ECE 661 HW # 5

Joonsoo Kim(PUID : 00258 – 41316)

Email : kim1449@purdue.edu

10-16-2014

1. Finding corresponding points based on SIFT between 2 images.

In this experiment, we used SIFT features and descriptors to find corresponding points between 2 images. Let two images be img1 and img2. Then the whole process to find corresponding points between 2 images can be explained as follows.

1) Extract SIFT features with descriptors in img1.
2) Extract SIFT features with descriptors in img2.
3) Compute all the distances (L2norm) between the descriptor of one feature point in img1 and the descriptors of all feature points in img2.
4) Find two feature points in img2 which have 1st minimum distance(d1) and 2nd minimum distance(d2) with the feature point in img1.
5) If d1/d2 is smaller than a threshold we fixed, we consider the feature point in img1 and the feature point having 1st minimum distance as corresponding points.
6) Iterate 3)~5) for all feature points in img1.

As above, if we use threshold for ratio between two closest feature points, we can get more accurate corresponding points.

2. RANSAC (Random sample consensus)

Since we have several corresponding points, we can estimate homography between two images(img1, img2). However, the corresponding points we obtained from SIFT matching can have false correspondences (outliers) which can affect the incorrect estimation of homography, so we need to do the process to remove outliers before estimating homography. Therefore, one of the robust estimation methods RANSAC can be used to do that. The whole process of RANSAC can be explained as follows.

1) Set parameters for RANSAC

   Since there are some parameters for RANSAC such as N, p, M, n, δ and ε we need to set before running RANSAC, we set the values for the parameters.

   N : the number of trials
   P : the probability that at least one of the N trials will be free of outliers.
   M : a minimum value for the size of the inlier set for it to be acceptable.
   n : the number of correspondences which are randomly chosen at each trial
   ε : probability that any single correspondence is a false inlier
   δ : decision threshold to decide if some corresponding points are inliers.(Here we consider this threshold as distance threshold)

Since we used the 10% rule described in the Lecture note, we set ε = 0.1. Additionally, we set all parameter as follows

ε = 0.1.

p = 0.99

n= 6

δ = 20(pixels)

M = n_total * (1- ε) (n_total : the total number of correspondences obtained by SIFT matching between two images)

N = $\frac{\ln(1-p)}{\ln(1-(1-\epsilon)^n)}$ = 6

2) Select n correspondences randomly from all correspondences obtained by SIFT matching.
3) Calculate homography H between two images using linear least squares method based on the n correspondences. Here we used homogeneous linear least square method. The homography H is computed using homogeneous linear least square method as follows.

Let (X,X') be correspondence between two images(img1 and img2) where X = [x y w]T and X'=[x' y' w']T. Then we can say X' = HX.

We can say that X'⊕HX=0 (⊕ : cross product). Then we can get a following equation.

$$\begin{bmatrix} 0 & 0 & 0 & -w'x & -w'y & -w'w & y'x & y'y & y'w \\ w'x & w'y & w'w & 0 & 0 & 0 & -x'x & -x'y & -x'w \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{22} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0$$

Since we can get this equation for one correspondences, we can get a following equation for n correspondences.

$$\begin{bmatrix} 0 & 0 & 0 & -w_1'x_1 & -w_1'\,y_1 & -w_1'w_1 & y_1'x_1 & y_1'y_1 & y_1'w_1 \\ w_1'x_1 & w_1'y_1 & w_1'w_1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1'w_1 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & -w_n'x_n & -w_n'\,y_n & -w_n'w_n & y_n'x_n & y_n'y_n & y_n'w_n \\ w_n'x_n & w_n'y_n & w_n'w_n & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n'w_n \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{22} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0$$

Where (x_i,y_i,w_i) : i^{th} point in img1. And w = 1 in experiment.

Then we can express above equation as Ah = 0 (A : 2n x 9 matrix, h : 9 x 1 vector). When we solve this equation, we use a constraint ||h||=1 to prevent h = 0. Then the solution of this equation is h = the eigenvector corresponding to the smallest eigenvalue for A^TA.

