ECE 661 Fall 2018
Homework #5

Ye Shi
shi349@purdue.edu

October 11, 2018

Contents

1 Methodology

In the homework, I select the SURF algorithm to find and match the interest points. The LSM in Sec. 1.1 is applied in RANSAC, Sec. 1.2, to find inliners. Levenberg-Marquardt Algorithm, Sec. 1.3, is applied to find homography by the inliners. For Monsaicing image, please see Sec. 1.4.

1.1 Least Square Method (LSM) by Singular Value Decomposition (SVD)

Assuming $x' = Hx$ that x and x' are expressed using HC:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ and } x' = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix}$$

with

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}.$$

We end up with

$$\begin{cases} x'_1 = h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \\ x'_2 = h_{21}x_1 + h_{22}x_2 + h_{23}x_3 \\ x'_3 = h_{31}x_1 + h_{32}x_2 + h_{33}x_3 \end{cases}$$

Since x' is in HC, the 2D physical coordinates (x', y') have

$$\begin{cases} x' = \dfrac{x'_1}{x'_3} = \dfrac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3} \\ y' = \dfrac{x'_2}{x'_3} = \dfrac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3} \end{cases}.$$

Since there is no ideal points in real image, $x' \neq 0$, we have

$$\begin{cases} h_{11}x_1x'_3 + h_{12}x_2x'_3 + h_{13}x_3x'_3 - h_{31}x_1x'_1 - h_{32}x_2x'_1 - h_{33}x_3x'_1 = 0 \\ h_{21}x_1x'_3 + h_{22}x_2x'_3 + h_{23}x_3x'_3 - h_{31}x_1x'_2 - h_{32}x_2x'_2 - h_{33}x_3x'_2 = 0 \end{cases}.$$

While we select the points on real images, $x_3 = 1$ and $x'_3 = 1$. And H in HC, we can assign $h_33 = 1$. We then have

$$\begin{cases} h_{11}x_1 + h_{12}x_2 + h_{13} - h_{31}x_1x'_1 - h_{32}x_2x'_1 - x'_1 = 0 \\ h_{21}x_1 + h_{22}x_2 + h_{23} - h_{31}x_1x'_2 - h_{32}x_2x'_2 - x'_2 = 0 \end{cases}.$$

We then formulate it into a matrix expression with a vector

$$\vec{H} = [h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33}]^T$$

$$A\vec{H} = 0, \tag{1}$$

where

$${}^{2i\times 9}A = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & 1 & 0 & 0 & 0 & -x_1^{(1)}x_1'^{(1)} & -x_2^{(1)}x_1'^{(1)} & -x_1'^{(1)} \\ 0 & 0 & 0 & x_1^{(1)} & x_2^{(1)} & 1 & -x_1^{(1)}x_2'^{(1)} & -x_2^{(1)} & -x_2'^{(1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_1^{(i)} & x_2^{(i)} & 1 & 0 & 0 & 0 & -x_1^{(i)}x_1'^{(i)} & -x_2^{(i)}x_1'^{(i)} & -x_1'^{(i)} \\ 0 & 0 & 0 & x_1^{(i)} & x_2^{(i)} & 1 & -x_1^{(i)}x_2'^{(i)} & -x_2^{(i)}x_2'^{(i)} & -x_2'^{(i)} \end{bmatrix}$$

We can solve this by SVD, which gives $A = UDV^H$. Please note, the number i of corresponding pairs needs to be greater than 5 in order to satisfy the rank of A is greater than 9, otherwise we cannot apply SVD on A. In my application, I set $i = 9$. The last column of V is $\vec{H}$, which is also corresponding to the minimum value of D, which is actually the minimal variance. We then reshape $\vec{H}$ to the matrix H.

1.2 Random Sample Consensus (RANSAC)

RANSAC is an algorithm based on probability to remove the bad samples. In our homework, it removes the outliers matched by SURF features. There are some parameters listed as follows,

- δ - decision threshold. If the distance d between the estimated point $\tilde{x'}$ and the real target point x' greater than $delta$, we will regard it as a outlier. I set $\delta = 20$ in chess board distance.
- ϵ - the proportion of outliers to all pairs. I set $\epsilon = 0.2$.
- p - the probability that at least one of the N trials will be free of outliers. I set $p = 0.99$.
- N - number of trails,
$$N = \frac{\ln(1-p)}{\ln(1-(1-\epsilon)^m)}$$
- m - the set of correspondences for calculating homography matrix H in each trail. I set $m = 9$, which is actually the i in Sec. 1.1.
- M - a minimum value for the size of the inlier set for it to be acceptable,
$$M = (1-\epsilon)n_{total}$$
- n_{total} - total number of SURF matched pairs.

The RANSAC is explained as following steps.

1. Randomly sample m corresponding pairs, then use LSM to find the homography.
2. Transform each point x_i by the homography and get the estimated point $\tilde{x'_i}$. Compute the distance between and x'_i which is
$$d_i = \text{sum}(\text{abs}(\tilde{x'_i} - x'_i))$$
.
3. Count the number of $d_i < \delta$, if it is grater than M, then we mark these pairs as a candidate inlier set.
4. Repeat 1st step N times, and chose the set with largest number of inlier set as the inliers.

