F);
% Check that F*[epipole_B;1] = 0 and F'*[epipole_A;1] = 0

% Calculate epipolar lines for the image
point1 = epipole;
point2 = [];
[point2(1), point2(2)] = getPoints(folderName, imgList, image); % select just one point from that image
ep_line_y = getEpipolarLine(point1, point2, folderName, imgList, image);

% Select points in image B and obtain
```

```
the EPIPOLAR LINES in A
[x, y] = getPoints(folderName, imgList, 'B');
FA = computeFundamentalAccuracy(x,y,F, folderName,imgList);
```

The function that obtains the coordinates of the epipole of one image is:

```

function epipole = getEpipoles(folderName, imgList, image, F)
% Epipole in image A is the intersection of all epipolar lines in image A
% Obtain:
    % Epipole in image B by solving F*e_b = 0 (since F is from B [x] to A [x'])
    % Epipole in image A by solving transpose(F)*e_a = 0
if image == 'A'
    i = 1; % image index
    [~, ~, V] = svd(F');
elseif image == 'B'
    i = 2;
    [~, ~, V] = svd(F);
end
last_eigen = V(:,end);
epipole = last_eigen/last_eigen(3); % normalize by dividing by z
epipole = ceil(epipole(1:2)); % 2D point pipole in image B

% Plot image and epipole
I = rgb2gray(imread(fullfile(folderName,imgList(i).name)));
figure;
imshow(I);
hold on
plot(epipole(1), epipole(2), 'b.', 'MarkerSize', 20)
%title(['Epipole in image ', image], 'FontSize', 20)

end
```

This is the function that obtains the epipolar line for a point in one image:

```
function ep_line_y = getEpipolarLine(point1, point2, folderName, imgList, image)
% Epipolar line in A contains interest point A and is obtained with
% interest point A and epipole A, viceversa for image B.

% Find line equation
syms x y
eqn = (y - point1(2) == (point2(2)-point1(2))/(point2(1)-point1(1))*(x - point1(1)));
v_y = solve(eqn, y);
ep_line_y = vpa(v_y, 5); % epipolar line equation for the image

% Plot image and epipolar line
if image == 'A'
    i = 1; % image index
elseif image == 'B'
    i = 2;
end
I = rgb2gray(imread(fullfile(folderName,imgList(i).name)));
figure;
imshow(I);
hold on
fplot(ep_line_y,[1,size(I,2)],'Color','blue','LineWidth',2)
plot(point1(1), point1(2), 'm.', 'MarkerSize', 20)
plot(point2(1), point2(2), 'r.', 'MarkerSize', 20)
%title(['\color{blue}Epipolar line \color{black}in image ', image, ' containing \color{magenta}epipole \color{black}and \color{red}interest point'], 'FontSize', 20)

end
```

Finally, the fundamental matrix accuracy is computed like so:

```
function FA = computeFundamentalAccuracy(X,Y,F, folderName,imgList)
% Find the accuracy by finding the epipolar lines in image A (from the points in B)
for i = 1:size(X,2)
    points_B = [X(i); Y(i)];
    epipolar_line_A = F*[points_B(1); points_B(2);1]; % line coefficients: ax + by + c = 0

    % Distance from a point to a line
    xo = points_B(1);
```

```
    yo = points_B(2);
    a = epipolar_line_A(1);
    b = epipolar_line_A(2);
    c = epipolar_line_A(3);
    distance_point_line(i) = abs(a*xo + b*yo + c)/sqrt(a^2 + b^2);

    % Plot one image of B and the corresponding epipolar lines in A
    if i == 1
        % Equation line
        syms x y
        eqn = (a*x + b*y + c == 0);
        v_y = solve(eqn, y);
        ep_line_y = vpa(v_y, 5); % epipolar line equation for image A

        % Point in image B
        I = rgb2gray(imread(fullfile(folderName,imgList(2).name))); % image B
        figure;
        subplot(1,2,1)
        imshow(I);
        hold on
        plot(xo, yo, 'b.', 'MarkerSize', 20)
        %title('(a)', 'FontSize', 20)
        %title('Selected point in image B','FontSize',15)