4) Compute the distances between the true location(HX) and measured location (X') of all correspondences between two images. And if the distance is smaller than decision thresholdδ, we add the correspondence into a inlier set. (
5) Do N trials for 2)~4). And keep the homography which has the most number of inliers. (the number of inlier should bigger than M)

3. Homography refinement using DogLeg

To implement this method, we need to know step size for Gradient Decent and Gaussian Newton. They are defined as follows.

$$\delta_{p,GD} = \frac{||J_f^T \varepsilon(p_k)||}{||J_f J_f^T \varepsilon(p_k)||} J_f^T \varepsilon(p_k), \quad \delta_{p,GN} = \left(J_f^T J_f + u_k I\right)^{-1} J_f^T \varepsilon(p_k)$$

Where J_f : Jacobian matrix of f with repect to p, $\varepsilon(p_k) = \sum ||X'_{phy} \; - f(p_k)||^2$, u_k : control parameter, X'_{phy}: point in physical plane. For applying this method to our system, we need to define vector p_k and $f(p_k)$ as follows

p_k = $[h_{11}\ h_{12}\ h_{13}\ h_{21}\ h_{22}\ h_{23}\ h_{31}\ h_{32}\ h_{33}]T$ where h_{ij} is component of matrix H.

$$f(p_k) = \begin{bmatrix} x_{measuredphy} \\ y_{measuredphy} \end{bmatrix} = \begin{bmatrix} f_1(p_k) \\ f_2(p_k) \end{bmatrix} = \begin{bmatrix} \frac{h_{11}x+h_{12}y+h_{13}}{h_{31}x+h_{32}y+h_{33}} \\ \frac{h_{21}x+h_{22}y+h_{23}}{h_{31}x+h_{32}y+h_{33}} \end{bmatrix}$$

And the update of p_k is done as follows.

$$p_{k+1} = p_k + \delta_k$$

$$\delta_k = [\begin{matrix} \delta_{p,GN} \quad \text{if} \quad ||\delta_{p,GN}|| < r_k \\ \delta_{p,GD} + B(\delta_{p,GN} - \delta_{p,GD}) \quad \text{if} \quad ||\delta_{p,GD}|| < r_k < ||\delta_{p,GN}|| \\ \frac{rk}{||\delta_{p,GD}||}\delta_{p,GD} \quad \text{otherwise} \end{matrix}]$$

Since B should satisfy $||\delta_{p,GD} + B(\delta_{p,GN} - \delta_{p,GD})||^2 = r_k^2$, we compute B by solving the equation when $||\delta_{p,GD}|| < r_k < ||\delta_{p,GN}||$.

We also update rk as follows.

$$r_{k+1} = \begin{cases} \frac{r_k}{4} & \text{if} \quad \rho_{dl} < \frac{1}{4} \\ r_k & \text{if} \ \frac{1}{4} < \rho_{dl} \leq \frac{3}{4} \\ 2r_k & \text{otherwise} \end{cases}$$

Where $\rho_{dl} = \frac{\varepsilon(p_k)^T \varepsilon(p_k) - \varepsilon(p_{k+1})^T \varepsilon(p_{k+1})}{2\delta_p^T J_f^T \varepsilon(p_k) - \delta_k^T J_f^T J_f \delta_k}$

'We iterate above equations and update p_k until $\rho_{dl} < 0$.

For initial value for pk, we used homography we computed from RANSAC.

4. Image Mosaicking

In this experiment, we used 5 different images taken in different angles. Let the most left image be image1 and the most right image be image5. Then the center image is image 3. To make panorama image, we compute homography H between two successive images such as (image 1, image 2), (image 2, image 3), (image 4, image 3), (image 5, image 4). Let the homographies be H12, H23, H43, and H54. To combine all images together on center image, we additionally need to know homography, H13, which map image1 to image3 and the homography H53 which maps image5 to image3. That homographies is computed as follows.

H13 = H12*H23, H53 = H54*H43

Now, we can put all images together on center image using Homographies(H13, H23, H43, H53).

5. Experimental Results

We used SIFT matching implemented in vlfeat. To extract SIFT feature robust to noise, we used peak threshold to discard small peak value of feature points. And we used another threshold for ratio between 1st minimum distance(d1) and 2nd minimum distance(d2) we mention before to find good correspondences. All parameters we used in this experiment are in following tables.

	Peak_threshold	Threshold for ratio
value	4	0.33

Table 1. SIFT Matching

	ε	p	n	δ	M	N
value	0.1	0.99	6	20	n_total*0.9	6

Table 2. RANSAC

	u_k	r_k
Initial value	1	5

Table 3. DogLeg

1) Inliers and Outliers (image set 1)