The final survive set are inliers, and we are going to use LM Algorithm to find the homography from this set in next Sec. 1.3.

1.3 Levenberg-Marquardt Algorithm (LM)

LM algorithm is nonlinear optimization method based on least squares. Assume the matched points x_i in the image A forms a set as X, the corrsponding matched points x'_i in the image B is X', where $X' = HX$. Please note, here we use Homogeneous Coordinates. Then we want to find a minimizer H, where
$$\tilde{H} = \underset{H}{\text{argmin}}\,(X' - HX)^2),$$
This is the problem we want to solve. The we regard H as a vector $\vec{H}$ as we stated in Sec. 1.1. We formulated a function $f(\vec{H}) = HX$ for notation convinence. Then the problem is re-stated as
$$\tilde{H} = \underset{H}{\text{argmin}}\,(X' - f(\vec{H}))^2)$$

To start a minimization, the user has to provide an initial guess for the parameter vector $\vec{H_0}$. Usually, I like to start with $H_0 = [1, \ldots, 1]$.
The Jacobina of $f(\vec{H})$ is
$${}^{3\times 9}J_f = \begin{bmatrix} \frac{\partial f_1}{\partial h_{11}} & \cdots & \frac{\partial f_1}{\partial h_{33}} \\ \frac{\partial f_2}{\partial h_{11}} & \cdots & \frac{\partial f_2}{\partial h_{33}} \\ \frac{\partial f_3}{\partial h_{11}} & \cdots & \frac{\partial f_3}{\partial h_{33}} \end{bmatrix}.$$

Please recall 1st order Taylor expansion,

$$f(\vec{H}+\vec{\delta}) \approx f(\vec{H}) + \epsilon(\vec{H}) = f(\vec{H}) + J_f\vec{\delta}),$$

where $\epsilon(\vec{H}) = f(\vec{H}+\vec{\delta}) - f(\vec{H})$.
Similar as Gaussian-Newton method, we need to find the best $\vec{\delta}$ for getting directly to the minimum.

$$(J_f^T J_f)\vec{\delta} = J_f^T \epsilon(\vec{H})$$

Till here all above are the same as Gaussian-Newton method. Focusing on the fact that, if $J_f^T J_f$ is purely diagonal, we add a damper as μI, I is an identity matrix. Therefore we have

$$\begin{aligned}
&(J_f^T J_f + \mu I)\vec{\delta} = J_f^T \epsilon(\vec{H}) \\
\Rightarrow &\vec{\delta} = (J_f^T J_f + \mu I)^{-1} J_f^T \epsilon(\vec{H}) \\
\Rightarrow &\tilde{\vec{H}} = \vec{H}_0 + \vec{\delta}
\end{aligned}$$

In practice, it is impossible to get $\tilde{\vec{H}}$ in one step. You may run several iterations from above equations to find the optima.

1.4 Mosaicing Method

The Mosacing is to stitch together a sequence of overlapping photos of a scene to create a single panoramic photo. In the homework, we are required to use pairwise homographies to project all images into the coordinate frame of the central image in the sequence.
Since we have done 2 images projection in the homework 2, the only problem is that after each 1-by-1 projection, the coordinate frame of the new image will change. The homography will not match to the new image. The problem can be easily solved by the following method.
We assume all images combing are the same size $h \times w$, and each sequential pair has the homography H_i, where $x_{i+1} = H_i x_i$. The center image is I_c.

1. Calculate all Homograhpy from each image I_i to the central image I_c.
   - For left side images ($c > i$), $H_{i\text{to}c} = \prod_{j=i}^{j<c} H_i$.
   - For right side images ($c < i$), $H_{i\text{to}c} = (\prod_{j=c}^{j<i} H_i)^{-1}$.
2. Use the corner points in the images $(0,0)$, $(h,0)$, $(0,w)$ and (h,w) to project by each $H_{i\text{to}c}$ and get new corner points $(h_i(k), w_i(k))$, where $k = 1,2,3,4$ corresponding sequentially to the projection of $(0,0)$, $(h,0)$, $(0,w)$ and (h,w).
3. Find the minimum and maximum separately of $h_i(k)$ and $w_i(k)$ as $\min(h_i(k))$, $\max(h_i(k))$, $\min(w_i(k))$ and $\max(w_i(k))$.
4. The Mosaicing image size is ($\max(h_i(k))$ - $\min(h_i(k))$) $\times$ ($\max(w_i(k))$ - $\min(w_i(k))$);
   There is an offset for every image is $\min(h_i(k))$ and $\min(w_i(k))$.
   You can change either manually add the offset while projecting, or change the homographies by the following steps,
   (a) Standardize the homography

   $$H_{i\text{toNew}} = \frac{H_{i\text{to}c}}{H_{i\text{to}c}(3,3)}$$

   (b) Add the offset on height

   $$H_{i\text{toNew}}(1,3) = H_{i\text{to}c}(1,3) + \min(h_i(k))$$

   (c) Add the offset on width

   $$H_{i\text{toNew}}(2,3) = H_{i\text{to}c}(2,3) + \min(w_i(k))$$

   (d) Project all images into the new Mosaicing image frame by $x_{New} = H_{i\text{toNew}} x_i$

2 Test Picture Set

The following 5 pictures are the original pictures for this homework. The pictures are taken sequentially from left to right by a iPhone 7.