        % Corresponding epipolar line in image A
        I = rgb2gray(imread(fullfile(folderName,imgList(1).name))); % image A
        subplot(1,2,2)
        imshow(I);
        hold on
        fplot(ep_line_y,[1,size(I,2)],'Color','blue','LineWidth',2)
        %title('(b)', 'FontSize', 20)
        %title('Corresponding epipolar line in image A','FontSize',15)
        pos = get(gca, 'Position');
        pos(1) = 0.47; % x
        set(gca, 'Position', pos)
    end
end
FA = mean(distance_point_line); % fundamental matrix accuracy

end
```

Question 2: Image Geometry

The code for the first section "Homography (using image HG)" is not shown here since it uses all the code previously shown in this Appendix. However, for the second subquestion, we did create new code, which will be shown as follows.

2.2 Stereo Vision (using images FD)

2.2(a) Disparity and depth map

First, in order to plot the epipoles and epipolar lines in both images, we use:

```
%% Q2.2b) Epipoles and epipolar lines for both images %%%
plotMoreEpipolarLinesEpipoles(folderName, imgList, fRANSAC)
```

This function is as follows:

```
function plotMoreEpipolarLinesEpipoles(folderName, imgList, F)

epipole_A = getEpipoles(folderName, imgList, 'A', F);
epipole_B = getEpipoles(folderName, imgList, 'B', F);
I_l = rgb2gray(imread(fullfile(folderName,imgList(1).name)));
I_r = rgb2gray(imread(fullfile(folderName,imgList(2).name)));
close all

figure(1);
% Left image
subplot(1,2,1)
imshow(I_l);
hold on

% Calculate epipolar lines for the image A
point1 = epipole_A;
point2 = [];
[x, y] = getPoints(folderName, imgList, 'A');
for i = 1:length(x)
    point2 = [x(i);y(i)];
    ep_line_y = getEpipolarLine(point1, point2, folderName, imgList, 'A');
```

```
    figure(1)
    fplot(ep_line_y,[1,size(I_l,2)],'Color','blue','LineWidth',2)
    plot(point2(1), point2(2), 'r.', 'MarkerSize', 20)
end
plot(epipole_A(1), epipole_A(2), 'm.', 'MarkerSize', 20)
%title('(A) Left image', 'FontSize', 26)
%title('Epipolar lines and epipole in image A', 'FontSize', 20)

% Right image
subplot(1,2,2)
imshow(I_r);
hold on

% Calculate epipolar lines for the image B
point1 = epipole_B;
point2 = [];
[x, y] = getPoints(folderName, imgList, 'B');
for i = 1:length(x)
    point2 = [x(i);y(i)];
    ep_line_y = getEpipolarLine(point1, point2, folderName, imgList, 'B');
    figure(1)
    fplot(ep_line_y,[1,size(I_r,2)],'Color','blue','LineWidth',2)
    plot(point2(1), point2(2), 'r.', 'MarkerSize', 20)
end
plot(epipole_B(1), epipole_B(2), 'm.', 'MarkerSize', 20)
%title('(B) Right image', 'FontSize', 26)
pos = get(gca, 'Position');
pos(1) = 0.47; % x
set(gca, 'Position', pos)

