COMPUTER VISION HOMEWORK 2

LIUJIANG YAN

The folder contains following matlab files.
Main functions:

* detect_features.m - interest feature detector
* match_features.m - match the interest features by minimum SSD
* compute_affine_xform.m - calculate the affine transformation by RANSAC
* compute_proj_xform.m - calculate the projective transformation by RANSAC
* ssift_descriptor.m - calculate the SIFT descriptor for each feature
* matches_ratio_test.m - match the features by ratio test

Supporting functions:

* display_matched.m - display side-by-side view and lines connected matched feature
* stitch.m - display stitched image with two image and transformation
* classify.m - accept a matrix of angles and return its corresponding matrix of bin with label 1 to 8
* nonmaxsuppts.m - given function to find local maximum

Driver funciton:

* hw2driver.m - driver file for this assignment

This report consists of three parts:

* Programming Assignment
* Supporting Functions
* Results and Discussion
* Driver and Feasible Parameters

Programming Assignment
(a) Feature Detection
Identify points of interest in the images using the Harris corner detection method.
Here we use Roberts operator to perform gradient.

```
function [rows,cols] = detect_features(image)
    image = rgb2gray(image);

    % robbert matrix for derivative calculation
    robbert_x = [0 1; -1 0];
    robbert_y = [1 0; 0 -1];

    % calculate the derivative of both directions
    Ix = filter2(robbert_x, double(image));
    Iy = filter2(robbert_y, double(image));

    % construct 3*3 gaussian matirx
```

```
    h=fspecial('gaussian',[5 5],2);

    % construct Ix^2 Iy^2 IxIy with gaussian filter
    Ix_square = filter2(h, (Ix.*Ix));
    Iy_square = filter2(h, (Iy.*Iy));
    Ix_Iy = filter2(h, (Ix.*Iy));

    % calculate the R response matrix
    k = 0.06;
    R = Ix_square.*Iy_square-Ix_Iy.*Ix_Iy-k*(Ix_square+Iy_square).^2;

    % get the 85% quantile of all response as the threshold
    threshold = quantile(R(:),0.85);

    % call the nonmaxsuppts to get the interest features
    [rows, cols] = nonmaxsuppts(R,2,threshold);
end
```

(b)Feature Matching

Here we describe the distance of two features by SSD (Sum of Squared Distance). For convenience we discard the corner and consider a patch around the feature pixel of size 21 by 21.

$$SSD(\vec{v}_1, \vec{v}_2) = \frac{||\vec{v}_1 - \vec{v}_2||}{||\vec{v}_1||||\vec{v}_2||}$$

We construct a whole matrix with SSD between each possible pair of matches in the first pass. In the second pass, we loop by the row, find the feature of minimum SSD for each row, labelled as l_2, and find the corresponding feature of minimum SSD for l_2, labelled as l_1. If l_1 agrees with the index of row, then $[l_1, l_2]$ is a match.

```
function matches = match_features(feature_coords1,feature_coords2,...
                                        image1,image2)
    image1_gray = rgb2gray(image1);
    image2_gray = rgb2gray(image2);

    matches = [];
    % predefine the ssd with the size
    ssd = inf(length(feature_coords1), length(feature_coords2));
    % boundary
    [h1, w1,~] = size(image1);
    [h2, w2,~] = size(image2);
    % patch size
    side = 10;
    patch = (2*side+1)^2;

    % loop feature_coords of first image and second image
    for i=1:length(feature_coords1)
        x1 = feature_coords1(i,1);
        y1 = feature_coords1(i,2);
        % ignore the border for convenience
```

```
        if x1 > 10 && x1 < h1-10 && y1 > 10 && y1 < w1-10
            vector_1 = double(reshape(image1_gray(x1-side:x1+side,...
                y1-side:y1+side),patch,1));

            for j=1:length(feature_coords2)
                x2 = feature_coords2(j,1);
                y2 = feature_coords2(j,2);

                if x2 > 10 && x2 < h2-10 && y2 > 10 && y2 < w2-10
                    vector_2 = double(reshape(image2_gray(x2-side:x2+side,...
                        y2-side:y2+side),patch,1));

                    ssd(i,j) = norm(vector_1 - vector_2)...
                        /norm(vector_1)/norm(vector_2);
                end
            end
        end
    end

    matches = [];
    % loop the line
    for i=1:length(feature_coords1)
        % get the index of maximum ncc of i row, store as label_2
        [~,label_2] = min(ssd(i,:));
        % get the index of maximum ncc of label_2, store as label_1
        [~,label_1] = min(ssd(:,label_2));
        % if label_1 and i are equal, then it is a good match, store.
        if label_1 == i
            matches = [matches; [label_1, label_2]];
        end
    end

    %display matches
    display_matched(image1, image2,...
        feature_coords1, feature_coords2, matches);
end
```

(c) Displaying Matches

To display the matches of features in two images, we draw a circle around each feature point in blue. Then we connect each matches by line. Here we draw the line in red at default. For the following transforming problems, we draw lines between each agreed matches in green and those not agreed in red.

```
function res = display_matched(image1, image2, ...
                               features1, features2, matches_mark)

    % get the offset
    [~,offset,~] = size(image1);

    [h1, w1] = size(image1);
```

```
    [h2, w2] = size(image2);
    % check if the size of images are consistent
    if h1 < h2
        image1(h1+1:h2,:)=0;
    elseif h1 > h2
        image2(h2+1:h1,:)=0;
    end

    % display the side-by-side image
    SbS = [image1 image2];
    imshow(SbS);

    % the radius of the mark of the feature
    r=3;
    for i=1:length(matches_mark)
        % draw the mark of every feature in circluar shape
        rectangle('Position',[features1(matches_mark(i,1),2)-r, ...
            features1(matches_mark(i,1),1)-r, 2*r, 2*r],...
            'Curvature',[1 1],'edgecolor','b');
        rectangle('Position',[features2(matches_mark(i,2),2)-r+offset, ...
            features2(matches_mark(i,2),1)-r, 2*r, 2*r],...
            'Curvature',[1 1],'edgecolor','b');
    end

    % deal with the non-marked matches (non-transformed case)
    [~,width] = size(matches_mark);
    if width == 2
        matches_mark(:,3) = 0;
    end

    for i=1:length(matches_mark)
        % get the endpoints of the line connecting two matched features
        startpoint = [features1(matches_mark(i,1),1), ...
                      features2(matches_mark(i,2),1)];
        endpoint = [features1(matches_mark(i,1),2), ...
                      features2(matches_mark(i,2),2)+offset];
        % if agree with the transformation, green
        % if not, red
        if matches_mark(i,3) == 1
            line(endpoint,startpoint,'LineWidth',0.3,'Color','g');
        else
            line(endpoint,startpoint,'LineWidth',0.3,'Color','r');
        end
    end
end
```