(a) 1.jpg

(b) 2.jpg

(c) 3.jpg

(d) 4.jpg

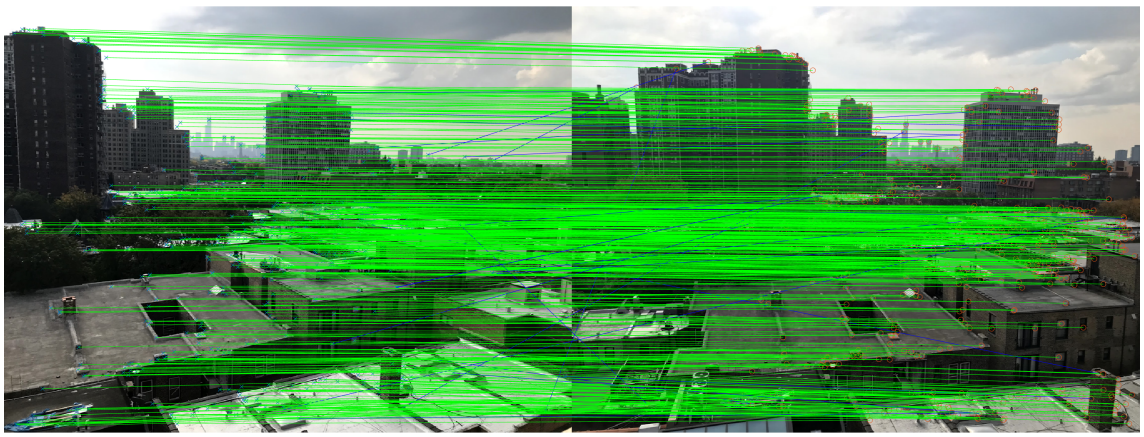
(e) 5.jpg

Figure 1: My picture set for homework.

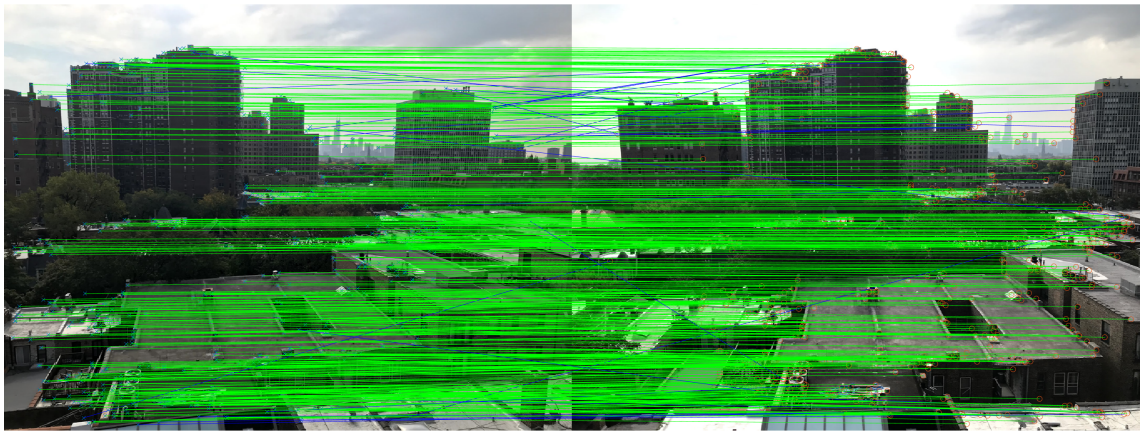
3 Results

3.1 SURF Match and RANSAC

The blue cross, $\times$, represents the interest points on the left images. The red circle, $\circ$, represents the interest points on the right images. The green line represents the inlier matching pairs. The blue line represents the outlier matching pairs.