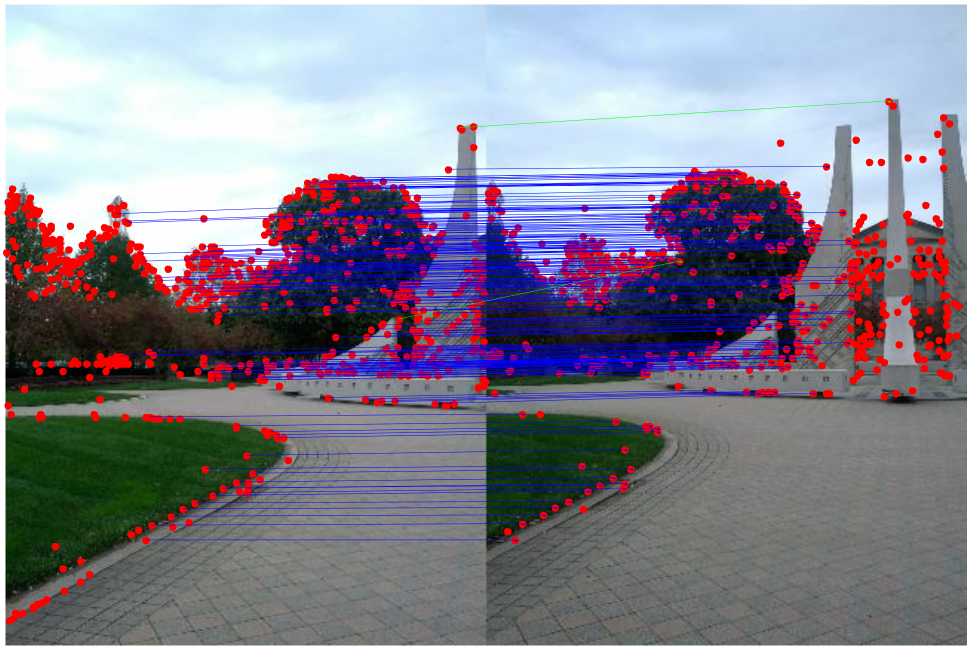

Fig 1. Correspondences between image1(left) and image2(right).
(red points: SIFT feature points, blue line : inliers, green line outliers)

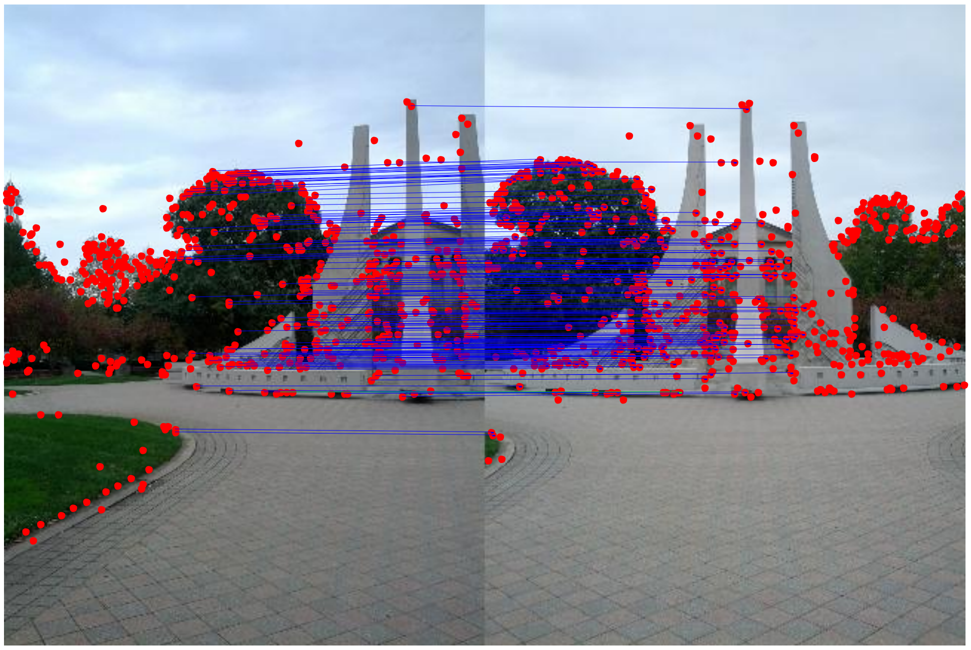

Fig 2. Correspondences between image2(left) and image3(right).
(red points: SIFT feature points, blue line : inliers, green line outliers)

