Homework 4 solution template

CMPSCI 370 Spring 2019, UMass Amherst

Name: DongWon Park

Here is a template that your solutions should roughly follow. Include outputs as figures, and code should be included in the end.

1 Scale Invariant Feature Transform

- What are the advantages of blob detection over the Harris corner detection?

  Blob detection is scale invariant whereas Harris corner detection is sensitive to the image scale. Blob detection tries multiple scales and see its maximum responses to that so it can achieve scale invariant detection.

- How is rotation invariance achieved in SIFT features?

  SIFT features creates histogram of local gradient directions in the patch. And then it assigns canonical orientation at peak of smoothed histogram. That canonical orientation is the direction where the histogram shows maximum direction. So by doing this, SIFT is rotation invariant even though images are rotated.

- When can matching patches using sum-of-squared-differences between the vector of pixel values fail? List two scenarios when the pixel values can change significantly.

  Because sum-of-squared-difference depends on each pixel values, it is very sensitive to the change of pixel values. For example, if there is a small intensity changes in a pixel, then the result will be very different by that small intensity change even though that feature is almost same. Or if there is a small image deformation, also the results varies by that small deformation.

- List two ways how the SIFT descriptor provides robustness during feature matching?

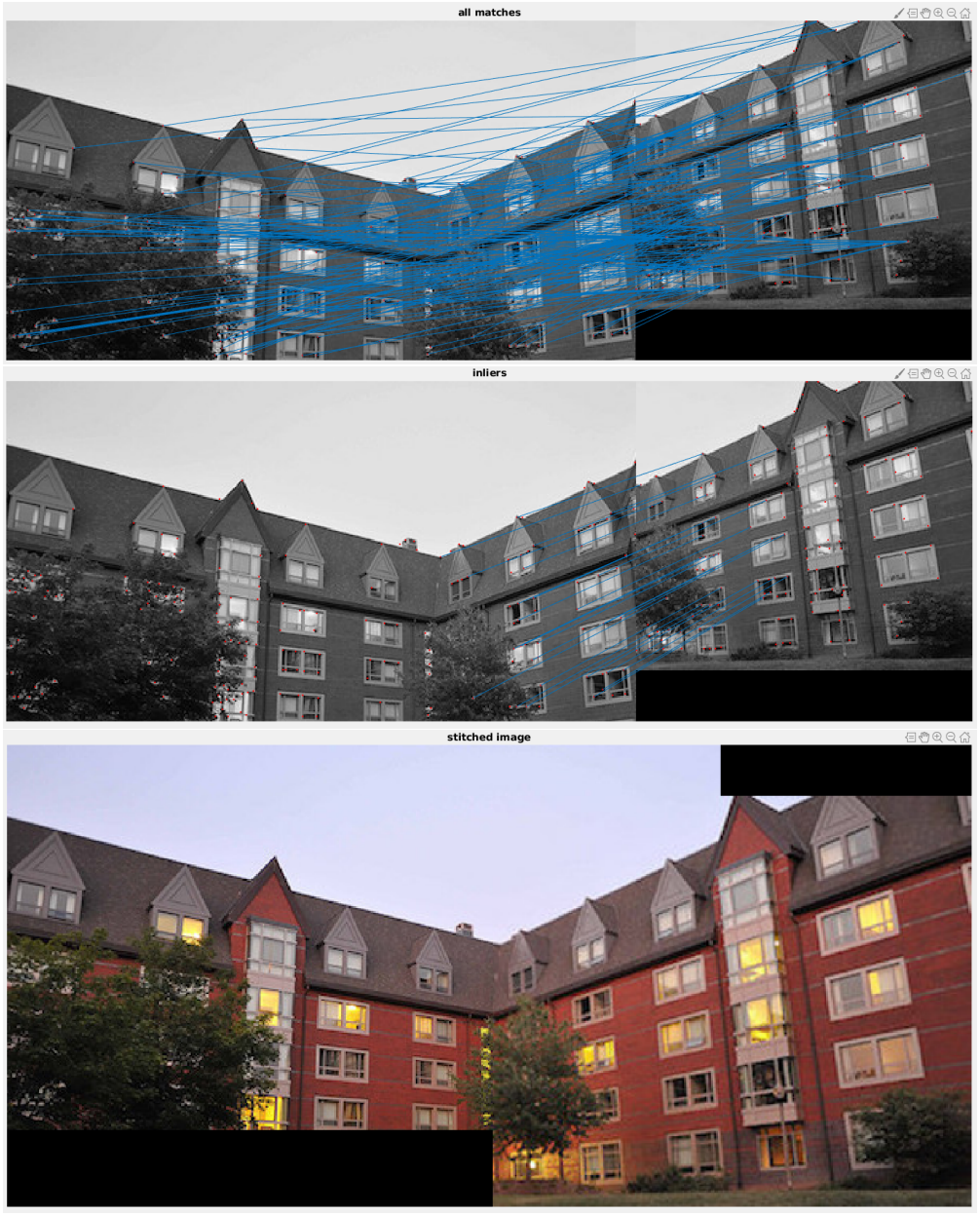
  SIFT descriptor detects features with characteristic scales and orientations. By detecting features with characteristics scales and orientations, it is invariant to the intensity changes and small deformations so it can provides robustness during feature matching.