(d) Alignment and Stitching - affine

Here we present RANSAC to select the feasible affine transformation. For the affine transformation, we need at least three pairs of matched features.

Let the point be in homogeneous representation.

$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

so the affine transformation can be calculated following,

$$Ra = b$$

$$R = ba^{-1}$$

$$R = \begin{bmatrix} x_{b1} & x_{b2} & x_{b3} \\ y_{b1} & y_{b2} & y_{b3} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_{a1} & x_{a2} & x_{a3} \\ y_{a1} & y_{a2} & y_{a3} \\ 1 & 1 & 1 \end{bmatrix}^{-1}$$

First pick three pairs of matched features randomly, then calculate the affine transformation, count the inlier and update the affine transformation when needed.
Here we use the third row of matches matrix to record whether this pair of matched features is inlier of this certain transformation.

```
function affine_xform = compute_affine_xform(matches,features1,features2,...
                                                    image1,image2)

    % predefine some matrix for following storing
    x1 = zeros(1,3); y1 = zeros(1,3); x2 = zeros(1,3); y2 = zeros(1,3);

    % let's take N rounds to get the best affine matrix
    N = 10000;

    % record the inlier with a label in third column of matches
    matches_mark = matches;
    matches_mark(:,3) = 0;

    for i=1:N
        % initialize the current mark matrix
        matches_mark_cur = matches_mark;
        matches_mark_cur(:,3) = 0;

        % get a random combination of 3 pairs out of n matching pairs
        trail = randsample(length(matches), 3);

        for j=1:3
            % from the index get the points accordingly
            x1(j) = features1(matches(trail(j),1),1);
            y1(j) = features1(matches(trail(j),1),2);
            x2(j) = features2(matches(trail(j),2),1);
            y2(j) = features2(matches(trail(j),2),2);
        end

        % Let Ra=b, where R stands for the transformation matirx
        % in homogeneous representation
```

```
        a = [x1;y1;1 1 1];
        b = [x2;y2;1 1 1];

        % get the parameter vector t by direct calculation
        t = b/a;

        % loop the points pairs in matches
        % compare the real point to mapped point
        for n=1:length(matches)
            feature = [features1(matches(n,1),:), 1]';
            feature_mapped = [features2(matches(n,2),:), 1]';
            feature_calculated = t*feature;

            % if the difference of the norm of vector difference is less than 10
            if norm(feature_calculated-feature_mapped)<10
                matches_mark_cur(n,3) = 1;
            end
        end

        % if the current count is larger than past record
        % update inlier and pass current t to our record
        if sum(matches_mark_cur(:,3)) > sum(matches_mark(:,3))
            affine_xform = t;
            matches_mark = matches_mark_cur;
        end
    end

    % display the matches
    display_matched(image1, image2, features1, features2, matches_mark)
    figure
    % display the stitched image
    stitch(image1,image2,proj_xform);
end
```

(e) Alignment and Stitching - projective

Like the affine case, here we also present the RANSAC to select the feasible projective transformation. But to calculate the projective transformation, we need at least 4 pairs of matched features.

The projective transformation is a nonlinear one, so we need a different calculation here. The transformation are in the form below,

$$\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_{ai} \\ y_{ai} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{bi} \\ y_{bi} \\ 1 \end{bmatrix}$$

and can translate to the linear system below,

$$\begin{bmatrix} x_{a1} & y_{a1} & 1 & 0 & 0 & 0 & -x_{b1}x_{a1} & -x_{b1}x_{a1} & -x_{b1} \\ 0 & 0 & 0 & x_{a1} & y_{a1} & 1 & -y_{b1}x_{a1} & -y_{b1}x_{a1} & -y_{b1} \\ \dots & & & & & & & & \\ x_{an} & y_{an} & 1 & 0 & 0 & 0 & -x_{bn}x_{an} & -x_{bn}x_{an} & -x_{bn} \\ 0 & 0 & 0 & x_{an} & y_{an} & 1 & -y_{bn}x_{an} & -y_{bn}x_{an} & -y_{bn} \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ \dots \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