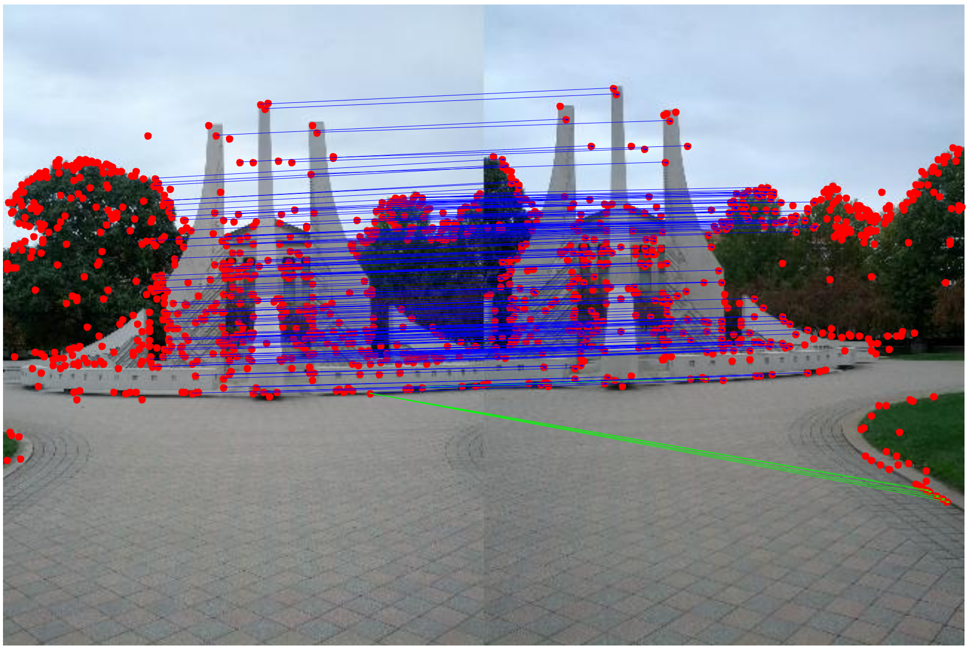

Fig 3. Correspondences between image3(left) and image4(right).
(red points: SIFT feature points, blue line : inliers, green line outliers)

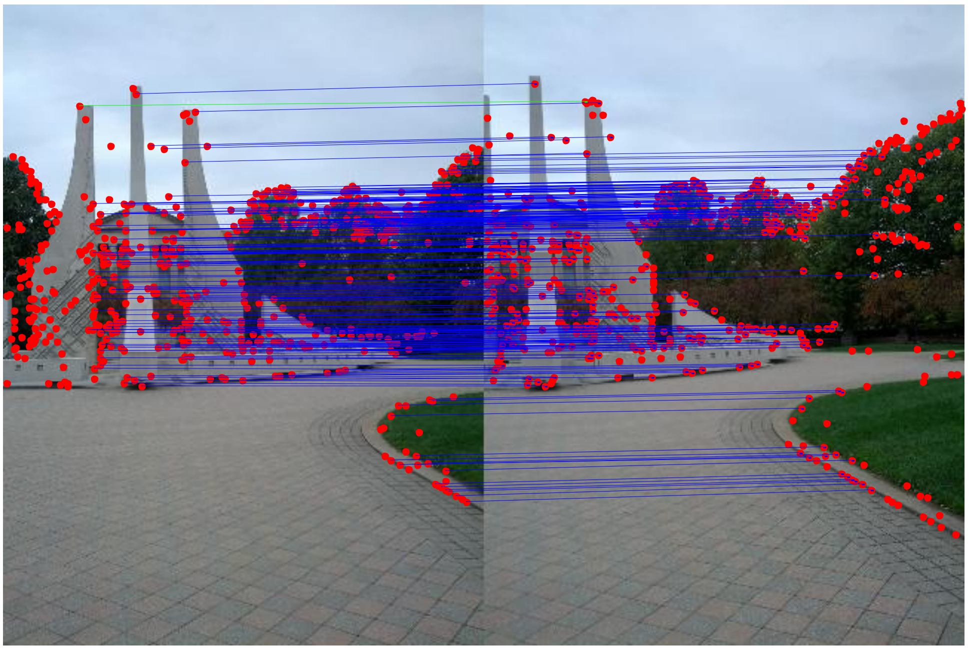

Fig 4. Correspondences between image4(left) and image5(right).
(red points: SIFT feature points, blue line : inliers, green line outliers)

2) Image Mosaicking(image set 1)

Fig 5. Image Mosaicking results(Without DogLeg)

Fig 6. Image Mosaicking results(With DogLeg)

As we can see, when we use homography refinement based on DogLeg, we can get more mosaicking image. (Especially, tower parts are combined better with DogLeg)

3) Inliers and Outliers (image set 2)