2 Image Stitching

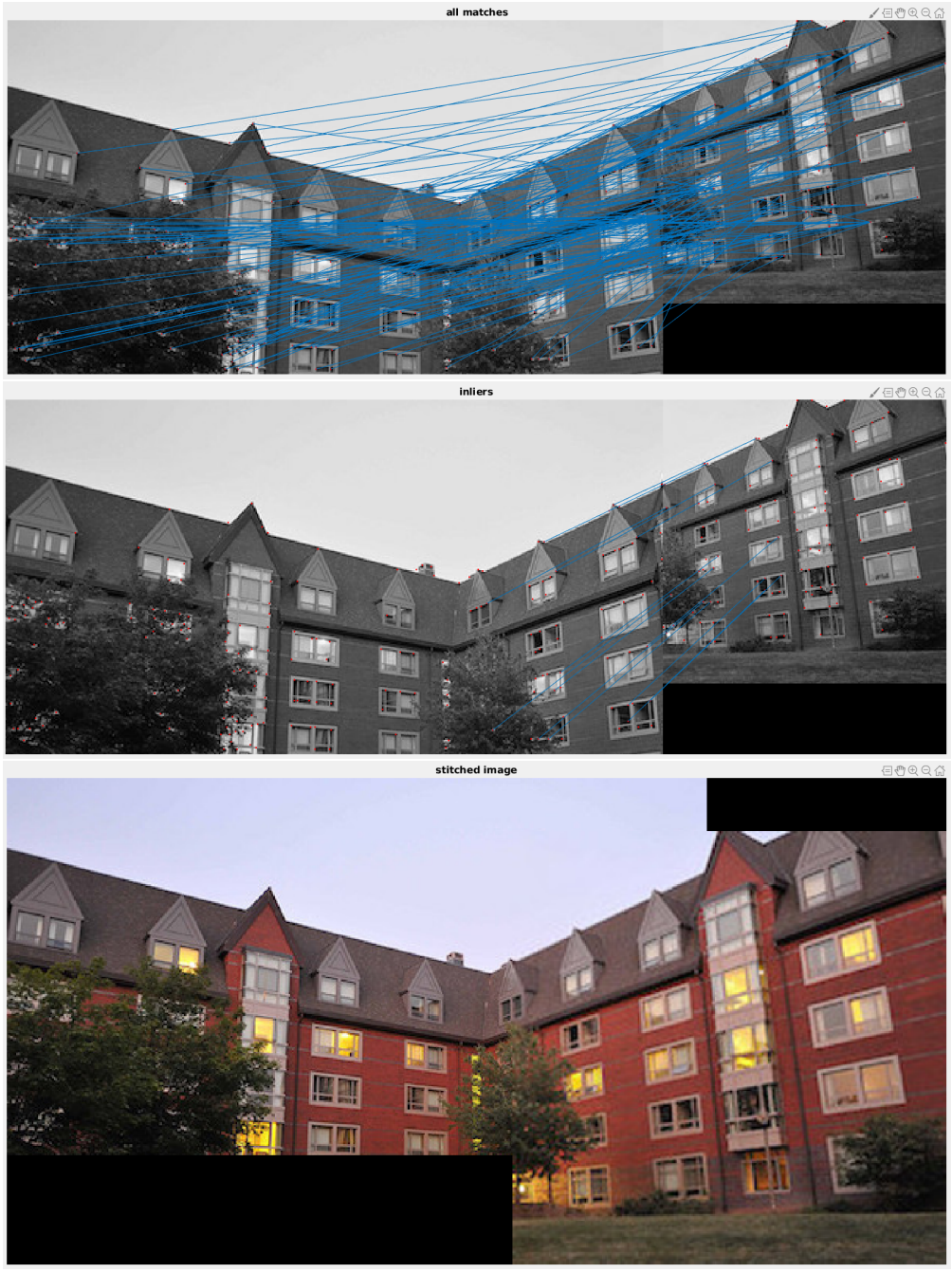
- Outputs for umass_building_right1.jpg

- Outputs for umass_building_right2.jpg

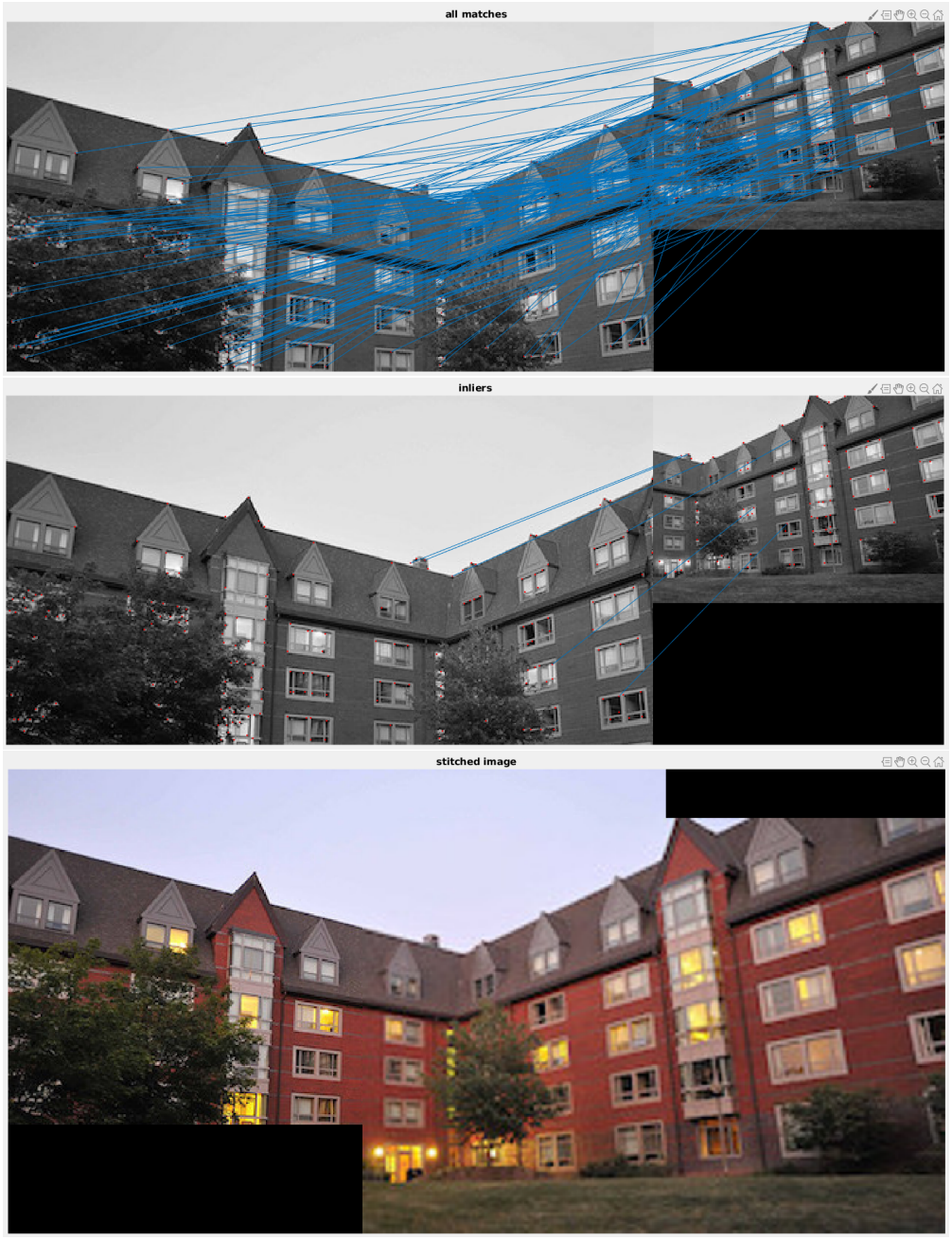
- Outputs for umass_building_right3.jpg



- Outputs for umass_building_right4.jpg



- Outputs for umass_building_right5.jpg



