EECE 5639 - Computer Vision
Project 2

Neeraj Sahasrabudhe | Girish Raut

Abstract:

The aim of this project was to find corner features in multiple images and align the images in a mosaic using the estimated homography between corresponding image features. The project was executed in MATLAB using the dataset provided by the instructor. The final image mosaic was obtained consisting of 5 stitched images. The results, algorithm, and observations are documented in this report.

Flowchart:

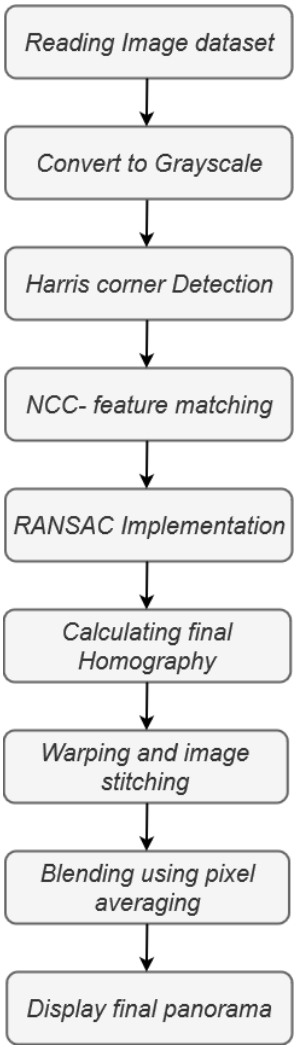

Figure 1: Flowchart for Image Mosaicing

Algorithm:

The Harris Corner detector method was used for detecting the corners in images. The detected corners between consecutive images were matched with each other by calculating their Normalized Cross-Correlation. The image pairs were segregated as inliers or outliers by running the RANSAC algorithm over multiple sets of homographies. Finally, a set of inliers having a maximum count was chosen and the correct homography was calculated.

The calculated homography was used for warping images to match the coordinate system of a reference image. Finally, the warped images were joined by aligning and adding them in the correct order. The resultant image obtained in grayscale was mapped to its corresponding RGB image to get the final mosaic.

Experimentation:

1) Corner Detection

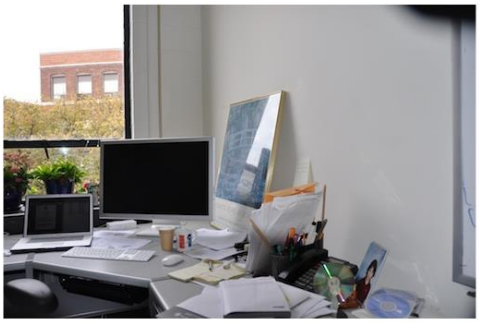

Figure 1: Original Image

Figure 2: Grayscale and Filtered Image

Before implementing the Harris Corner detection algorithm, the images were filtered using a Gaussian filter (\sigma = 1.5) to smoothen the image. This was done to eliminate any noisy high gradients present in the image. The gradients were calculated using the Prewitt operator in both X and Y axis. The resultant values of gradients were stored in G_x and G_y matrices.

The **G_x^2 , G_y^2 and G_xG_y** matrices were calculated for calculating the Harris Corner Detection Matrix (A). The calculated matrices were again filtered using a Gaussian filter for avoiding sharp gradients in the images.

The matrix A was calculated as follows:

$$A = \begin{bmatrix} \boldsymbol{Gx^2} & \boldsymbol{GxGy} \\ \boldsymbol{GxGy} & \boldsymbol{Gy^2} \end{bmatrix}$$

After calculating matrix A for all pixels in an image., the Response Function (R) was calculated The R function is defined as:

$$\boldsymbol{R = det(A) - k * (trace(A))^2}$$

Thresholding was performed to eliminate corner points having lower values of R. Corners with R values less than 20,000 were neglected. The threshold value was decided on a trial and error basis. After thresholding, **Non-max suppression** (NMS) was done to eliminate corner points closer to each other. This was done to ensure that **uniqueness** was maintained in each corner feature. For NMS, *imregionalmax()* function was used in MATLAB.