$$A_{2n\times 9}h_{9\times 1} = 0_{2n\times 1}$$

where h is the eigenvecor of A^TA with smallest eigenvalue.

```
function proj_xform = compute_proj_xform(matches,features1,features2,...
                                                    image1,image2)
    % predefine some vectors for following storing
    x1 = zeros(4,1); y1 = zeros(4,1);
    x2 = zeros(4,1); y2 = zeros(4,1);

    % let's take N times to get the best affine matrix
    N = 30;

    % record the final matrix and inlier number
    proj_xform = zeros(3,3);

    matches_mark = matches;
    matches_mark(:,3) = 0;

    for i=1:N
        matches_mark_cur = matches_mark;
        matches_mark_cur(:,3) = 0;
        % clear A
        A = [];

        % get the index of pairs for following valuing
        trail = randsample(length(matches), 4);

        for j=1:4
            % from the index get the points accordingly
            x1(j) = features1(matches(trail(j),1),1);
            y1(j) = features1(matches(trail(j),1),2);
            x2(j) = features2(matches(trail(j),2),1);
            y2(j) = features2(matches(trail(j),2),2);

            % construct A
            subA = [x1(j) y1(j) 1 0 0 0 -x2(j)*x1(j) -x2(j)*y1(j) -x2(j); ...
                    0 0 0 x1(j) y1(j) 1 -y2(j)*x1(j) -y2(j)*y1(j) -y2(j)];
            A = [A; subA];
        end

        % get the squre matrixA^TA for eigenvector computing
```

```
        ATA = A'*A;
        [V,D] =  eig(ATA);

        % get the index of minimum eigenvalue
        % the corresponding eigenvector is just h
        D(D==0)=inf;
        [~,index] = min(min(D));
        h = V(:,index);

        % assign the elements of h for correction
        h = reshape(h,3,3)';

        % loop the points pairs to get the inliers for this h
        for n=1:length(matches)
            feature = [features1(matches(n,1),:),1]';
            feature_mapped = [features2(matches(n,2),:),1]';
            feature_calculated = h*feature;
            feature_calculated = feature_calculated/feature_calculated(3);

            % if the difference of the norm of vector difference is less than 10
            if norm(feature_calculated-feature_mapped)<10
                matches_mark_cur(n,3) = 1;
            end
        end

        % if the current count is larger than past record
        % update inlier and pass current t to our record
        if sum(matches_mark_cur(:,3)) > sum(matches_mark(:,3))
            proj_xform = h;
            matches_mark = matches_mark_cur;
        end
    end

    % normalize the transformation matrix so that the last element is 1
    proj_xform = proj_xform/proj_xform(3,3);

    % display the matches
    display_matched(image1, image2, features1, features2, matches_mark)
    figure
    % display the stitched image
    stitch(image1,image2,proj_xform);
end
```

(f) Stitch Images

The affined image has different size from the original one. So first of all we need to find the size of the stitched images, which uses the maximum/minimum height and width index of two images. Also we need to find the displacement from the original image to the final stitched image to align the two images correctly. When we get the two full size image separately, we could just average them and get the output image.

```
function output = stitch(image1, image2, affine_xform)
    % get the size of each image
    [h1, w1, ~] = size(image1);
    [h2, w2, ~] = size(image2);

    % affine_image*image1 = image2
    % get the affined position of each pixel of image 1
    affine_position = zeros(h1, w1, 2);
    for h=1:h1
        for w=1:w1
            position_homo = affine_xform*[h;w;1];
            affine_position(h,w,:) = position_homo(1:2);
        end
    end

    % get the maximum/minimum x y position
    affine_position_h_max = ceil(max(max(affine_position(:,:,1))));
    affine_position_h_min = floor(min(min(affine_position(:,:,1))));

    affine_position_w_max = ceil(max(max(affine_position(:,:,2))));
    affine_position_w_min = floor(min(min(affine_position(:,:,2))));

    % get the range of the stitched image
    range_h = range([1,h2,affine_position_h_min,affine_position_h_max])+1;
    range_w = range([1,w2,affine_position_w_max,affine_position_w_min])+1;

    % define the displacement
    dh = abs(affine_position_h_min) + 1;
    dw = abs(affine_position_w_min) + 1;

    % resize image 1 to the affined postion of the output image size
    img1_resized = zeros(range_h + dh, range_w + dw, 3);
    for h=1:h1
        for w=1:w1
            i = round(affine_position(h,w,1)+dh);
            j = round(affine_position(h,w,2)+dw);
            img1_resized(i,j) = image1(h,w);
        end
    end

    % displace image 2 to the output image size
    img2_resized = zeros(range_h + dh, range_w + dw, 3);
    img2_resized(1+dh:h2+dh,1+dw:w2+dw,:) = image2;

    % stitch together
    output = (img1_resized + img2_resized)/2;
    imshow(uint8(output));

    % or you can just show both by calling the following commands
    % img1 = imshow(uint8(img1_resized));
```

```
    % set(img1, 'AlphaData', 0.5);
    % hold on;
    % img2 = imshow(uint8(img2_resized));
    % set(img2, 'AlphaData', 0.5);
end
```

(g) Feature Description - SIFT
We grid the patch around the feature to 4 by 4 matrix, each of which consists of 10*10 pixels. Then we assign each pixel with a label from 1 to 8 corresponding to its orientation I_x/I_y. Sum up the magnitude of each pixel in the same bin, so that we can represent the submatrix as a 8 by 1 vector. Do the same process for 16 times around the feature, then we get a 128 by 1 vector, which is the descriptor of this feature.
Finally we normalize every vector as the instruction suggests.

```
function [ descriptors ] = ssift_descriptor(feature_coords,image)

    image = rgb2gray(image);
    descriptors = zeros(length(feature_coords),128);

    % Robert
    robert_x = [0 1; -1 0];
    robert_y = [1 0; 0 -1];

    % calculate the derivative of both directions
    Ix = filter2(robert_x, double(image));
    Iy = filter2(robert_y, double(image));
    % the magnitude of each pixel
    Im = sqrt(Ix.^2+Iy.^2);

    % calculate each pixel's orientation and throw it to exact bin
    gradient = zeros(size(Im));
    for i=1:length(Ix(:,1))
        for j=1:length(Ix(1,:))
            gradient(i,j)=atan2(Ix(i,j),Iy(i,j))/pi*180;
        end
    end
    gradient_classified = classify(gradient);

    % loop each feature
    for n=1:length(feature_coords)
        xc = feature_coords(n,1);
        yc = feature_coords(n,2);

        % ignore boundary
        if xc > 20 && xc < length(image(:,1))-20 ...
            && yc > 20 && yc < length(image(1,:))-20

            % count is the label of the 4*4 patch
            count = 0;
```

```
            for i = [-2,-1,0,1]
                for j = [-2,-1,0,1]
                    if i<0
                        x = (xc+i*10):(xc+i*10+9);
                    else
                        x = (xc+i*10+1):(xc+i*10+10);
                    end
                    if j<0
                        y = (yc+j*10):(yc+j*10+9);
                    else
                        y = (yc+j*10+1):(yc+j*10+10);
                    end

                    % get the sub matrix
                    % sum the magnitude of each bin to construct the vector
                    % there are 16 this kind of vector (4*4 submatrix)
                    submatrix = gradient_classified(x, y);
                    subsum = Im(x,y);
                    for bin=0:7
                        % get the index of this exact bin
                        index = (submatrix==(bin+1));
                        % sum up the magnitude
                        sumofbin = subsum(index);
                        sumofbinS = sum(sumofbin(:));
                        % assign to the exact position
                        descriptors(n, 8*count+bin+1) = sumofbinS;
                    end
                    count = count + 1;

                end
            end
        end
    end

    % normalize
    for i=1:length(descriptors(:,1))
        % normalize
        descriptors(i,:) = descriptors(i,:)/norm(descriptors(i,:));
        for j=1:128
            % threshold to 0.2 as maximum
            if descriptors(i,j)>0.2
                descriptors(i,j) = 0.2;
            end
        end
        % normalize again
        descriptors(i,:) = descriptors(i,:)/norm(descriptors(i,:));
    end

end
```

