Computer Vision – Programming Project 3

王順興 0210184

Structure:

1. SurfHomography.m: The main program, contains RANSAC process.
2. surfFindMatchPoints.m: Feature point detection /extraction/matching
3. findHomography.m: Non-linear homography finding

Files for testing are omitted.

Full project: https://github.com/vicodin1123/CV-project/tree/master/p3

SurfHomography.m

Read and resize the images

```
disp('Read images...');

% Target
Img1 = rgb2gray(imread('./data/cover.jpg'));
Img1 = imresize(Img1, 0.5);

% Photo
Img2 = rgb2gray(imread('./data/b2.jpg'));
Img2 = imresize(Img2, 0.5);

% Pattern: pattern contains the image;
%   alpha contains the transparent information (a mask)
[pattern,map,alpha] = imread('./data/s2.png');
[pattern map] = imresize(pattern, 0.5);
alpha = imresize(alpha, 0.5);
```

Find matched SURF feature points

```
disp('Find SURF matching points...');
[T, W] = surfFindMatchPoints(Img1, Img2);

NumOfMPs = size(W, 1);

disp(sprintf('Num of MPs: %d', NumOfMPs));
```

RANSAC

```
disp('RANSAC...');
HomographyIterations = 300; % # of maximum iterations for non-linear optimization
RANSACiteration = min(max(500, NumOfMPs*10), 1000); % # of maximum iterations for
RANSAC
InlierThreshold = 5;    % Thershold for projection error in pixels

maxInliers = zeros(1,1);    % Store the largest set of inliers
maxInlierCount = -1;

for i = 1 : RANSACiteration
% Randomly pick four points
Indices = randperm(NumOfMPs, 4);
WorldCoord = W(Indices, :);
TargetCoord = T(Indices, :);

% Find homography using the four points
phi = findHomography(WorldCoord, TargetCoord, HomographyIterations);
H = double(reshape(phi, [3 3]));

% Find point-wise error - psi
exW = [W ones(NumOfMPs, 1)];
H = [phi(1:8);1];
denom = exW*H(7:9);
x = exW*H(1:3)./denom;
y = exW*H(4:6)./denom;
X = [x y];
psi = T - X;

% Find squared root error for each point
sqE = sqrt(psi(:, 1).^2 + psi(:, 2).^2);

% Count the number of inliers
Inliers = find(sqE<InlierThreshold);
InlierCount = numel(Inliers);
```

```
% If a better set of inliers is found, store it
if InlierCount > maxInlierCount
          maxInleirs = Inliers;
    maxInlierCount = InlierCount;
    disp(sprintf('Itr: %d InlierCount: %d', i, maxInlierCount));

    if double(maxInlierCount)/NumOfMPs >= 0.8
        break;
    end
end
end
```

Use the best set of inliers to find homography

```
disp('Find the final homography...');
disp(sprintf('NumOfMPs: %d Inliers: %d', NumOfMPs, maxInlierCount));

WorldCoord = W(maxInleirs, :);
TargetCoord = T(maxInleirs, :);

phi = findHomography(WorldCoord, TargetCoord, HomographyIterations);

% Here it is
H = double(reshape(phi, [3 3]));
```

Append the pattern onto the image

```
figure1 = figure;



% Two images to put = Two axes to put on
ax1 = axes('Parent',figure1);
ax2 = axes('Parent',figure1);
set(ax1,'Visible','off');
set(ax2,'Visible','off');
```

```
% Project the pattern onto the photo using H^-1
% -Since some pixel will be projected out of the region of original image,
% Matlab projects that image and added translation to contain the result.
% RNex stores the translation on X and Y
RNe = imref2d(size(pattern));
t = projective2d(inv(H));
[pattern RNex] = imwarp(pattern, RNe, t);
alpha = imwarp(alpha, RNe, t);

% Use the translation parameter X and Y stored in RNex to make sure
% that pattern and photo is in the same coordinate system
xa = uint8(zeros([size(Img2) 3]));
xa( floor(RNex.YWorldLimits(1)):floor(RNex.YWorldLimits(1)) + size(pattern, 1) -
1,...
    floor(RNex.XWorldLimits(1)):floor(RNex.XWorldLimits(1)) + size(pattern, 2) -
1, :)...
    = pattern;
Rxa = imref2d(size(xa));

% The transparent mask requires the same treatment
xalpha = uint8(zeros(size(Img2)));
xalpha( floor(RNex.YWorldLimits(1)):floor(RNex.YWorldLimits(1)) + size(pattern, 1)
- 1,...
    floor(RNex.XWorldLimits(1)):floor(RNex.XWorldLimits(1)) + size(pattern, 2) -
1)...
    = alpha;

% Draw the photo
imshow(Img2, 'Parent',ax1);

% Draw the pattern and let it have a transparent background
I = imshow(xa, 'Parent',ax2);
set(I,'AlphaData',xalpha);
```