Figure 3: Output of Harris Corner Detection Algorithm

2) Feature Matching using Normalized Cross Correlation

After successfully calculating the corners in every image, the next task was to match each corner with its counterpart in the neighboring image. To determine a matching corner point, Normalized Cross Correlation(NCC) between every corner point in Image A with that of image B was calculated. The NCC was calculated by using the ***corr2()*** function in MATLAB. The function returned values ranging between -1 and 1. In order to eliminate wrong matches, corner pairs with NCC score higher than **0.95** were selected.

The matched points were visualized using ***showMatchedFeatures()*** function in MATLAB. The output of the function was as shown in Fig. 4.

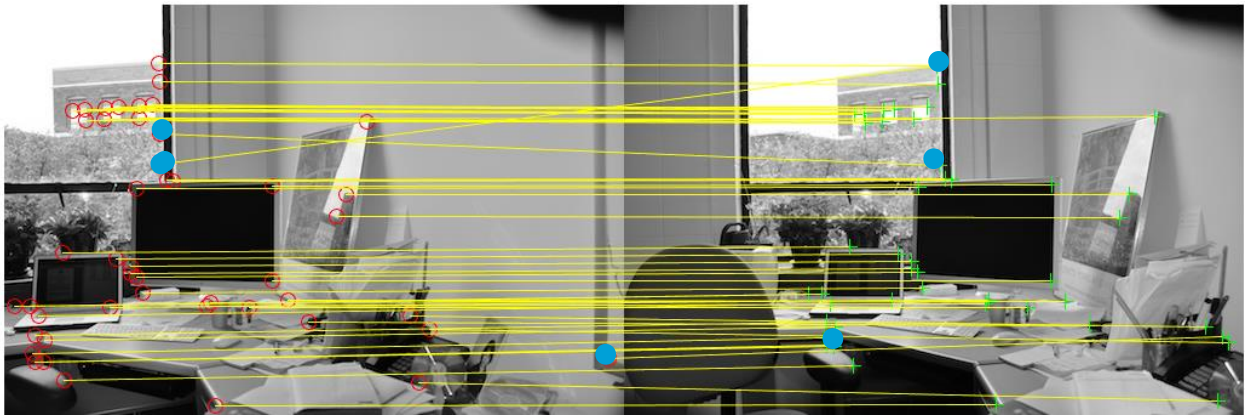

Figure 2: Matched features with Outliers

From Fig. 1, it can be observed that there are some feature matches which are incorrect (as marked in blue). Estimating homography using these incorrect matches would result in a **substantial error** in the final mosaic. Thus, to eliminate these outliers, RANSAC algorithm was implemented.

3) Implementation of RANSAC Algorithm

At least **4 feature-matches** are required to estimate homography between any two consecutive images. The detected Harris Corner features were used for finding corresponding corner features in neighboring images (described in section 2). These feature-matches between two neighboring images was used for calculating the homography by solving the following matrix equation:

$$\boldsymbol{A\,h = 0}$$

Where,
A is (2N X 9) matrix with N being the number of points present in each image.
h is homography (9 X 1) matrix containing.
0 is zero valued (2N X 1) column matrix .

$$\boldsymbol{(A^T A)h = A^T 0}$$

As $A^T A$ is a 9X9 matrix, its SVD was calculated

$$SVD\ of\ A^T A = [\mathrm{U\ S\ V}]$$

The **homography matrix h** is the column of U which represents the smallest eigen value in **S**.

In order to eliminate any **outliers** present in the matched-features data set, **4 points** were **repeatedly sampled** to calculate the homography matrix h. After calculating the homography h, each feature point in image 1 was mapped to a point in image 2. The Euclidean distance between the actual feature point in image 2 and the estimated feature point was considered as the error. A threshold value (3 pixels) for the error term was selected to classify the point pair as an inlier or an outlier.

A set of point-pairs which generated maximum number of inliers was selected for further operations. Final **Homography H** was calculated using all the inliers. From here on, this calculated Homography H was used for all the transformations. The matched features between 2 images after removing the outliers can be seen in Fig. 3.