(g) Feature matching with Ratio Test
To get the matched features with two descriptors input, here we use ratio test. First of all, we construct a SSD matrix to store the sum of squared distance of each possible matched pair of features. Then we loop the row and get the shortest and second shortest SSD. If the difference between those two distance satisfies our threshold criteria, then we consider it a matched pair.

```
function matches = matches_ratio_test(descriptors1, descriptors2)
    % the threshold
    threshold = 0.6;
    matches = [];

    % get the ssd between each possible pair of matching
    ssd = zeros(length(descriptors1(:,1)), length(descriptors2(:,1)));
    for i=1:length(descriptors1(:,1))
        for j=1:length(descriptors2(:,1))
            ssd(i,j) =  norm(descriptors1(i,:)-descriptors2(j,:));
        end
    end

    % loop each row, getting the first and second ssd
    % if the difference satisfied the threshold then match
    for i=1:length(descriptors1(:,1))
        [first_shortest_distance, first] = min(ssd(i,:));
        ssd(i,first) = inf;
        [second_shortest_distance, second] = min(ssd(i,:));
        if first_shortest_distance < threshold*second_shortest_distance
            matches = [matches; i, first];
        end
    end
end
```

Supporting Funcitons
(a) Classify angles to specfic bin
Take a matrix of angles lying in $[-\pi, \pi]$.
Return a matrix of label which corresponding to the angle.

```
function m_classfied = classify(m)
    [row, column] = size(m);
    m_classfied = zeros(size(m));
    for r=1:row
        for c=1:column
            if m(r,c) > 157.5 || m(r,c) <= -157.5
                m_classfied(r,c) = 1;
            elseif m(r,c) > -157.5 && m(r,c) <= -112.5
                m_classfied(r,c) = 2;
            elseif m(r,c) > -112.5 && m(r,c) <= -67.5
                m_classfied(r,c) = 3;
            elseif m(r,c) > -67.5 && m(r,c) <= -22.5
```

```
                m_classfied(r,c) = 4;
            elseif m(r,c) > -22.5 && m(r,c) <= 22.5
                m_classfied(r,c) = 5;
            elseif m(r,c) > 22.5 && m(r,c) <= 67.5
                m_classfied(r,c) = 6;
            elseif m(r,c) > 67.5 && m(r,c) <= 122.5
                m_classfied(r,c) = 7;
            elseif m(r,c) > 122.5 && m(r,c) <= 157.5
                m_classfied(r,c) = 8;
            end
        end
    end
end
```

Results and Discussion

(a) Feature Matching

From the results displayed below, the matching criteria based on sum of squared distance does turn out matches in each pair. The matches of features between different images of bikes and between different images leuvens seem good. However the matches of features between walls and between grafs seem a little messy.

Following we will perform algorithms, finding the suitable affine transformation and projective transformation, to check whether the matches are feasible or not.

FIGURE 1. Feature Matching between bikes1 to bikes2 and bikes1 to bikes3

FIGURE 2. Feature Matching between graf1 to graf2 and bikes1 to graf3

FIGURE 3. Feature Matching between leuven1 to leuven2 and leuven1 to leuven3

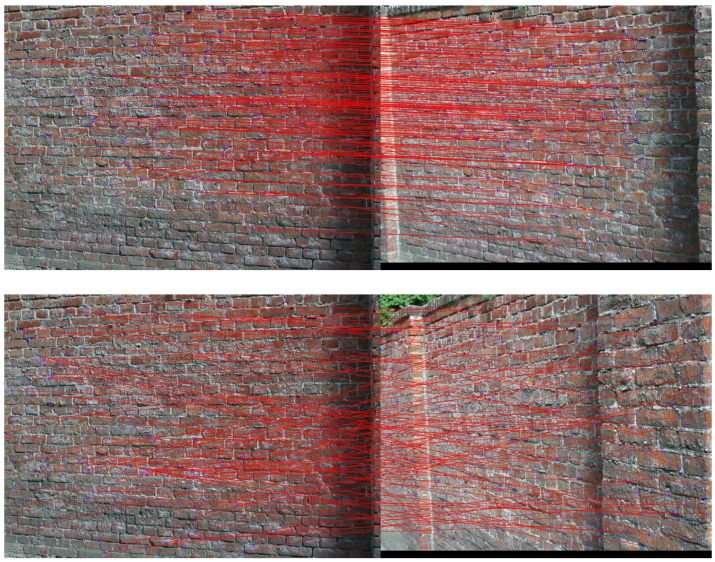

FIGURE 4. Feature Matching between wall1 to wall2 and wall1 to wall2

(b) Affine Transform with SSD matching
First of all, we perform RANSAC methods to find the best affine transformation between matches of two images. To be clear, the green line between features means this matches agree with the affine transformation, while the red line does not agree.
The results show that the matches in bikes' images mostly agree with the derived affine transformation, mainly because there are almost no significant changes but only some blur among images.