- Estimated transformations and number of inliers

im2	tx	ty	s	#inliers
umass_building_right1.jpg	-240	-32	1.0051	62
umass_building_right2.jpg	-252	-35	0.8998	50
umass_building_right3.jpg	-304	-32	0.7980	26
umass_building_right4.jpg	-323	-34	0.6989	14
umass_building_right5.jpg	-241	-33	0.5081	6

Table 1: Estimated transformation.

3 Extra Credit: Affine Transformation

To get the affine transformation we can use this equation:

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

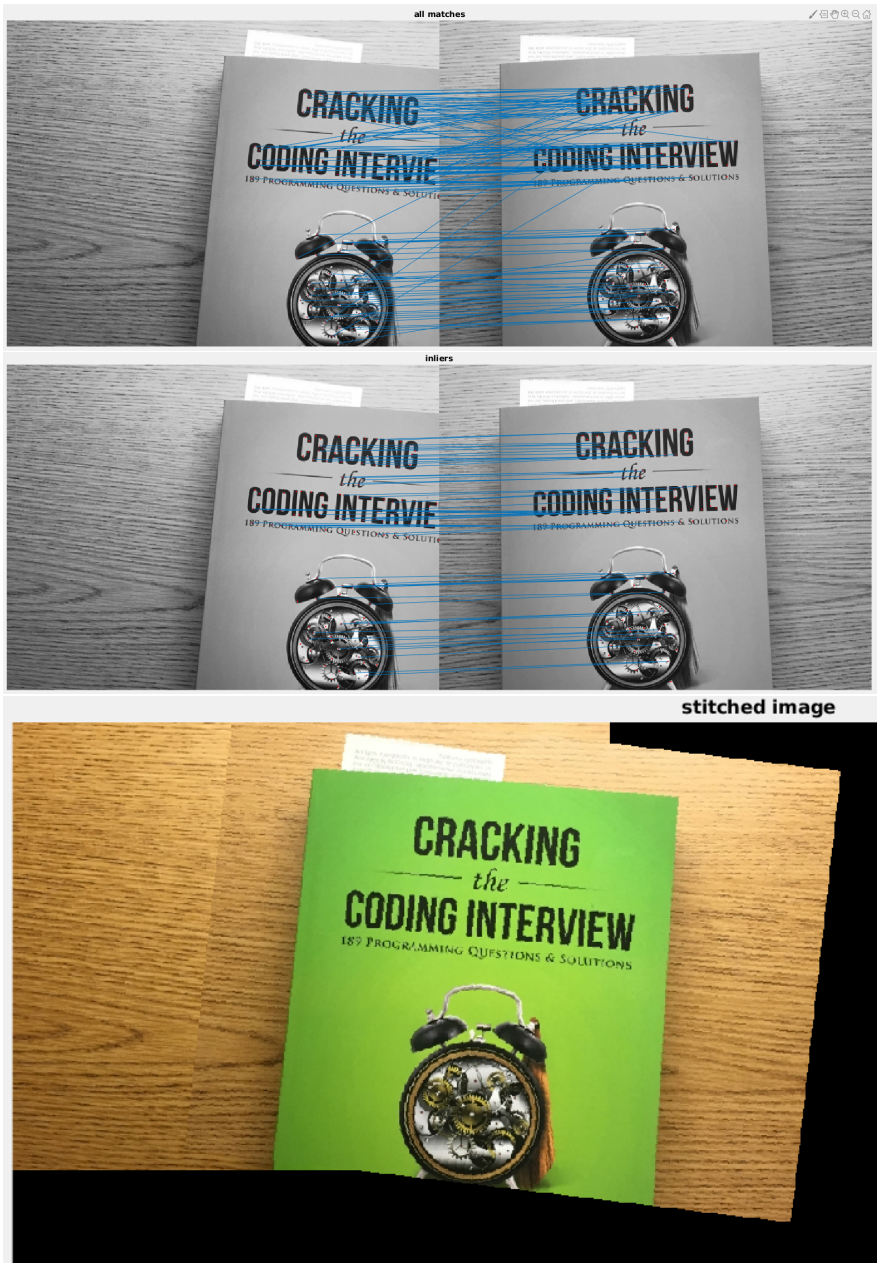
And, to get six unknowns m1, m2, m3, m4, t1 and t2, we need at least three pairs. Then we can derive this equation from above:

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ x_3 & y_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_3 & y_3 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t1 \\ t2 \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \end{bmatrix}$$

To get the six unknowns parameter matrix we do inverse operation:

$$\begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t1 \\ t2 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ x_3 & y_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_3 & y_3 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \end{bmatrix}$$

Outputs for extracredit.jpg



The parameter of this transformation was:

$$\begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \\ t_1 & t_2 \end{bmatrix} = \begin{bmatrix} 0.9707 & 0.1074 \\ -0.1175 & 0.9774 \\ -213.7062 & 50.0161 \end{bmatrix}$$

#inliers = 59
And, that matrix M shows that this was an "Arbitrary linear transformation", not a combination of uniform scaling + translation that we did for this homework.

4 Solution code

Include the source code for your solutions as seen below (only the files you implemented are necessary). In latex the command verbatiminput{alignChannels.m} allows you to include the code verbatim as seen below. Regardless of how you do this the main requirement is that the included code is readable (use proper formatting, variable names, etc.) A screenshot of your code works to provided you include a link to source files.

4.a extractFeatures.m

```
function f = extractFeatures(im, c, patchRadius)
% This code is part of:
%
%   CMPSCI 370: Computer Vision, Spring 2018
%   University of Massachusetts, Amherst
%   Instructor: Subhransu Maji
%
%   Homework 4

% Convert color image to gray
if size(im, 3) > 1
    im = rgb2gray(im);
end

% Pad image by radius patchRadius pixels
im = padarray(im, [patchRadius patchRadius], 'replicate', 'both');
c(1,:) = patchRadius + c(1,:); % move x axis by patchRadius
c(2,:) = patchRadius + c(2,:); % move y axis by patchRadius

% Create feature container
f = zeros([(2*patchRadius+1)^2 size(c,2)]);

% Extract features on each corners
for i = 1:size(c, 2)
    cornerX = c(1, i);
    cornerY = c(2, i);
    feature = im(cornerY-patchRadius: cornerY+patchRadius, ...
                 cornerX-patchRadius: cornerX+patchRadius);
    feature = feature(:);
    f(:,i) = feature;
end
```

4.b computeMatches.m

```
function m = computeMatches(f1,f2)
% This code is part of:
%
%   CMPSCI 370: Computer Vision, Spring 2018
%   University of Massachusetts, Amherst
%   Instructor: Subhransu Maji
%
%   Homework 4

% Create matcher container m
```

```
m = zeros(size(f1, 2), 1);

% Compute matches using Sum of Square Difference (ssd)
for i = 1 : size(f1, 2)
    % minimum ssd and its index
    ssd_min = inf;
    ind_min = 0;

    % find matches of f2 for a feature of f1
    for j = 1 : size(f2, 2)
        diff = f1(:,i) - f2(:,j);   % difference
        ssd = sum(diff(:).^2);      % ssd
        if ssd < ssd_min        % if this ssd is less than previous
            ssd_min = ssd;      % record ssd
            ind_min = j;        % record its index
        end
    end
    m(i) = ind_min;     % record the index to matches
end
```