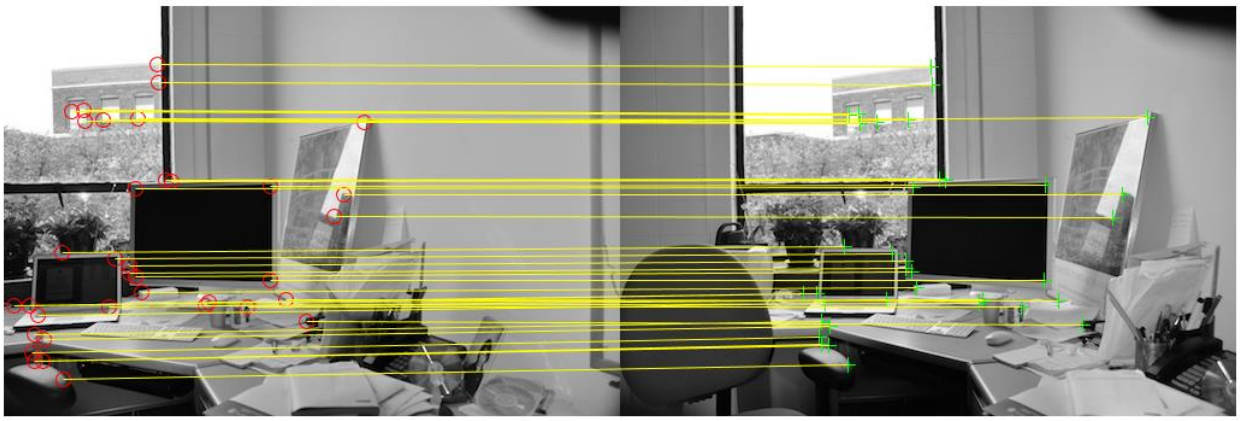

Figure 3: Matched features without outliers

4) Image Warping

Further, the central image in the dataset was selected as the reference image for all the transformations. Transformation of each image with respect to this reference image was calculated and used for the rest of the computations.

The next step after calculating the correct homography was to estimate the size of the final stitched image. The corners of each image in the dataset were warped using the calculated homography. The output limits of each transformed image were obtained using the ***outputLimits()*** function in MATLAB. The difference in the maximum and minimum limits in both X and Y axes was considered the **width** and **height** of the final panorama. An array of zeros with the calculated height and width as its dimensions was generated as the base panorama.

For every image in the data set, ***meshgrid()*** function was used for generating an array with ascending numbers. This grid of numbers was multiplied with the calculated homography H to get coordinates of the output image. The ***interp2()*** function was used for mapping every point in source image with the destination image. The generated images can be seen in Fig. 4

5) Image Stitching and Pixel Averaging

The warped images were merged into a single panorama by using the **pixel intensity averaging** method. The panorama was initially loaded with the first image in the series. Every pixel in the panorama was traversed and compared with the next image in the series. The overlapping pixels were replaced with the average of the current image and the next image in the series.