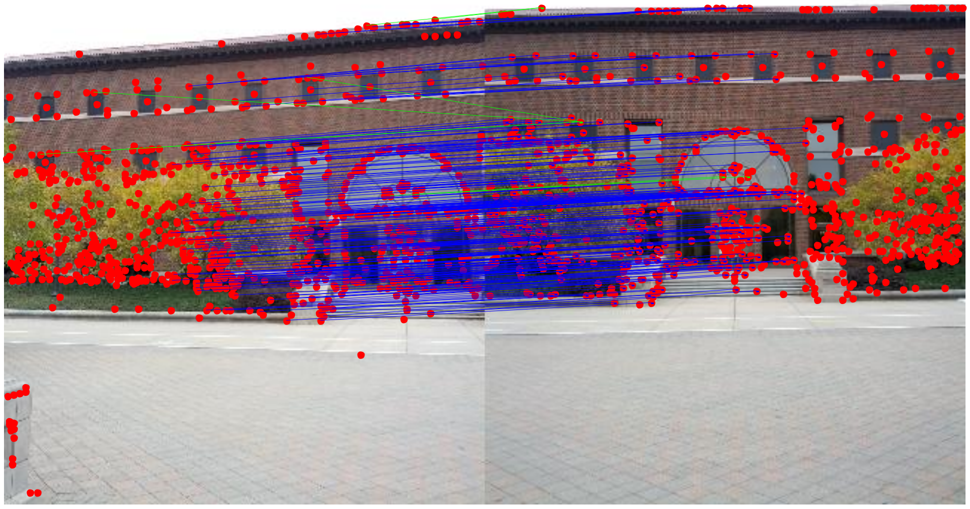

Fig 4. Correspondences between image1(left) and image2(right).
(red points: SIFT feature points, blue line : inliers, green line outliers)

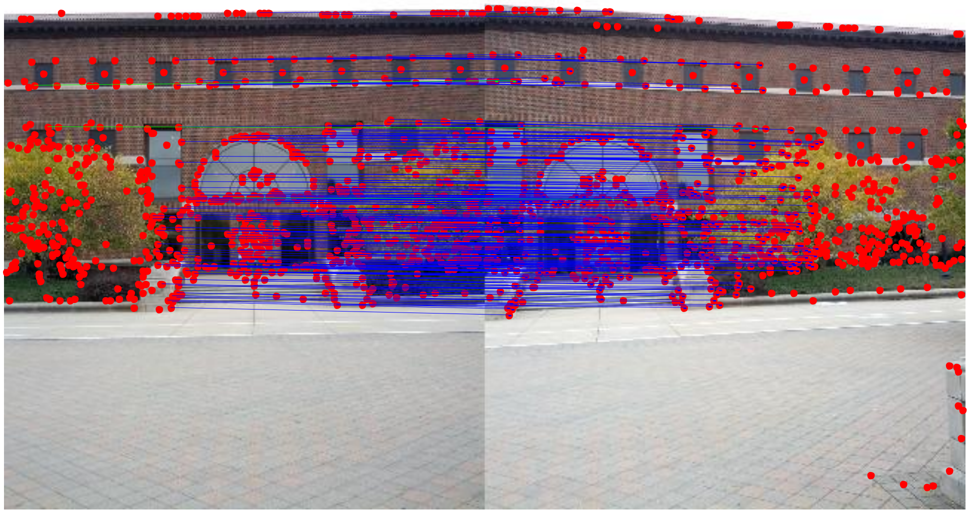

Fig 5. Correspondences between image2(left) and image3(right).
(red points: SIFT feature points, blue line : inliers, green line outliers)

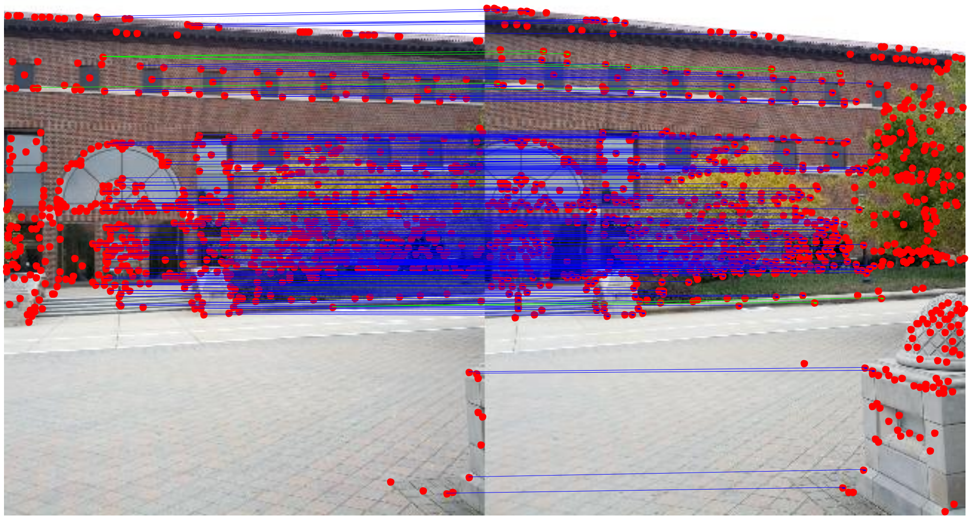

Fig 6. Correspondences between image3(left) and image4(right).
(red points: SIFT feature points, blue line : inliers, green line outliers)