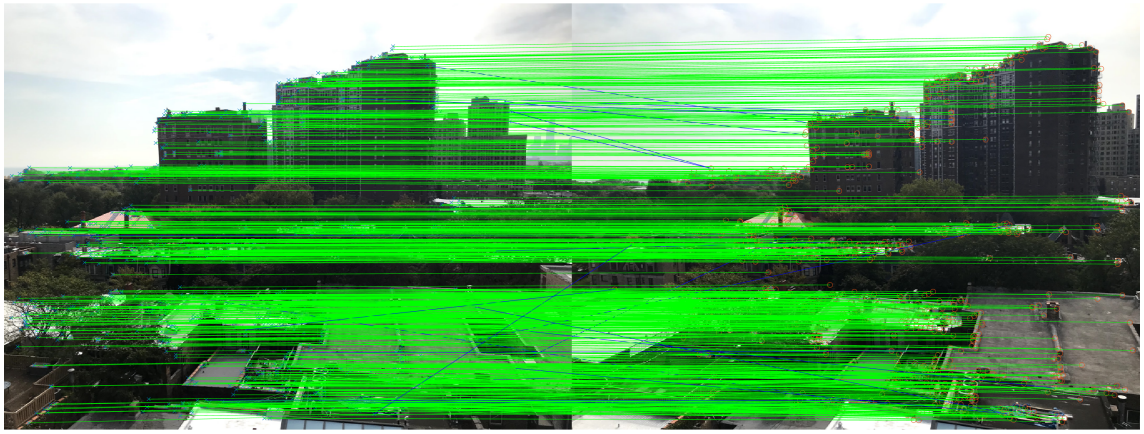


(a) SURF match between 1.jpg and 2.jpg



(b) SURF match between 2.jpg and 3.jpg

(c) SURF match between 3.jpg and 4.jpg



(d) SURF match between 4.jpg and 5.jpg

Figure 2: The SURF match results after RANSAC algorithm.

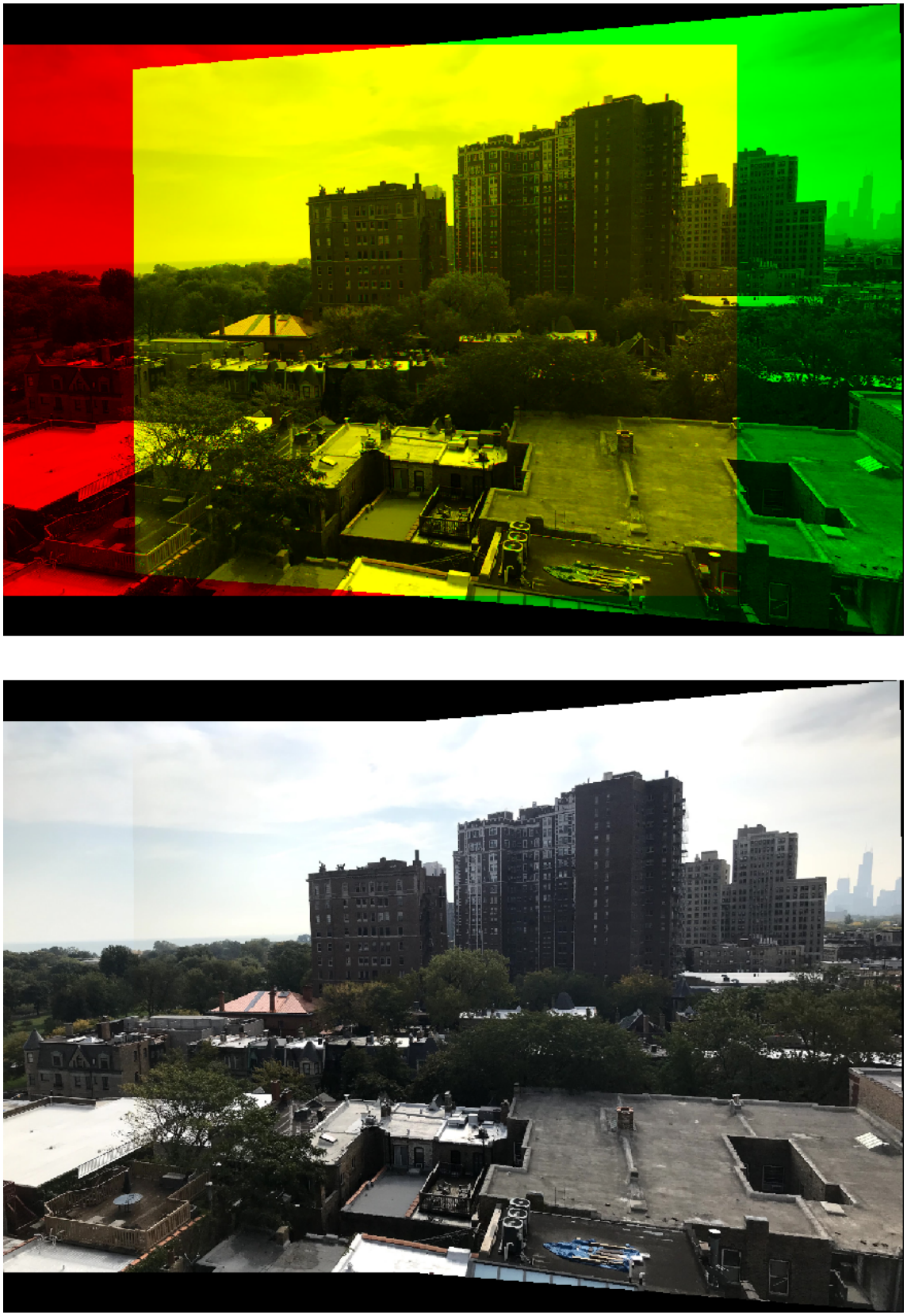
3.2 Projection by Homography from AuHoCa()

The following pictures are generate by the transformation according to the pairwise homographies calculated by the AuHoCa function. Please notice, for better visualization, the projecting picture is tinted in green; the projected picture is tinted in red; their intersection is tinted in yellow due to the overlapping. There is an un-tinted picture shown below each tinted picture.