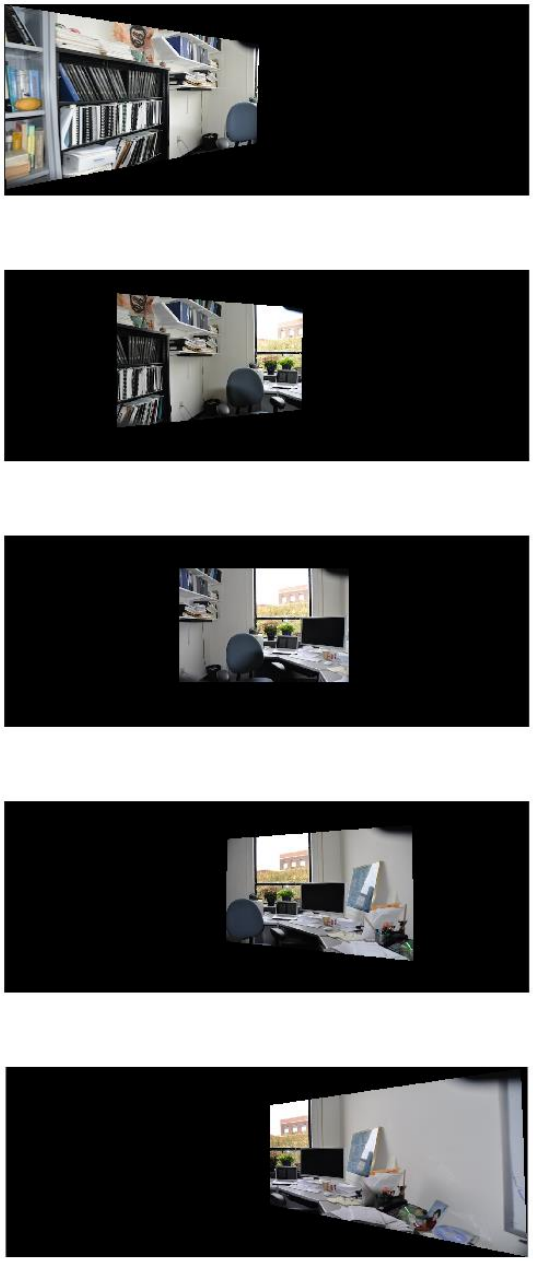

Figure 4: Warped Images (5 images from Office Dataset)

Observations

1) It was observed that while detecting corner features using Harris Corner Detector, running the detector over original grayscale images resulted in **clustered** and **incorrect** corner features. Thus, **smoothening** the images using a Gaussian filter helped in reducing unnecessary variance in gradient.
2) **Thresholding** the R function using a suitable value and running **Non-max Suppression** over all the detected features resulted in unique and correct corner features.
3) RANSAC algorithm helped in eliminating all the incorrect correspondences between images and resulted in correct Homography estimation.

4) Using the first image as a reference image resulted in a disoriented and distorted panorama. This happens because as the **chain of homographies** increases from image 1 to image 5, the noise in the warped images increases.

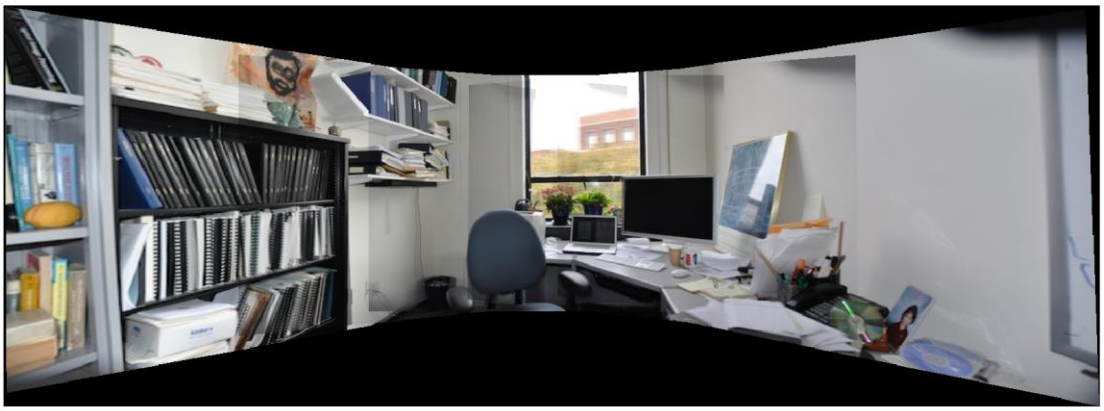

Figure 5: Final Panorama

Conclusions

1) Harris Corner Detector algorithm can be used to detect corner features in input images.
2) Normalized cross correlation can be used to determine matched features between two consecutive image frames,
3) Outlier rejection algorithm like RANSAC can be used for eliminating incorrect correspondences and thus estimate better homography.
4) The calculated Homography can be used to warp images and generate a panorama of multiple images.