FIGURE 5. Matching with affine between bikes1 to bikes2 and to bikes3

FIGURE 6. stitched images

The results shows the matches agree with each other in most cases while comparing graf1 to graf2. But the result is poor while comparing graf1 to graf3 mainly because not only rotation and translation but also scaling and changing of perspective are involved.

FIGURE 7. Matching with affine between graf1 to graf2 and to graf3

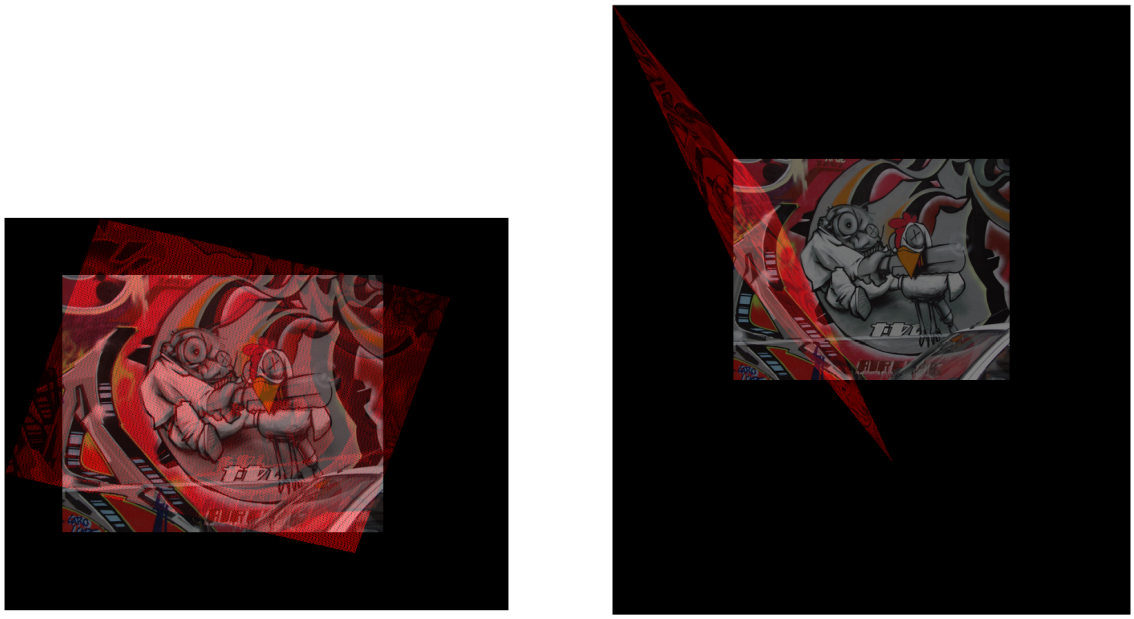

FIGURE 8. stitched images

The results in first pair leuven1 and leuven2 shows that the matches are more agreeable to the derived affine transformation than the second pair between leuven1 and leuven3. The main reason may be that the difference of luminance are greater in second pair which leads to wrong matches.

FIGURE 9. Matching with affine between leuven1 to leuven2 and to leuven3

FIGURE 10. stitched images

The result shows there are many matches in the first pair of images but rarely no matches in the second pair, mainly due to the scaling and changing of perspective.