(a) Projection of 1.jpg on 2.jpg

(b) Projection of 2.jpg on 3.jpg

(c) Projection of 3.jpg on 4.jpg

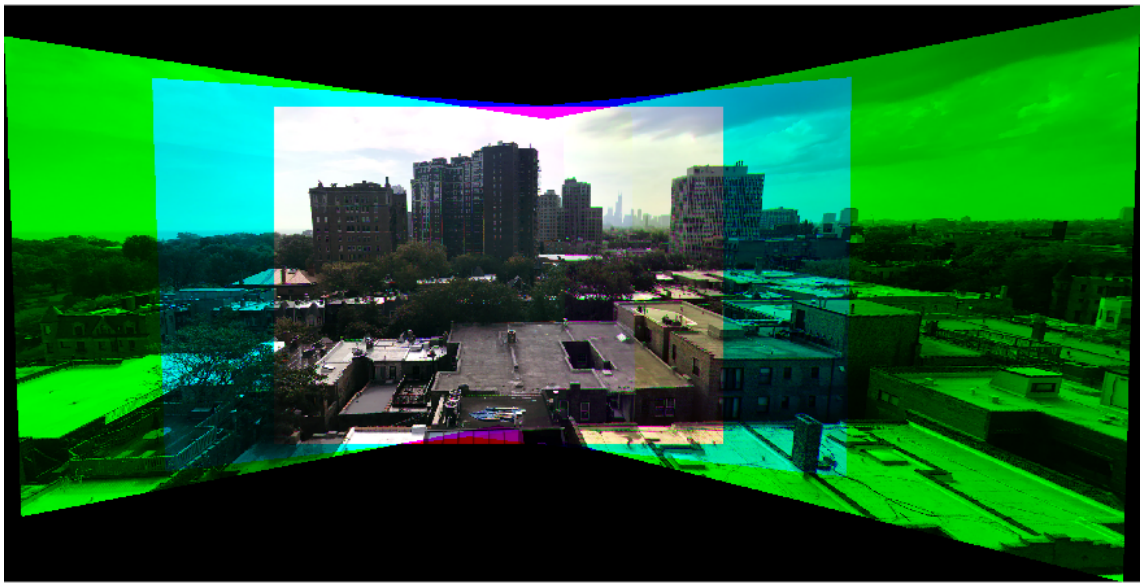


(d) Projection of 4.jpg on 5.jpg

Figure 3: The projection results by AuHoCa Homographies.

3.3 Monsaicing Image

The Monsaicing image Fig. 4 is generate by the transformation according to the pairwise homographies calculated by the AuHoCa function. In upper figure, Fig. 4a, the 3.jpg is tinted in red as the base image; the 1.jpg and 5.jpg is tinted in green; the 2.jpg and 4.jpg is tinted in blue. Due to the overlapping, the color varies at the intersection. There is an un-tinted picture Fig. 4b shown after tinted picture.



(a) The tinted Monsaicing image

(b) The un-tinted Monsaicing image

Figure 4: The final Monsaicing image.

4 Source Code

The code is written in Matlab 2018b and Windows 10.

4.1 Main Script

```
%% ECE 661 2018 Fall Homework 5
% Ye Shi
% shi349@purdue.edu

close all;clear; clc;

%% read all images
img = cell(5,1);
for i = 1:5
    img{i} = imresize(imread([pwd , '\MyPics\',num2str(i),'.jpg']),[900,1200]);
% img{i} = imread([pwd , '\MyPics\',num2str(i),'.jpg']);
end

%% Auto Homography Calculation
H = cell(4,1);
Newimg = cell(4,1);
for i = 1:4 % i=1 corresponds to 1.jpg to 2.jpg
    H{i} = AuHoCa(img{i},img{i+1},[num2str(i),'To',num2str(i+1)]);
    projectH(img{i},img{i+1},H{i},1,[num2str(i),'To',num2str(i+1)]);
    Newimg{i} = projectH(img{i},img{i+1},H{i},0,[num2str(i),'To',num2str(i+1)]);
end

%% Mosaicing 5 pictures
Finalimg = mosaicing5(img,H,['SuperFinal']);
```

4.2 Automatic Homography Calculator

```
%% ECE 661 2018 Fall Homework 5
% Ye Shi
% shi349@purdue.edu

function H = AuHoCa(img1,img2,filename)
% Automatic Homography Calculator
% Find matched SURF features pairs
if nargin <3
    filename = datetime('today');
end

pairs =  surfMatch(img1,img2,filename);
% RANSAC
[bestIdx,~] = ransacH(pairs);


pltPairs(img1,img2,pairs, bestIdx,[filename,'S']);

% LM optimization
H = LMfindH(pairs(bestIdx,:));
end
```