4.c ransac.m

```
function [inliers, transf] = ransac(matches, c1, c2)
% This code is part of:
%
%   CMPSCI 370: Computer Vision, Spring 2018
%   University of Massachusetts, Amherst
%   Instructor: Subhransu Maji
%
%   Homework 4

max_num_inliers = 0;    % record max num of item in inlier
best_tx = 0;            % record best tx
best_ty = 0;            % record best ty
best_scale = 0;         % record best scale
best_inliers = [];      % record best inliers

N = 10000;              % number of iterations for RANSAC
threshold = 100;        % THRESHOLD

for n = 1:N
    % Select two random indices i and j
    i = randi(size(matches,1)); % random index i
    % if matched index to i is zero, then find another
    while matches(i) == 0
        i = randi(size(matches, 1));
    end
    j = randi(size(matches,1)); % random index j
    while i == j || matches(j) == 0
        j = randi(size(matches,1));
    end

    % Sample two random pairs from im1 and im2
    rand_x1 = c1(1,i);              % x1 (original)
```

```
    rand_x1p = c2(1,matches(i));    % x1' (transformed)
    rand_y1 = c1(2,i);
    rand_y1p = c2(2,matches(i));
    rand_x2 = c1(1,j);
    rand_x2p = c2(1,matches(j));
    rand_y2 = c1(2,j);
    rand_y2p = c2(2,matches(j));

    % Get transformation T(scale, tx, ty)
    scale = sqrt( (rand_x1p-rand_x2p)^2 + (rand_y1p-rand_y2p)^2 ) / ...
            sqrt((rand_x1-rand_x2)^2 + (rand_y1-rand_y2)^2);
    tx = (rand_x1p+rand_x2p-scale*rand_x1-scale*rand_x2) / 2.0;
    ty = (rand_y1p+rand_y2p-scale*rand_y1-scale*rand_y2) / 2.0;

    % Calculate every distances using the above transformation
    inliers = [];                   % container to record inliers
    for k = 1:size(matches,1)
        % get a point of corner1 and its matched corner 2
        x1 = c1(1,k);               % get x
        x1p = c2(1,matches(k));     % get matched x
        y1 = c1(2,k);               % get y
        y1p = c2(2,matches(k));     % get matched y

        % Map a predicted transformation point of im2 to im1 using transformation T
        Tx1p = (x1p-tx)/scale;      % get transformed x of x1p
        Ty1p = (y1p-ty)/scale;      % get transformed y of y1p
        d = (x1 - Tx1p)^2 + (y1 - Ty1p)^2; % get distance

        % if the distance d is less than THRESHOLD, put into inliers
        if d < threshold
            inliers = [inliers k];
        end
    end

    % if number of inliers is greater than previous one,
    % then, accept this new inliers as the best inliers
    % also record best transformation - tx, ty, scale
    if numel(inliers) > max_num_inliers
        max_num_inliers = numel(inliers);
        best_inliers = inliers;
        best_tx = tx;
        best_ty = ty;
        best_scale = scale;
    end

end

transf = [best_tx, best_ty, best_scale];
inliers = best_inliers;
```