Extra Credit:

1) Read the source and destination images.
2) Got the points in the destination image to be replaced by the source image using ***ginput()*** in array 1, and the corner points of the source image in array 2.
3) Calculated the Homography matrix H between the two sets of points using SVD calculation method.
4) Using H, warped the source image so as to fit in the selected area.
5) Displayed the final output by overlaying the destination image with the warped source image.

Output-

Figure 6: Output for Extra Credit Question

Project 2 - Image Mosacing

Read Image Dataset

```
% Set data set directories
office_dir = fullfile("DanaOffice/");
hallway_dir = fullfile("DanaHallWay1/");
check_board_dir = fullfile("CheckerBoard/");

% Select Directory Handle to be used
curr_dir_handle = imageDatastore(office_dir);
I = readimage(curr_dir_handle,1);
imshow(I);

% Get number of images in the data set
numOfImages = numel(curr_dir_handle.Files)
```

Matrix for stroring Original and Grayscale images

```
% Generating Arrays of original image and Grayscale Images
gray_scale_images = zeros(size(I,1),size(I,2),numOfImages);
color_images = zeros(size(I,1),size(I,2),3,numOfImages);

for i=1:numOfImages
    I = readimage(curr_dir_handle,i);
    grayImage = im2gray(I);
    gray_scale_images(:,:,i) = grayImage;
    color_images(:,:,:,i) = I;
end
disp(numOfImages + " Images stored in 3D array : gray_scale_images")
```

Implement Harris Detector over all images

```
% Array for storing detected feature coordinates
harris_features = [];

for i=1:numOfImages
    [x,y] = getHarrisFeatures(gray_scale_images(:,:,i),size(gray_scale_images(:,:,i)), 1.5, 21000, 0.04,1);
    harris_features = [harris_features; [x,y,zeros(numel(x),1) + i]];
end
```

Feature Matching and Outlier Rejection (RANSAC)

```
homographies = [];
tforms(numOfImages) = projtform2d;
tforms(:).A
% Initialize variable to hold image sizes.
imageSize = zeros(numOfImages,2);

for n=2:numOfImages

    img1 = n;
    img2 = n-1;

    img1_features = getImageFeatures(harris_features,img1);
    img2_features = getImageFeatures(harris_features,img2);

    window_size = 11;

    % Implement Normatlized Cross-corelation

    NCC_ARRAY = getNCCArray(gray_scale_images,img1,img2,getImageFeatures(harris_features,img1),getImageFeatures(harris_features,img2), wir

    % Homography Estimation and RANSAC

    [inliers, img1_matched, img2_matched] = getRANSACInliers(img1_features,img2_features,NCC_ARRAY);

    figure;
    % Show matched features - ALL
    showMatchedFeatures(gray_scale_images(:,:,img1),gray_scale_images(:,:,img2),img1_matched,img2_matched,"montage");

    figure;
    % Show matched features with inliers
    showMatchedFeatures(gray_scale_images(:,:,img1),gray_scale_images(:,:,img2),img1_matched(inliers,:),img2_matched(inliers,:),"montage")
```

```
    % Etimate homography from Inliers

    % Calculate A matrix
    A = getMatrixA(numel(inliers),img1_matched(inliers,:),img2_matched(inliers,:));

    % Estimate homography by using SVD method
    [U,S,V] = svd(A' * A);
    H = reshape(U(:,9),[3 3])';
    H = H ./ H(3,3)

    tforms(n).A = tforms(n-1).A * H;

    % SAVE IMAGE SIZE
    I = readimage(curr_dir_handle, n);

    % Convert image to grayscale.
    grayImage = im2gray(I);

    % Save image size.
    imageSize(n,:) = size(grayImage);
end

tforms(:).A
```

Warping Images

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

% Determine Center Image index
avgXLim = mean(xlim, 2);
[~,idx] = sort(avgXLim);
centerIdx = floor((numel(tforms)+1)/2);
centerImageIdx = idx(centerIdx);

Tinv = invert(tforms(centerImageIdx))
for i = 1:numel(tforms)
    tforms(i).A = Tinv.A * tforms(i).A;
end

for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), [1 imageSize(i,2)], [1 imageSize(i,1)]);
end

maxImageSize = max(imageSize);