end
```

Then, the disparity map is calculated like this:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Q2: Image Geometry %%
%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 2) Stereo Vision %%

I_l = rgb2gray(imread(fullfile(folderName,imgList(1).name))); % left image: A
I_r = rgb2gray(imread(fullfile(folderName,imgList(2).name))); % right image: B

% Define parameters
f = 26; % focal length of camera (typically 18-55mm)
b = 200; % baseline (distance between left and right camera: 20cm)

%% Q2.2b) Disparity map %%%
% The closer the object, the larger the disparity.
% Disparity: d = x_l - x_r;

% Window of pixels
W = [5 21]; % size window: only odd
originalSize = [];
a = [];
figure
for i = 1:length(W)
    a(i) = subplot(1,2,i);
    disparity_map = disparityMap(I_l, I_r, W(i));
%     norm_disparity_map = (disparity_map-min(range_d))/(max(range_d)-min(range_d));
%     imshow(norm_disparity_map,[0 1]);
    imshow(disparity_map, [0, size(I_l,2)-1]);
    colormap(gca,jet)
    originalSize(i,:) = get(gca, 'Position');
    title(['W = ', num2str(W(i))],'FontSize',20)
end
c = colorbar('FontSize',16);
c.Label.String = 'Disparity (pixels)';
c.Label.FontSize = 18;
set(a(1), 'Position', originalSize(1,:))
set(a(2), 'Position', originalSize(2,:))
pos = get(a(2), 'Position');
pos(1) = 0.48; % x
set(a(2), 'Position', pos)
```

The created function being called above that obtains the disparity map is the following:

```
function disparity_map = disparityMap(I_l, I_r, W)
disparity_map = zeros(size(I_l)); % initialize
for y = 1:size(I_l,1)
    for x = 1:size(I_l,2)
        % Window
        xo = x - (W-1)/2;
        xf = x + (W-1)/2;
        yo = y - (W-1)/2;
        yf = y + (W-1)/2;

        % Check we are within limits
        range_y = yo:yf;
        idx_y_valid = find(range_y > 0 & range_y <= size(I_l,1));
        range_x = xo:xf;
        idx_x_valid = find(range_x > 0 & range_x <= size(I_l,2));

        % Create window matrix of intensity for left image
        initial_w_l = zeros(W,W);
        initial_w_l(idx_y_valid, idx_x_valid) = I_l(range_y(idx_y_valid),range_x(idx_x_valid));

        % Disparity
        %range_d = -(size(I_l,2)-x):(x-1); % explore all the points in the right image
        range_d = 0:(x-1); % in our case there will only be positive disparities

        C = NaN(1,length(range_d)); % SSD cost
        for idx_d = 1:length(range_d)
            d = range_d(idx_d); % disparity
            w_r = zeros(W,W); % right window will contain intensities

            % Check we are within limits
            range_xr = xo-d:xf-d;
            idx_xr_valid = find(range_xr > 0 & range_xr <= size(I_l,2));
            global_x_valid = idx_x_valid(ismembc(idx_x_valid,idx_xr_valid));

            % Change left window and keep only the valid rows
            w_l = zeros(W,W);
            w_l(idx_y_valid, global_x_valid) = initial_w_l(idx_y_valid, global_x_valid);

            % Create window matrix of intensities for right image
            w_r(idx_y_valid, global_x_valid) = I_r(range_y(idx_y_valid), range_xr(global_x_valid));

            % Assign SSD cost
            C(idx_d) = sum(sum((w_l - w_r).^2));

%             if C(idx_d) < 10e-5
%                 break
%             end
        end

        % Plot SSD cost vs disparity
%         figure
%         plot(range_d, C)
%         xlabel('Disparity')
%         ylabel('SSD')

        % Best matching disparity for this point: with highest similarity measure
        [~, index] = min(C);
        disparity_map(y,x) = range_d(index);
    end
end
end
```