4.2.1 Match SURF Feature Function

```
%% ECE 661 2018 Fall Homework 5
% Ye Shi
% shi349@purdue.edu

function pairs=surfMatch(img1,img2,filename)
% This function is to perform SURF match process
I1 = rgb2gray(img1);
I2 = rgb2gray(img2);

% detect and generate SURF features
points1 = detectSURFFeatures(I1);
points2 = detectSURFFeatures(I2);

[features1,valid_points1] = extractFeatures(I1,points1);
[features2,valid_points2] = extractFeatures(I2,points2);

% match SURF features
indexPairs = matchFeatures(features1,features2);

matchedPoints1 = valid_points1(indexPairs(:,1),:);
matchedPoints2 = valid_points2(indexPairs(:,2),:);

pairs = [matchedPoints1.Location, matchedPoints2.Location];

% Plot the image
%pltPairs(img1,img2,valid_points1.Location,valid_points2.Location ,pairs
    (:,[2,1,4,3]),[filename,'SURF'])
pairs = pairs(:,[2,1,4,3]);

end
```

4.2.2 RANSAC

```
%% ECE 661 2018 Fall Homework 5
% Ye Shi
% shi349@purdue.edu

function [bestIdx,bestH] = ransacH(pairs)
% This function can find the most fitting Homograpy from pairs by RANSAC
% pairs is the set of all matched points.
nt= size(pairs,1); % Total number of matches
% parameters:
p = .99; % The probability that at least one of the N trials will be free of
    outliers.
m = 9; % the set of correspondences for calculating homography matrix H in each
    trial.
esp = 0.2; % the rough estimation of false correspondences from the total set.
```

```
13  del = 20; % The decision threshold to construct the inlier set.
14  N  = ceil(log(1-p)/log(1-(1-esp)^m))% Number of trials
15  M = floor((1-esp)*nt) % A minimum value for the size of the inlier set for it to be
        acceptable
16
17  Maxcount = 0;
18  bestH  = zeros(3);
19  bestIdx = zeros(nt,1);
20  for i = 1:N
21      randRowIdx = randsample(nt,m);
22      H = findH(pairs(randRowIdx,:));
23      estPoints = H*[pairs(:,1:2)';ones(1,nt)];
24      estPoints = estPoints./ estPoints(end,:);
25      dist  = sum(abs( (estPoints(1:2,:) - pairs(:,3:4)')));
26      inIdx = find(dist<del);
27      count = length(inIdx) % use a naive cheese board distance
28      if count > M &&  count > Maxcount
29          bestH = H
30          bestIdx = inIdx;
31          Maxcount = count
32      end
33  end
34  if sum(bestIdx) == 0
35      error('No inline pairs found!');
36  end
37
38
39  end
```

4.2.3 LSM Homography Calculator

```
1  %% ECE 661 2018 Fall Homework 5
2  % Ye Shi
3  % shi349@purdue.edu
4
5  function H = findH(pairs)
6  % This function can find homography by SVD
7  % pairs are m (m>5) pairs of corresponding points by (x1,y1,x2,y2)
8  m = size(pairs,1);
9  if m < 5
10     disp("No necessary number of pairs (m<5)");
11     return
12 end
13 disp([num2str(m)," pairs points are used to find the homography."]);
14 A = zeros(m*2,9);
15 % Construct A
16 for i = 0:m-1
17     A(i*2+1:i*2+2,:) = [0 0 0 -pairs(i+1,1) -pairs(i+1,2) -1 pairs(i+1,4)*pairs(i
           +1,1) pairs(i+1,4)*pairs(i+1,2) pairs(i+1,4);
18         pairs(i+1,1) pairs(i+1,2) 1 0 0 0 -pairs(i+1,1)*pairs(i+1,3) -pairs(i+1,3)*
               pairs(i+1,2) -pairs(i+1,3)];
19 end
20 if rank(A)< 9
21     disp("Warning: Rank(A) is less than 9!");
22 end
23 [~,~,V] = svd(A); % V is order by the descending order of S
24 H = V(:,end); % H is the vector with smallest S
25 H = [H(1:3)';H(4:6)';H(7:9)'];
26 end
```

4.2.4 Levenberg-Marquardt Homography Calculator

```
1  %% ECE 661 2018 Fall Homework 5
2  % Ye Shi
3  % shi349@purdue.edu
4
5  function H = LMfindH(pairs)
6  pairs = double(pairs);
7  H0 = 1*ones(1,9);
8  fun = @(H) myfun(H,pairs);
9  options.Algorithm = 'levenberg-marquardt';
10 H = lsqnonlin(fun,H0,[],[],options);
11 H = [H(1:3);H(4:6);H(7:9)];
```

```
12
13  end
14
15  function err = myfun(H,pairs)
16  % This is the function for non linear least square optimzation
17  H = [H(1:3);H(4:6);H(7:9)];
18  n = size(pairs,1);
19  xdata = [pairs(:,1:2),ones(n,1)]';
20  ydata = [pairs(:,3:4),ones(n,1)]';
21  estydata = H*xdata;
22  estydata = estydata./estydata(end,:);
23  err = ydata - estydata;
24  end
```