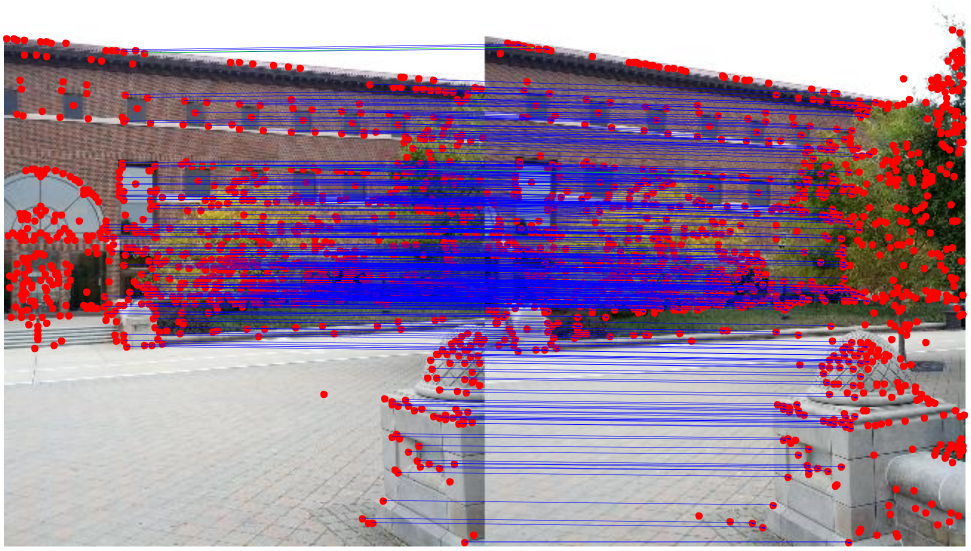

Fig 4. Correspondences between image4(left) and image5(right).
(red points: SIFT feature points, blue line : inliers, green line outliers)

4) Image Mosaicking (image set 2)