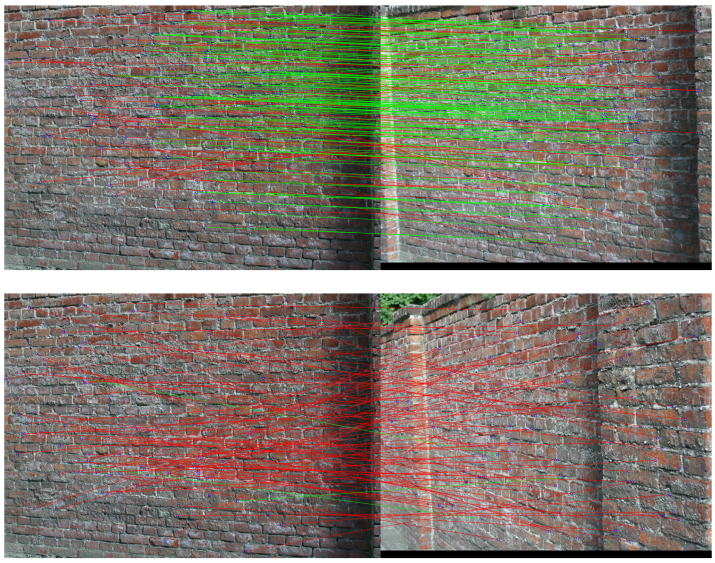

FIGURE 11. Matching with affine between wall1 to wall2 and to wall3

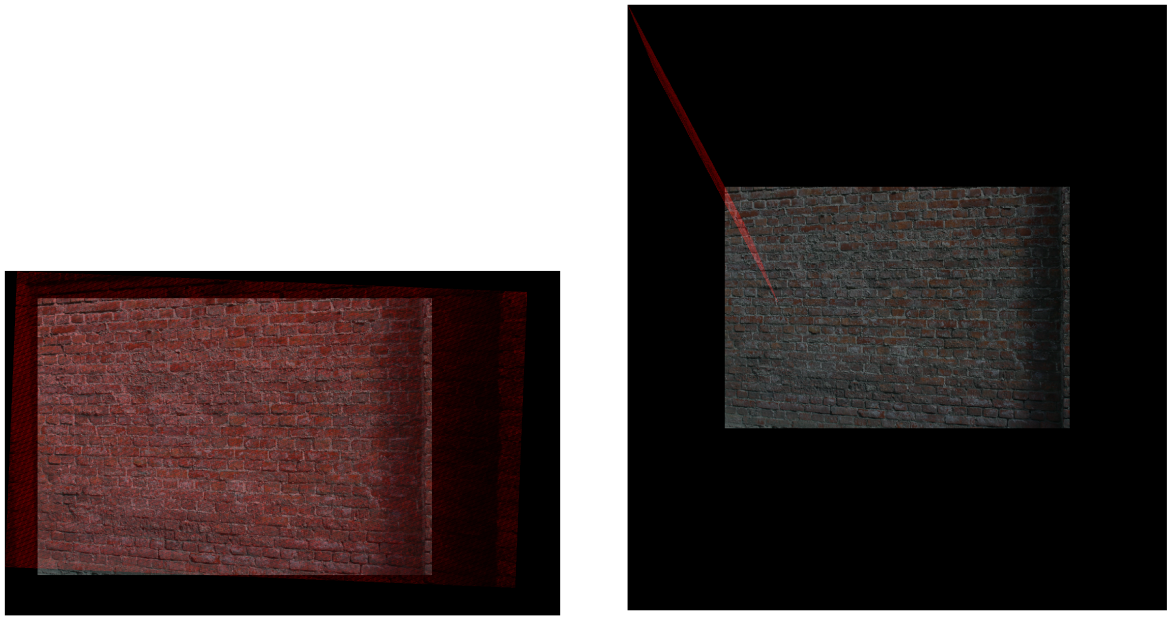

FIGURE 12. stitched images

(c) Projective Transform with SSD matching
The projective transformation are derived from the same matches by performing minimum SSD so that it does not outperform the affine transformation too much. Due to the poor matches in pairs between 'graf's and between 'walls', the stitched images are not acceptable. Also, projective transformation takes scaling in two direction in account, which leads to two consequence: firstly, if the matches are enough and good, the stitched image will be better than affine, however, if the matches are poor, due to two more variable of the matrix, the stitched images are more unstable.