findHomography.m

```
function phi = findHomography(W, T, NumOfIterations)
% Find Corresponding Points Using SURF Features
% INPT: W: Nx2 matrix. The [x y] coordinates of world FPs
%       T: Nx2 matrix. The [x y] coordinates of target FPs
%       NumOfIterations: The # of maximum iterations for non-linear optimization
% OUPT: phi: 1x9 vector. The vectorized homography parameters

NumOfMPs = size(W, 1);

% Append 1 for homogenous coordinate
W = [W ones(NumOfMPs, 1)];
T = [T ones(NumOfMPs, 1)];

A = zeros(2*NumOfMPs, 9);
for i = 1 : NumOfMPs
    w = W(i, :);
    x = T(i, :);
    a = [0 0 0 -w x(2)*w; ...
         w 0 0 0 -x(1)*w];
    A((i-1)*2+1:i*2, :) = a;
end
[U,S,V] = svd(A);
phi = V(:, 9);

% Reparameterize Phi
phi(1) = phi(1) + 1;
phi(5) = phi(5) + 1;
phi = phi(1:8);
exphi = [phi(1:8);1];

T = T(:, 1:2);
X = zeros(NumOfMPs, 2);

for iteration = 1 : NumOfIterations
```

```
% Find Psi
exphi = [phi(1:8);1];
denom = W*exphi(7:9);
x = W*exphi(1:3)./denom;
y = W*exphi(4:6)./denom;
X = [x y];
psi = T - X;

% Construct J
J = zeros(2*NumOfMPs, 8);
for i = 1 : NumOfMPs
    w = W(i, :);
    x = T(i, :);
    j = [w 0 0 0 -x(1)*w(1) -x(1)*w(2); ...
         0 0 0 w -x(2)*w(1) -x(2)*w(2)];
    J((i-1)*2+1:i*2, :) = j./(w*exphi(7:9));
end

% Find A and b
A = zeros(8, 8);
b = zeros(8, 1);
for i = 1 : NumOfMPs
    j = J((i-1)*2+1:i*2, :);
    A = A + j'*j;
    b = b + j'*psi(i, :)';
end

if numel(find(isinf(A))) ~= 0 || numel(find(isnan(A)))
    break;
end

% Find gradient
dPhi = pinv(A)*b;
phi = phi + dPhi;
```

```
% Stop when the projection error is 0
AvgError = (sum(abs(psi(:, 1))) + sum(abs(psi(:, 2))))/2/size(psi, 1);
if AvgError == 0
```

```
    break;
end

end
phi = [phi(1:8);1];
end
```

surfFindMatchPoints.m

```
function [matchedPoints1 matchedPoints2] = surfFindMatchPoints(Img1, Img2)
% Find Corresponding Points Using SURF Features
% INPT: Img1, Img2: grayscale image of any size
% OUPT: matchedPoints1, matchedPoints2:
%       Nx2 matrix. [x y] coordinates of matched N FPs corresponding to Img1,Img2

% Find the SURF features
points1 = detectSURFFeatures(Img1);
points2 = detectSURFFeatures(Img2);

% Extract the features
[f1,vpts1] = extractFeatures(Img1,points1);
[f2,vpts2] = extractFeatures(Img2,points2);

% Retrieve the locations of matched points
indexPairs = matchFeatures(f1,f2) ;
matchedPoints1 = vpts1(indexPairs(:,1));
matchedPoints2 = vpts2(indexPairs(:,2));

% Remove points that are too close (mainly duplicates)
radius = 1;
p1BadIndices = zeros(size(matchedPoints1));
for i = 1 : size(matchedPoints1, 1)
    pts = matchedPoints1.Location(i, :);
    xDupe = abs(pts(1)-matchedPoints1.Location(:, 1)) < radius;
    yDupe = abs(pts(2)-matchedPoints1.Location(:, 2)) < radius;
    xDupe(i) = 0; yDupe(i) = 0;
    p1BadIndices = p1BadIndices | (xDupe&yDupe);
end
```

```
p2BadIndices = zeros(size(matchedPoints2));
for i = 1 : size(matchedPoints2, 1)
    pts = matchedPoints2.Location(i, :);
    xDupe = abs(pts(1)-matchedPoints2.Location(:, 1)) < radius;
    yDupe = abs(pts(2)-matchedPoints2.Location(:, 2)) < radius;
    xDupe(i) = 0; yDupe(i) = 0;
    p2BadIndices = p2BadIndices | (xDupe&yDupe);
end

GoodIndices = ~(p1BadIndices | p2BadIndices);
matchedPoints1 = matchedPoints1(GoodIndices);
matchedPoints2 = matchedPoints2(GoodIndices);

% Return the coordinate
matchedPoints1 = matchedPoints1.Location;
matchedPoints2 = matchedPoints2.Location;

end
```