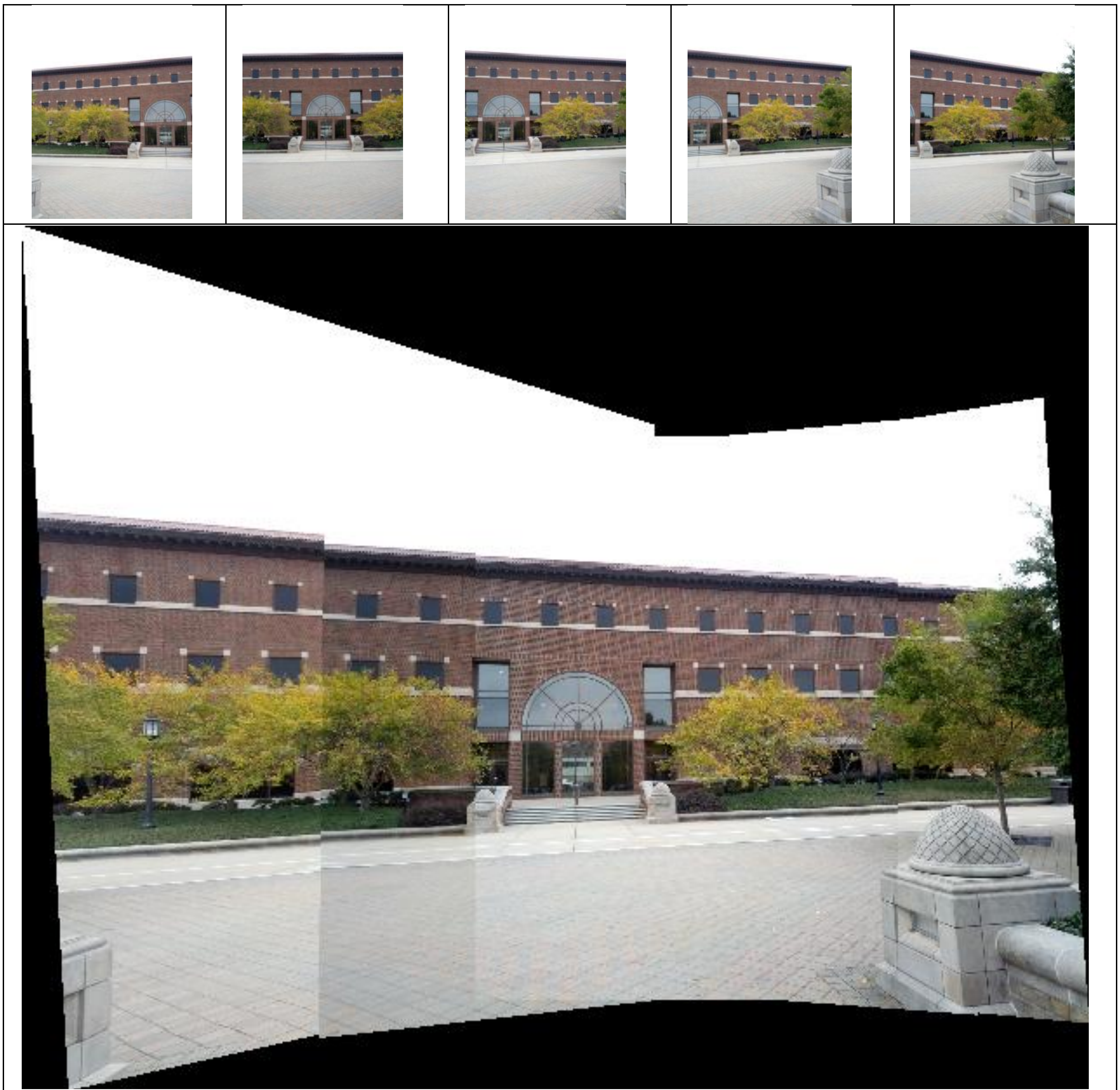

Fig 5. Image Mosaicking results(Without DogLeg)

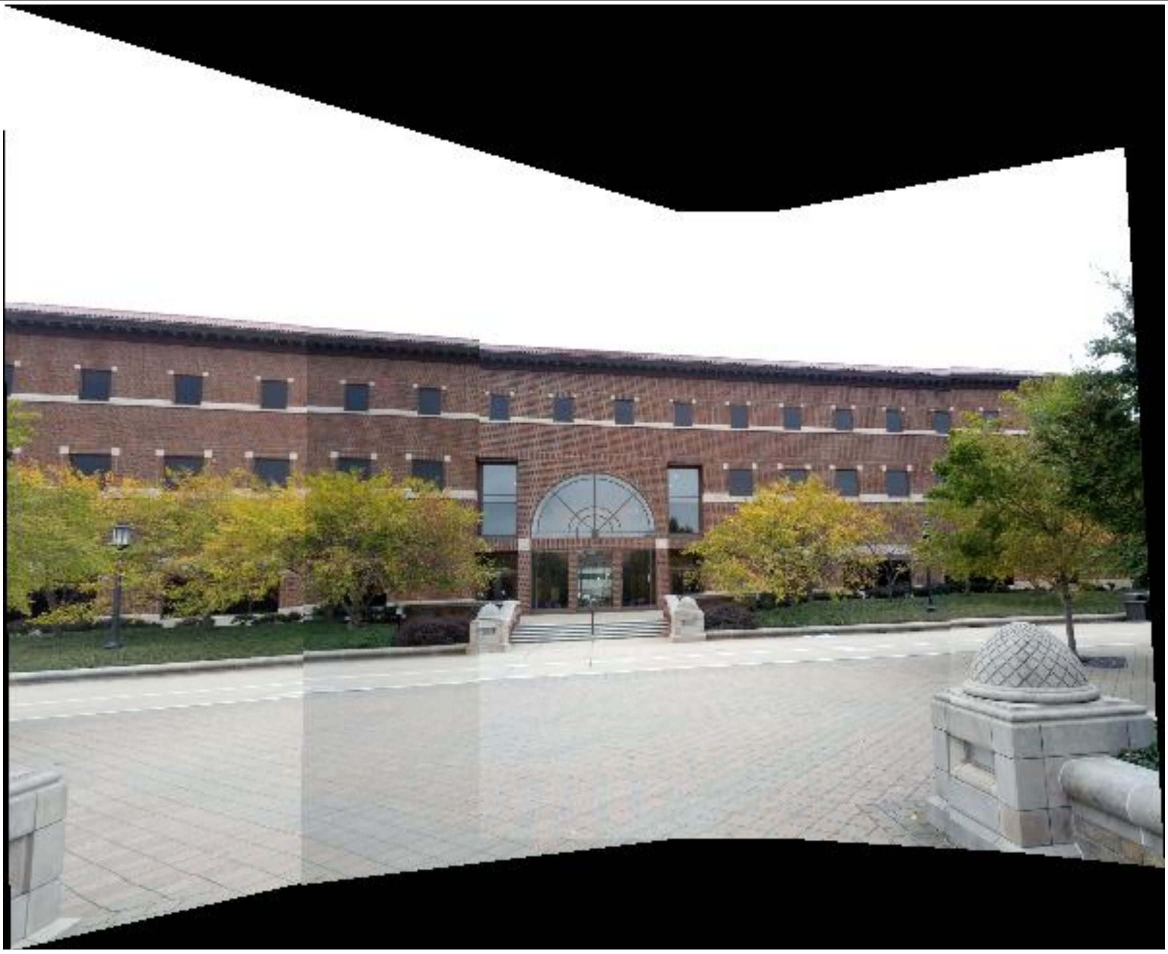

Fig 6. Image Mosaicking results(With DogLeg)