FIGURE 13. Matching with projective between bikes1 to bikes2 and to bikes3

FIGURE 14. stitched images

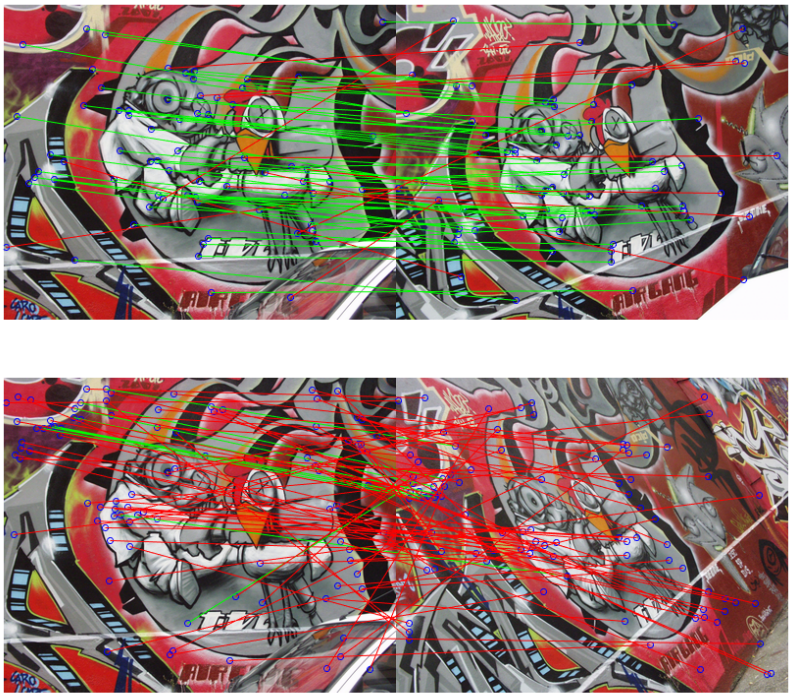

FIGURE 15. Matching with projective between graf1 to graf2 and to graf3

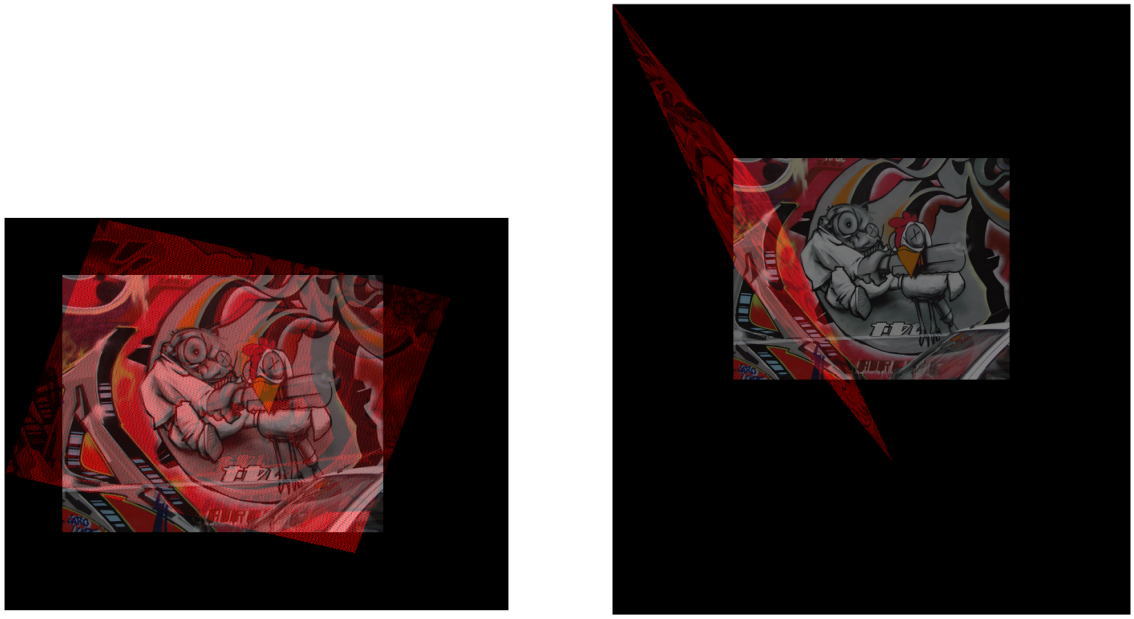

FIGURE 16. stitched images

FIGURE 17. Matching with projective between leuven1 to leuven2 and to leuven3

FIGURE 18. stitched images

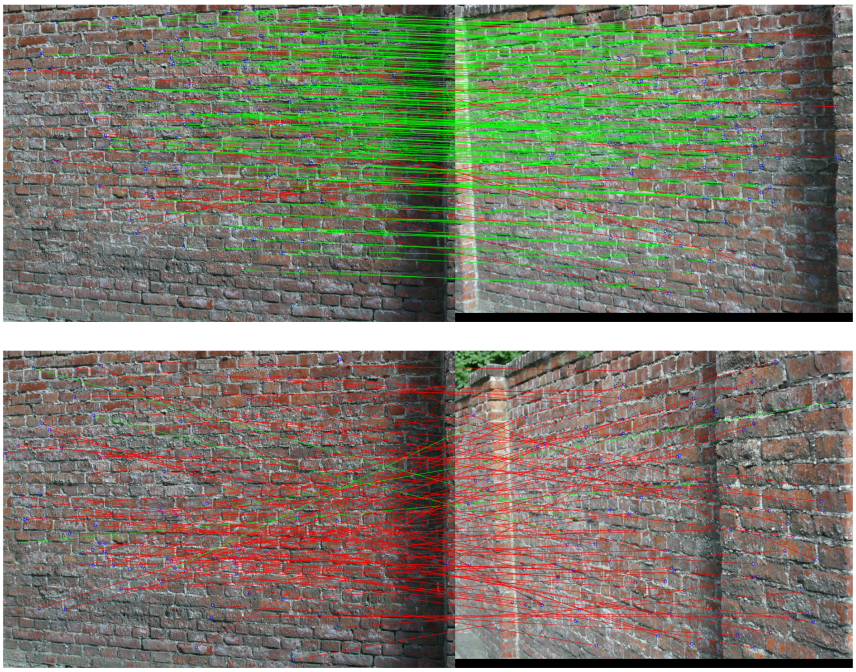

FIGURE 19. Matching with projective between wall1 to wall2 and to wall3

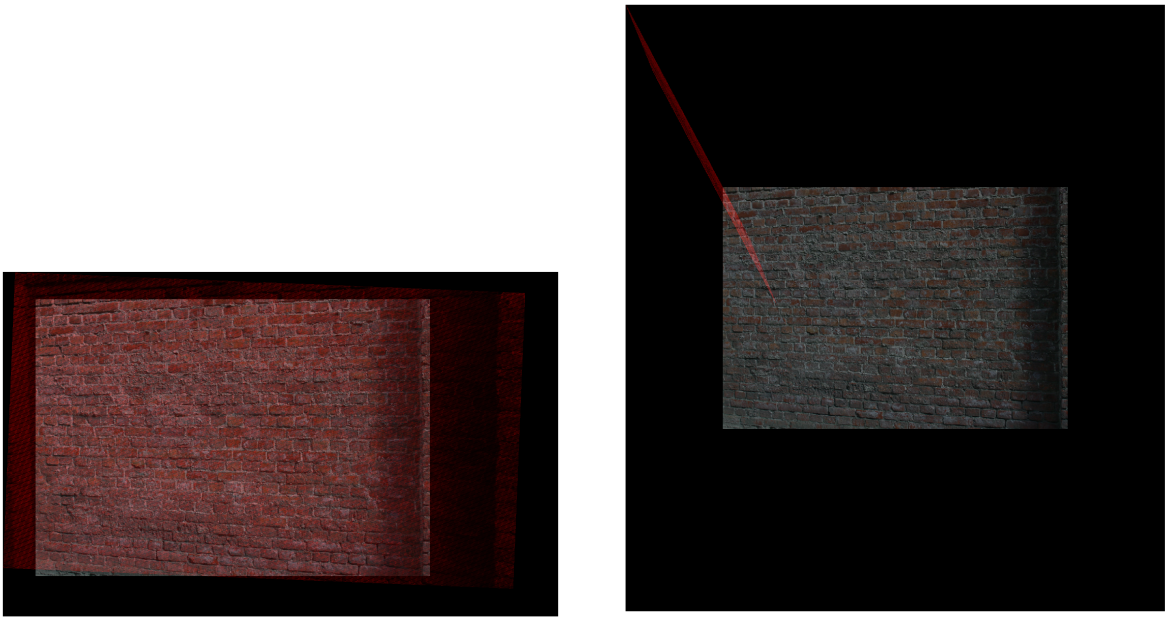

FIGURE 20. stitched images

(d) SIFT Descriptor and Matches with Ratio Test.
The blue circle shows where the feature lies and the green lines connecting features in the image pair means that this match agree with the derived affine transformation.
Comparing to Harris Corner + SSD + affine method, SIFT descriptor performs well in the 'graf' and 'wall' image, but still not perfect, especially when the image are twisted much like graf1 to graf3.
However, from the pair leuven1 to leuven3 we can find that, SIFT did very good to get the correct matches, which means that the difference of luminance does not influence the descriptors a lot.