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

yMin = min([1; ylim(:)]);
yMax = max([maxImageSize(1); ylim(:)]);

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

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

imshow(panorama);
```

Image Warping

```

% Create a 2-D spatial reference object defining the size of the panorama
xLimits = [xMin xMax];
yLimits = [yMin yMax];
panoramaView = imref2d([height width], xLimits, yLimits);

% Defining arrays for 3 channels of colored image
RGB_img_R = zeros(height,width,numOfImages);
RGB_img_G = zeros(height,width,numOfImages);
RGB_img_B = zeros(height,width,numOfImages);

figure;
hold on;
```

```
% Create the panorama.
for i = 1:numOfImages
    H_ = tforms(i).A;
    H_ = inv(H_) * [1 0 xlim(1,1);0 1 ylim(1,1); 0 0 1];
    [xi, yi] = meshgrid(1:width,1:height);

    xx = (H_(1,1)*xi + H_(1,2) * yi + H_(1,3)) ./ (H_(3,1)*xi + H_(3,2)*yi + H_(3,3));
    yy = (H_(2,1)*xi + H_(2,2) * yi + H_(2,3)) ./ (H_(3,1)*xi + H_(3,2)*yi + H_(3,3));

    % Implement interp2 function to interpolate points from Image 2 to
    % Image 1
    foo_R = uint8(interp2(color_images(:,:,1,i),xx,yy));
    foo_G = uint8(interp2(color_images(:,:,2,i),xx,yy));
    foo_B = uint8(interp2(color_images(:,:,3,i),xx,yy));

    RGB_img_R(:,:,i) = foo_R;
    RGB_img_G(:,:,i) = foo_G;
    RGB_img_B(:,:,i) = foo_B;

    IMG1(:,:,:,i) = cat(3,foo_R,foo_G,foo_B);
    subplot(5,1,i);
    imshow(IMG1(:,:,:,i),[0 255]);

end
hold off;
figure
```

Image Stitching using Pixel Averaging Method

```
panorama(:,:,:) = IMG1(:,:,:,1);

for i =2:numOfImages
    for j=1:size(panorama,1)
        for k =1:size(panorama,2)
            if(sum(panorama(j,k,:),'all')~=0 && sum(IMG1(j,k,:,i),'all')~=0)
                panorama(j,k,:) =  mean([IMG1(j,k,:,i) panorama(j,k,:)]);
            else
                panorama(j,k,:) = max([panorama(j,k,:) IMG1(j,k,:,i)]);
            end
        end
    end
end

imshow(panorama,[]);
```

Function Defination

```
function [x,y] = getHarrisFeatures(image,image_size,sigma,thresh,k,disp)

    % filtering the original image
    image = imgaussfilt(image,sigma);

    % Prewitt OPerator
    Ix = [-1 0 1];
    Iy = [-1 0 1]';

    % Convolution
    Gx = imfilter(image,Ix,"symmetric");
    Gy = imfilter(image,Iy,"symmetric");

    Gx_squared = Gx.^2;
    Gy_squared = Gy.^2;
    Gx_Gy = Gx .* Gy;

    Gx_squared = imgaussfilt(Gx_squared,sigma);
    Gy_squared = imgaussfilt(Gy_squared,sigma);
    Gx_Gy = imgaussfilt(Gx_Gy,sigma);

    % Computing the response function R
    R = zeros(image_size);
    for i=1:image_size(1)
        for j=1:image_size(2)
            A_harris = [Gx_squared(i,j) Gx_Gy(i,j);Gx_Gy(i,j) Gy_squared(i,j)];
            R(i,j) = det(A_harris) - (k*(trace(A_harris))^2);
        end
    end
```

```
    % Thresholding the R function
    R = (R>thresh) .* R;

    % Performing Non Max Supression
    R = imregionalmax(R);

    % Finding the pixel coordinates of the non-zero values in th R matrix
    [x,y] = find(R);