5 Extra credit: Affine Transformation Solution Code

5.a ransac_affine.m

```
function [inliers, transf] = ransac_affine(matches, c1, c2)
% This code is part of:
%
%   CMPSCI 370: Computer Vision, Spring 2018
%   University of Massachusetts, Amherst
%   Instructor: Subhransu Maji
%
%   Homework 4

max_num_inliers = 0;    % record max num of item in inlier
best_affine_M = [];     % record best affine matrix M
best_translate_M = [];  % record best translate matrix
best_inliers = [];      % record best inliers

N = 10000;      % number of iterations for RANSAC
threshold = 1;  % THRESHOLD

for n = 1:N
    % Select three random indices i, j and h
    i = randi(size(matches,1)); % random index i
    % if matched index to i is zero, then find another
    while matches(i) == 0
        i = randi(size(matches, 1));
    end
    j = randi(size(matches,1)); % random index j
    while i == j || matches(j) == 0
        j = randi(size(matches,1));
    end
    h = randi(size(matches,1)); % random index h
    while h == i || h == j || matches(h) == 0
        h = randi(size(matches,1));
    end

    % Sample three random pairs from im1 and im2
    x1 = c1(1,i);               % x1 (original)
    x1p = c2(1,matches(i));     % x1' (transformed)
    y1 = c1(2,i);
    y1p = c2(2,matches(i));
    x2 = c1(1,j);
    x2p = c2(1,matches(j));
    y2 = c1(2,j);
    y2p = c2(2,matches(j));
    x3 = c1(1,h);
    x3p = c2(1,matches(h));
    y3 = c1(2,h);
    y3p = c2(2,matches(h));

    % matrix of random points
    randpoints_M = [x1 y1 0 0 1 0;
                    0 0 x1 y1 0 1;
```

```
                    x2 y2 0 0 1 0;
                    0 0 x2 y2 0 1;
                    x3 y3 0 0 1 0;
                    0 0 x3 y3 0 1;];
    % matrix of transformed points
    transformed_M = [x1p;
                     y1p;
                     x2p;
                     y2p;
                     x3p;
                     y3p;];
    % matrix of parameters
    parameter_M = inv(randpoints_M) * transformed_M;

    % Affine matrix and Translate matrix
    affine_M = [parameter_M(1) parameter_M(2);
                parameter_M(3) parameter_M(4);];
    translate_M = [parameter_M(5);
                   parameter_M(6);];

    % Calculate every distances using the above transformation
    inliers = [];  % container to record inliers
    for k = 1:size(matches,1)
        % get a point of corner1 and its matched corner 2
        x1 = c1(1,k);           % get x
        x1p = c2(1,matches(k)); % get matched x
        y1 = c1(2,k);           % get y
        y1p = c2(2,matches(k)); % get matched y

        % Map a predicted transformation point of im2 to im1 using transformation T
        mapped_M = affine_M * [x1; y1] + translate_M;
        Tx1p = mapped_M(1);
        Ty1p = mapped_M(2);
        d = (x1p - Tx1p)^2 + (y1p - Ty1p)^2; % get distance

        % if the distance d is less than THRESHOLD, put into inliers
        if d < threshold
            inliers = [inliers k];
        end
    end

    % if number of inliers is greater than previous one,
    % then, accept this new inliers as the best inliers
    % also record best transformation - tx, ty, scale
    if numel(inliers) > max_num_inliers
        max_num_inliers = numel(inliers);
        best_inliers = inliers;
        best_affine_M = affine_M;
        best_translate_M = translate_M;
    end

end

transf = [best_affine_M; best_translate_M';];
```

```
inliers = best_inliers;
```

5.b mergeImages_affine.m

```
function im = mergeImages_affine(im1, im2, transf)
% This code is part of:
%
%   CMPSCI 370: Computer Vision, Spring 2018
%   University of Massachusetts, Amherst
%   Instructor: Subhransu Maji
%
%   Homework 4

% Create bigger canvas to merge two images
xmax = uint16(size(im1, 2)*2.5);
ymax = uint16(size(im1, 1)*2.5);
canvas = zeros(ymax, xmax, size(im1,3), 'uint8');

% Get information of affine transformation and translation
affine_M = transf(1:2, 1:2); % 2x2 matrix
translate_M = round(transf(3, :))'; % 2x1 matrix

% Put im1 into the canvas
canvas(1:size(im1, 1), 1:size(im1,2), :) = im1;

% Loop over pixels in canvas, and see if im2 maps into them
for y=1:size(canvas, 1)
    for x=1:size(canvas, 2)
        % calculate transformation
        T_points = affine_M * [x; y] + translate_M;
        xp = round(T_points(1));
        yp = round(T_points(2));
        % copy into canvas if in range of im2
        if xp > 0 && xp < size(im2, 2) && yp > 0 && yp < size(im2, 1)
            canvas(y, x, :) = im2(uint16(yp), uint16(xp), :);
        end
    end
end

im = canvas;
```