FIGURE 21. Matching SIFT with affine between bike1 to bikes2 and to bikes3

FIGURE 22. stitched images

FIGURE 23. Matching SIFT with affine between graf1 to graf2 and to graf3

FIGURE 24. stitched images

FIGURE 25. Matching SIFT with affine between leuven1 to leuven2 and to leuven3

FIGURE 26. stitched images

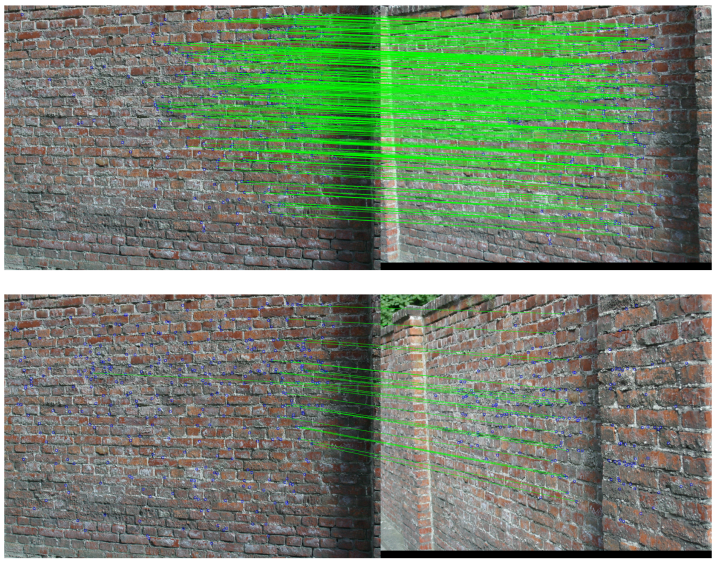

FIGURE 27. Matching SIFT with affine between wall1 to wall2 and to wall3

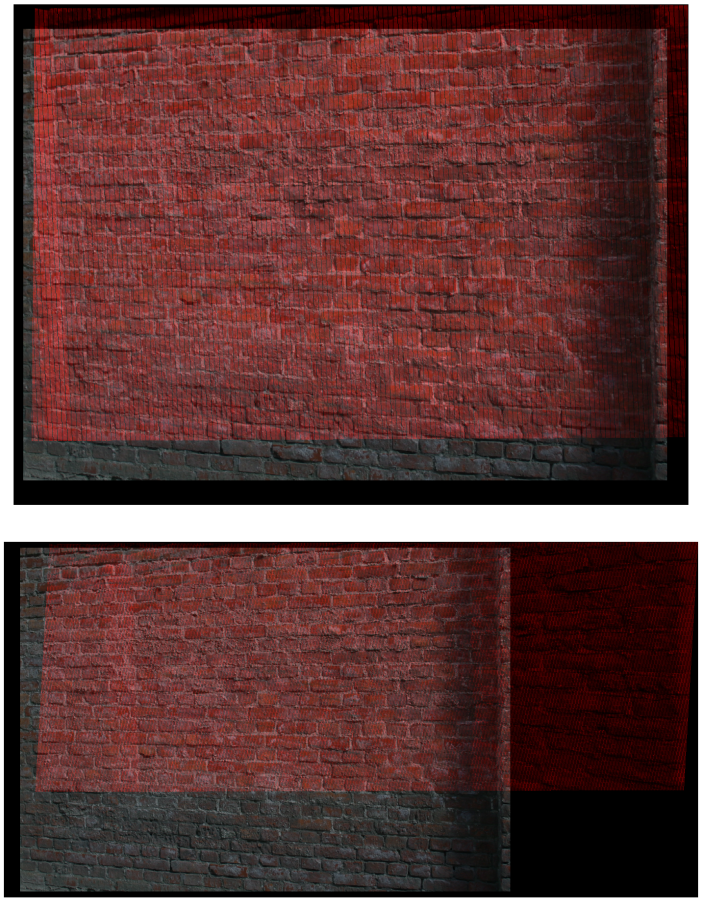

FIGURE 28. stitched images

Driver and Feasible Parameters

```
clear
clc
format compact
% the value following each pair are for detect features threshold
% bikes1 – bikes2 0.99
% bikes1 – bikes3 0.99
% img_1 = imread('bikes3.png');
% img_2 = imread('bikes1.png');
% graf1 – graf2 0.9
% graf1 – graf3 0.3
% img_1 = imread('graf3.png');
% img_2 = imread('graf1.png');
% leuven1 – leuven2 0.95a
% leuven1 – leuven3 0.85
% img_1 = imread('leuven2.png');
% img_2 = imread('leuven1.png');
% wall1 – wall2 0.99
% wall1 – wall3 0.9
% img_1 = imread('wall3.png');
% img_2 = imread('wall1.png');

[rows,cols] = detect_features(img_1); feature_coords1 = [rows,cols];
[rows,cols] = detect_features(img_2); feature_coords2 = [rows,cols];

% matches = match_features(feature_coords1,feature_coords2,img_1,img_2);
% affine_xform = compute_affine_xform(matches,feature_coords1,...
%                                     feature_coords2,img_1,img_2);
% proj_xform = compute_proj_xform(matches,...
%                             feature_coords1,feature_coords2,img_1,img_2);

% uncomment the following commands for sift descriptor, matches and affine
descriptors_1 = ssift_descriptor(feature_coords1,img_1);
descriptors_2 = ssift_descriptor(feature_coords2,img_2);

% the values following each pair are the threshold for ratio test
% bikes1 – bikes2 0.6
% bikes1 – bikes3 0.6
% graf1 – graf2 0.85
% graf1 – graf3 0.95
% leuven1 – leuven2 0.6
% leuven1 – leuven3 0.6
% wall1 – wall2 0.7
% wall1 – wall3 0.95

matches_sift = matches_ratio_test(descriptors_1, descriptors_2);
affine_xform = compute_affine_xform(matches_sift,feature_coords1,...
                                    feature_coords2,img_1,img_2);
```