Then, the depth map is calculated and we plot a subplot comparing different types of depth map (the original one, after changing the focal length and adding Gaussian noise).

```
%% Q2.2c) Q2.2d) Depth maps %%%
% Depth is inversely proportional to
```

```
disparity

% Samsung Galaxy S7: Sensor size (5.76 mm x 4.29mm) and 12MP
% Pixel to mm
%disparity_mm = disparity_map*5.76/(12*10^6);

a = [];

% 1 - Original depth map
z = f*b./disparity_map;
z(z == Inf) = max(z(isfinite(z))); % cap max depth
figure
a(1) = subplot(1,3,1);
imshow(z)
originalSize1 = get(gca, 'Position');
title('Original depth map','FontSize',20);
colormap(gca,gray);

% 2 - Changing focal length
new_f = f+2;
z = new_f*b./disparity_map;
z(z == Inf) = max(z(isfinite(z))); % cap max depth
a(2) = subplot(1,3,2);
imshow(z)
originalSize2 = get(gca, 'Position');
title('Changing focal length','FontSize',20);
colormap(gca,gray);

% 3 - Add random noise to the disparity map
mean_noise = 1;
std_noise = 1;
noise = normrnd(mean_noise, std_noise, size(disparity_map,1), size(disparity_map,2));
disparityMapNoise = disparity_map + noise;
%J = imnoise(disparity_map, 'gaussian', 1, 0.5);
% figure
% imshow(disparityMapNoise, [0, size(I_l,2)-1]);
% colormap(gca,jet)
z = f*b./disparityMapNoise;
z(z == Inf) = max(z(isfinite(z))); % cap max depth
a(3) = subplot(1,3,3);
imshow(z);
originalSize3 = get(gca, 'Position');
title('Adding noise','FontSize',20);
colormap(gca,gray);
c = colorbar('FontSize',16);
c.Label.String = 'Depth (mm)';
c.Label.FontSize = 18;
c.Ticks = linspace(0, 1, 12);
c.TickLabels = [' ',num2cell(500:500:5000), ' '];
set(a(1), 'Position', originalSize1)
set(a(2), 'Position', originalSize2)
set(a(3), 'Position', originalSize3)
pos = get(a(2), 'Position');
pos(1) = 0.36; % x
set(a(2), 'Position', pos)
pos = get(a(3), 'Position');
pos(1) = 0.59; % x
set(a(3), 'Position', pos)
```

2.2(b) Stereo image rectification

In the main script, we perform stereo image rectification:

```
%% Q2.2e) Stereo image rectification %%%

% Create orthogonal unit vectors
r1 = [epipole/norm(epipole);0];
r2 = [-epipole(2), epipole(1), 0]'/norm(epipole);
r3 = cross(r1,r2);

% Orthogonal matrix
R_rect = [r1'; r2'; r3'];

R = eye(3); % rotation matrix
R_l = R_rect; % left rotation matrix
R_r = R*R_rect; % right rotation matrix

% Obtain rectified images
tic
I_l_rectified = rectifyImage(I_l,R_l,f); % left image
I_r_rectified = rectifyImage(I_r,R_r,f); % right image
toc

figure
imshowpair(I_l_rectified,I_r_rectified,'montage')
```

The function being called that rectifies the images is:

```
function I_rectified = rectifyImage(I,R,f)
% It stereo rectifies the image given a rotation matrix.

% Create matrix with all the point coordinates
idx = 1;
points_coord = zeros(3,size(I,1)*size(I,2));
for y = 1:size(I,1)
    for x = 1:size(I,2)
        points_coord(:,idx) = [x,y,f]'; % save point coordinates
        idx = idx + 1;
    end
end

% Apply rotation (in a vectorized way)
new_points = R*points_coord;
rectified_points = f./new_points(3,:).*new_points; % position of this point
rectified_points = ceil(rectified_points(1:2,:));

I_rectified = [];
for i = 1:size(rectified_points,2)
    if rectified_points(:,i) > 0 % keep only positive coordinates
        I_rectified(rectified_points(2,i),rectified_points(1,i)) = I(points_coord(2,i),points_coord(1,i));
    end
end

I_rectified = uint8(I_rectified);
end
```