4.2.5 SURF Matching Plot with Inliers and Outliers

```
1  %% ECE 661 2018 Fall Homework 5
2  % Ye Shi
3  % shi 349@purdue.edu
4
5  function pltPairs(C1,C2,pairs, pairsIdx,filename)
6  % This function is to generate the final image of SURF matching
7  % also indicates inliners and outliners
8
9  [h,w,c] = size(C1);
10 fig = figure;
11 imshow([C1,C2]);
12 hold on
13 V1 = pairs(:,[2,1]);
14 V2 = pairs(:,[4,3]);
15 plot(V1(:,1),V1(:,2),'x');
16 hold on
17 plot(V2(:,1)+w,V2(:,2),'o');
18 hold on
19 set(gca,'position',[0 0 1 1],'units','normalized')
20 x1 = pairs(:,1);
21 y1 = pairs(:,2);
22 x2 = pairs(:,3);
23 y2 = pairs(:,4)+w;
24
25 for i = 1:size(pairs,1)
26     if ismember(i, pairsIdx)
27     line([y1(i),y2(i)],[x1(i),x2(i)],'Color','g','LineStyle','-');
28     hold on
29     else
30             line([y1(i),y2(i)],[x1(i),x2(i)],'Color','b','LineStyle','-');
31     hold on
32     end
33 end
34 hold off;
35 saveas(fig,[filename,'.png'])
36 end
```

```
1  %% ECE 661 2018 Fall Homework 5
2  % Ye Shi
3  % shi349@purdue.edu
4
5  function H = LMfindH(pairs)
6  pairs = double(pairs);
7  H0 = 1*ones(1,9);
8  fun = @(H) myfun(H,pairs);
9  options.Algorithm = 'levenberg-marquardt';
10 H = lsqnonlin(fun,H0,[],[],options);
11 H = [H(1:3);H(4:6);H(7:9)];
12
13 end
14
15 function err = myfun(H,pairs)
16 % This is the function for non linear least square optimzation
17 H = [H(1:3);H(4:6);H(7:9)];
18 n = size(pairs,1);
19 xdata = [pairs(:,1:2),ones(n,1)]';
20 ydata = [pairs(:,3:4),ones(n,1)]';
```

```
estydata = H*xdata;
estydata = estydata./estydata(end,:);
err = ydata - estydata;
end
```

4.3 Projection Function by Homography

```
%% ECE 661 2018 Fall Homework 5
% Ye Shi
% shi349@purdue.edu

function img = projectH(img1,img2,H,mono,filename)
% This function can project img1 to img2 by H, x2 = H*x1
if nargin <4
    filename = datetime('today');
end
% Mono switch
% mono  = 0

[h1,w1,c1] = size(img1);
[h2,w2,c2] = size(img2);
% H = H'
% define new img size
points = [0 0 1; h1 0 1; 0 w1 1; h1 w1 1]';
newpoints = H*points;
newpoints = newpoints./newpoints(end,:);
minh1 = floor(min(newpoints(1,:)));
maxh1 = ceil(max(newpoints(1,:)));
minw1 = floor(min(newpoints(2,:)));
maxw1 = ceil(max(newpoints(2,:)));

minh = min([minh1,0]);
maxh = max([maxh1,h2]);
minw = min([minw1,0]);
maxw = max([maxw1,w2]);

newh = ceil(maxh - minh);
neww = ceil(maxw-minw);

img = zeros(newh,neww,c1);


for x = 1:h2
    for y = 1:w2
        if mono == 1
            img(x-minh,y-minw,1)= img2(x,y,1);
        else
            img(x-minh,y-minw,:)= img2(x,y,:);
        end

    end
end

invH = H^-1;

for x = 1:newh
    for y = 1:neww
        Loc1 = invH*[x+minh,y+minw,1]';
        Loc1 = round(Loc1./Loc1(end));
        if Loc1(1) > 0 && Loc1(1)<= h1 && Loc1(2) > 0 && Loc1(2)<= w1
            if mono == 1
                img(x,y,2) = img1(Loc1(1),Loc1(2),2);
            else
                if sum(img1(Loc1(1),Loc1(2),:)) ~= 0
                    img(x,y,:) = img1(Loc1(1),Loc1(2),:);
                end
            end
        end
    end
end

if mono == 1
    filename = [filename,'M'];
```

```
end


img = uint8(img);
fig = figure;
set(gca,'position',[0 0 1 1],'units','normalized')
imshow(img)
saveas(fig,[filename,'.png'])
end
```