    % Displaying Image with Harris Features
    if disp == 1
        figure;
        imshow(image,[]);
        hold on;
        plot(y,x,'+g',"LineWidth",1);
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function image_features = getImageFeatures(image, image_number)
    img = image .* (image_number == image(:,3));
    img = nonzeros(img');
    image_features = reshape(img',[3 size(img,1)/3])';
    image_features = image_features(:,1:2);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function A = getMatrixA(num_of_points,img1_matched,img2_matched)
    A = zeros(1,9);
    random_index = randperm(size(img1_matched,1),num_of_points);

    for i=1:num_of_points
        P1 = img1_matched(random_index(i),:);
        P2 = img2_matched(random_index(i),:);

        x1 = P1(1);
        x2 = P2(1);
        y1 = P1(2);
        y2 = P2(2);

        A(end+1,:) = [x1 y1 1 0 0 0 -x1*x2 -y1*x2 -x2];
        A(end+1,:) = [0 0 0 x1 y1 1 -x1*y2 -y1*y2 -y2];
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % FUNCTION getNCCArray() - Sample FUNCTION CALL
    % 1) IMAGE ARRAY
    % 2) Img1
    % 3) Img2
    % 4) Image1 Features
    % 5) Image2 Features
    % 6) Window Size

function NCC_ARRAY = getNCCArray(gray_scale_images,img_1,img_2,img1_features,img2_features,window_size)

    % Define Image 1 and Image 2
    img1 = gray_scale_images(:,:,img_1);
    img2 = gray_scale_images(:,:,img_2);

    % Padding for consttructing neighbourhood
    padding = floor(window_size/2);

    % Defining arrays
    NCC_ARRAY = zeros(1,3);
    final_ncc_score = zeros(1,3);

    for i=1:size(img1_features,1)

            prev_score = -1;

            % NEIGHBOURHODD in IMAGE 1
            try
                I = img1(img1_features(i,1)-padding:img1_features(i,1)+padding,img1_features(i,2)-padding:img1_features(i,2)+padding);
            catch
                continue
            end
```

```
            for j=1:size(img2_features,1)

                % NEIGHBOURHOOD in IMAGE 2
                try
                    J = img2(img2_features(j,1)-padding:img2_features(j,1)+padding,img2_features(j,2)-padding:img2_features(j,2)+padding);
                catch
                    continue
                end

                % Finding the cross-corelation value
                NCC_score =  corr2(I,J);

                % For ensuring one to one mapping
                if NCC_score>prev_score
                    final_ncc_score = [i j NCC_score];
                    prev_score = NCC_score;
                end
            end

            % Thresholding the value of NCC
            if(final_ncc_score(1,3)>0.96)
                NCC_ARRAY(end+1, :) = final_ncc_score;
            end

    end

    NCC_ARRAY = NCC_ARRAY(2:end,:);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [highest_inliers, img1_matched, img2_matched] = getRANSACInliers(img1_features,img2_features,NCC_ARRAY)
    img1_matched = img1_features(NCC_ARRAY(:,1),:);
    img2_matched = img2_features(NCC_ARRAY(:,2),:);

    img1_matched = fliplr(img1_matched);
    img2_matched = fliplr(img2_matched);

    % Sample random points

    num_of_points = 4;

    inliers = [];
    highest_inliers = [];

    for j=1:100
        try
            A = getMatrixA(num_of_points,img1_matched,img2_matched);
        catch
            continue
        end
        [U,S,V] = svd(A' * A);
        H = reshape(U(:,9),[3 3])';
        H = H ./ H(3,3);

        inliers = [];

        for i= 1:size(img1_matched)
            img2_res = H * [img1_matched(i,1) img1_matched(i,2) 1]';
            img2_res = img2_res ./ img2_res(3,1);
            error_ = dist([img2_res(1,1) img2_res(2,1)], [img2_matched(i,1) img2_matched(i,2)]');
            if error_ < 3
                inliers(end+1) = i;
            end
        end

        if numel(inliers)>numel(highest_inliers)
            highest_inliers = inliers;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```