Finally, some Matlab functions were used to test our images. Here, we show the Matlab implementation to obtain the disparity map, depth map and stereo rectified images of the Tsukuba sequence.

```
%% MATLAB FUNCTIONS

I1 = imread('scene1.row3.col1.ppm');
I2 = imread('scene1.row3.col2.ppm');
I1gray = rgb2gray(I1);
I2gray = rgb2gray(I2);

%% Disparity map and depth with Tsukuba
disparityRange = [0 16];
disparity_tsukuba = disparity(I1gray,I2gray,'BlockSize',21,'DisparityRange',disparityRange);
a = [];
figure
a(1) = subplot(1,2,1);
imshow(disparity_tsukuba,disparityRange);
title('Disparity map','FontSize',20);
colormap(gca,jet)
originalSize1 = get(gca, 'Position');
c = colorbar('FontSize',16);
c.Label.String = 'Disparity (pixels)';
c.Label.FontSize = 18;

z = f*b./disparity_tsukuba;
z(z == Inf) = max(z(isfinite(z))); % cap max depth
a(2) = subplot(1,2,2);
imshow(z,[unique(min(min(z))) unique(max(max(z)))]);
title('Depth map','FontSize',20);
colormap(gca,gray)
originalSize2 = get(gca, 'Position');
c = colorbar('FontSize',16);
c.Label.String = 'Depth (mm)';
c.Label.FontSize = 18;

set(a(1), 'Position', [originalSize1(1)-0.05 originalSize1(2:4)])
set(a(2), 'Position', originalSize2)
pos = get(a(2), 'Position');
pos(1) = 0.53; % x
set(a(2), 'Position', pos)

%% Stereo rectified images with Tsukuba
% Set 1 to visualize and 0 else
visualize = 1;

if (visualize == 1)
    figure;
    imshowpair(I1, I2,'montage');
    title('I1 (left); I2 (right)');
    figure(2);
    imshow(stereoAnaglyph(I1,I2));
    title('Composite Image (Red - Left Image, Cyan - Right Image)');
end

% Collect interest points
blobs1 = detectSURFFeatures(I1gray, 'MetricThreshold', 2000);
blobs2 = detectSURFFeatures(I2gray, '
```

```
    MetricThreshold', 2000);

if (visualize == 1)
    figure;
    imshow(I1);
    hold on;
    plot(selectStrongest(blobs1, 30));
    title('Thirty strongest SURF features in I1');

    figure;
    imshow(I2);
    hold on;
    plot(selectStrongest(blobs2, 30));
    title('Thirty strongest SURF features in I2');
end

% Find point correspondences
[features1, validBlobs1] = extractFeatures(I1gray, blobs1);
[features2, validBlobs2] = extractFeatures(I2gray, blobs2);

% Match featues using SAD
indexPairs = matchFeatures(features1, features2, 'Metric', 'SAD','MatchThreshold', 5);

matchedPoints1 = validBlobs1(indexPairs(:,1),:);
matchedPoints2 = validBlobs2(indexPairs(:,2),:);

if (visualize == 1)
    figure;
    showMatchedFeatures(I1, I2, matchedPoints1, matchedPoints2);
    legend('Putatively matched points in I1', 'Putatively matched points in I2');
end

% Remove outliers using Epopolar Constraints
[fMatrix, epipolarInliers, status] = estimateFundamentalMatrix(...
  matchedPoints1, matchedPoints2, 'Method', 'RANSAC', ...
  'NumTrials', 10000, 'DistanceThreshold', 0.8, 'Confidence', 99.99);

if status ~= 0 || isEpipoleInImage(fMatrix, size(I1)) ...
  || isEpipoleInImage(fMatrix', size(I2))
  error(['Either not enough matching points were found or '...
         'the epipoles are inside the images. You may need to '...
         'inspect and improve the quality of detected features ',...
         'and/or improve the quality of your images.']);
end

inlierPoints1 = matchedPoints1(epipolarInliers, :);
inlierPoints2 = matchedPoints2(epipolarInliers, :);

if (visualize == 1)
    figure;
    showMatchedFeatures(I1, I2, inlierPoints1, inlierPoints2);
    legend('Inlier points in I1', 'Inlier points in I2');
end

% Rectify Images
[t1, t2] = estimateUncalibratedRectification(fMatrix, ...
  inlierPoints1.Location, inlierPoints2.Location, size(I2));
tform1 = projective2d(t1);
tform2 = projective2d(t2);

[I1Rect, I2Rect] = rectifyStereoImages(I1, I2, tform1, tform2);
if (visualize == 1)
    figure;
    imshowpair(I1Rect, I2Rect,'montage');
    figure;
    imshow(stereoAnaglyph(I1Rect, I2Rect));
    title('Rectified Stereo Images (Red - Left Image, Cyan - Right Image)');
end
```