4.4 Mosaicing Function by Homography

```
%% ECE 661 2018 Fall Homework 5
% Ye Shi
% shi349@purdue.edu

function img = mosaicing5(imgset,H,filename)
% imgset is a 5-image cell
% H is a 4-pairwise-homography cell
if nargin <3
    filename = datetime('today');
end
% Mono switch
mono = 1

% All images are the same size
[h1,w1,c1] = size(imgset{1});

% define new img size
points = [0 0 1; h1 0 1; 0 w1 1; h1 w1 1]';
newpoints1 = H{1}*H{2}*points;
newpoints1 = newpoints1./newpoints1(end,:);
minh1 = floor(min(newpoints1(1,:)));
maxh1 = ceil(max(newpoints1(1,:)));
minw1 = floor(min(newpoints1(2,:)));
maxw1 = ceil(max(newpoints1(2,:)));

newpoints2 = H{2}*points;
newpoints2 = newpoints2./newpoints2(end,:);
minh2 = floor(min(newpoints2(1,:)));
maxh2 = ceil(max(newpoints2(1,:)));
minw2 = floor(min(newpoints2(2,:)));
maxw2 = ceil(max(newpoints2(2,:)));

newpoints3 = H{3}^-1 * points;
newpoints3 = newpoints3./newpoints3(end,:);
minh3 = floor(min(newpoints3(1,:)));
maxh3 = ceil(max(newpoints3(1,:)));
minw3 = floor(min(newpoints3(2,:)));
maxw3 = ceil(max(newpoints3(2,:)));

newpoints4 = H{3}^-1 * H{4}^-1 *points;
newpoints4 = newpoints4./newpoints4(end,:);
minh4 = floor(min(newpoints4(1,:)));
maxh4 = ceil(max(newpoints4(1,:)));
minw4 = floor(min(newpoints4(2,:)));
maxw4 = ceil(max(newpoints4(2,:)));

% Find the extrima of all projections
minh = min([minh1,minh2,minh3,minh4,0]);
maxh = max([maxh1,maxh2,maxh3,maxh4,h1]);
minw = min([minw1,minw2,minw3,minw4,0]);
maxw = max([maxw1,maxw2,maxw3,maxw4,w1]);


newh = ceil(maxh - minh);
neww = ceil(maxw-minw);

img = zeros(newh,neww,c1);

% middle image is the base
for x = 1:h1
    for y = 1:w1
```

```
62            if mono == 1
63                img(x-minh,y-minw,1)= imgset{3}(x,y,1);
64            else
65                img(x-minh,y-minw,:)= imgset{3}(x,y,:);
66            end
67
68        end
69    end
70
71    invH = (H{1}*H{2})^-1;
72
73    % Left image
74    for x = 1:newh
75        for y = 1:neww
76            Loc1 = invH*[x+minh,y+minw,1]';
77            Loc1 = round(Loc1./Loc1(end));
78            if Loc1(1) > 0 && Loc1(1)<= h1 && Loc1(2) > 0 && Loc1(2)<= w1
79                if mono == 1
80                    img(x,y,2) = imgset{1}(Loc1(1),Loc1(2),2);
81                else
82                    if sum(imgset{1}(Loc1(1),Loc1(2),:)) ~= 0
83                        img(x,y,:) = imgset{1}(Loc1(1),Loc1(2),:);
84                    end
85                end
86            end
87        end
88    end
89
90
91    invH = H{2}^-1;
92    % Left-middle image
93    for x = 1:newh
94        for y = 1:neww
95            Loc1 = invH*[x+minh,y+minw,1]';
96            Loc1 = round(Loc1./Loc1(end));
97            if Loc1(1) > 0 && Loc1(1)<= h1 && Loc1(2) > 0 && Loc1(2)<= w1
98                if mono == 1
99                    img(x,y,3) = imgset{2}(Loc1(1),Loc1(2),3);
100               else
101                   if sum(imgset{2}(Loc1(1),Loc1(2),:)) ~= 0
102                       img(x,y,:) = imgset{2}(Loc1(1),Loc1(2),:);
103                   end
104               end
105           end
106       end
107   end
108
109
110
111   invH = H{3};
112   % Right-middle image
113   for x = 1:newh
114       for y = 1:neww
115           Loc1 = invH*[x+minh,y+minw,1]';
116           Loc1 = round(Loc1./Loc1(end));
117           if Loc1(1) > 0 && Loc1(1)<= h1 && Loc1(2) > 0 && Loc1(2)<= w1
118               if mono == 1
119                   img(x,y,3) = imgset{4}(Loc1(1),Loc1(2),3);
120               else
121                   if sum(imgset{4}(Loc1(1),Loc1(2),:)) ~= 0
122                       img(x,y,:) = imgset{4}(Loc1(1),Loc1(2),:);
123                   end
124               end
125           end
126       end
127   end
128
129   invH = H{4}*H{3};
130   % Right-middle image
131   for x = 1:newh
132       for y = 1:neww
133           Loc1 = invH*[x+minh,y+minw,1]';
134           Loc1 = round(Loc1./Loc1(end));
```

```
135             if Loc1(1) > 0 && Loc1(1)<= h1 && Loc1(2) > 0 && Loc1(2)<= w1
136                 if mono == 1
137                     img(x,y,2) = imgset{5}(Loc1(1),Loc1(2),2);
138                 else
139                     if sum(imgset{5}(Loc1(1),Loc1(2),:)) ~= 0
140                         img(x,y,:) = imgset{5}(Loc1(1),Loc1(2),:);
141                     end
142                 end
143             end
144         end
145     end
146
147     if mono == 1
148         filename = [filename,'M'];
149     end
150
151
152     img = uint8(img);
153     fig = figure;
154     set(gca,'position',[0 0 1 1],'units','normalized')
155     imshow(img)
156     saveas(fig,[filename,'.png'])
157     end
```