As we can see, when we use homography refinement based on DogLeg, we can get more mosaicking image.

6. Source Code

Main Function

```
clear all;
close all;
clc;
addpath(genpath('.\vlfeat-0.9.17'));
run('.\vlfeat-0.9.17\toolbox\vl_setup');


% parameter setting
peak_thresh = 4;
thres = 0.3;
n = 6;
p = 0.99;
etha = 0.1;
thres_dist12 = 20;
thres_dist23 = 20;
```

```
thres_dist43 = 20;
thres_dist54 = 20;
N = round(log(1-p)/log(1-(1-etha)^n));





img1 = imread('1.jpg');
img2 = imread('2.jpg');
img3 = imread('3.jpg');
img4 = imread('4.jpg');
img5 = imread('5.jpg');

img1 =imresize(img1, 0.5);
img2 =imresize(img2, 0.5);
img3 =imresize(img3, 0.5);
img4 =imresize(img4, 0.5);
img5 =imresize(img5, 0.5);

rgbimg1 = img1;
rgbimg2 = img2;
rgbimg3 = img3;
rgbimg4 = img4;
rgbimg5 = img5;

[row1 col1 ~] = size(img1);
[row2 col2 ~] = size(img2);
[row3 col3 ~] = size(img3);
[row4 col4 ~] = size(img4);
[row5 col5 ~] = size(img5);

% Check if RGB
       if numel(size(img1)) == 3
            I_img1 = rgb2gray(img1);
       else
          I_img1 = img1;
       end

       % Check if RGB
       if numel(size(img2)) == 3
            I_img2 = rgb2gray(img2);
       else
          I_img2 = img2;
       end

        % Check if RGB
       if numel(size(img3)) == 3
            I_img3 = rgb2gray(img3);
       else
          I_img3 = img3;
       end

        % Check if RGB
       if numel(size(img4)) == 3
            I_img4 = rgb2gray(img4);
       else
          I_img4 = img4;
       end
```

```
        % Check if RGB
       if numel(size(img5)) == 3
            I_img5 = rgb2gray(img5);
       else
           I_img5 = img5;
       end



%convert single type for vl_sift
       I_img1 = single(I_img1);
       I_img2 = single(I_img2);
       I_img3 = single(I_img3);
       I_img4 = single(I_img4);
       I_img5 = single(I_img5);



%excute SIFT for test image
       [f1 d1]= vl_sift(I_img1,'PeakThresh', peak_thresh); %
constraint => Peak valkue thresholding
       [f2 d2]= vl_sift(I_img2,'PeakThresh', peak_thresh); %
constraint => Peak valkue thresholding
       [f3 d3]= vl_sift(I_img3,'PeakThresh', peak_thresh); %
constraint => Peak valkue thresholding
       [f4 d4]= vl_sift(I_img4,'PeakThresh', peak_thresh); %
constraint => Peak valkue thresholding
       [f5 d5]= vl_sift(I_img5,'PeakThresh', peak_thresh); %
constraint => Peak valkue thresholding

       % remove same positioned points from different scale
       [~, index1] = unique(f1(1:2,:)','rows');
       [~, index2] = unique(f2(1:2,:)','rows');
       [~, index3] = unique(f3(1:2,:)','rows');
       [~, index4] = unique(f4(1:2,:)','rows');
       [~, index5] = unique(f5(1:2,:)','rows');

       d1 = d1(:, index1);
       f1 = f1(:, index1);
       d2 = d2(:, index2);
       f2 = f2(:, index2);
       d3 = d3(:, index3);
       f3 = f3(:, index3);
       d4 = d4(:, index4);
       f4 = f4(:, index4);
       d5 = d5(:, index5);
       f5 = f5(:, index5);


% figure(1)
% imshow(rgbimg1);
% hold on
% plot(f1(1,:), f1(2,:),  'r.');

% figure(2)
% imshow(rgbimg2);
```

```
% hold on
% plot(f2(1,:), f2(2,:),  'rx');
       d1 = double(d1);
       d2 = double(d2);
       d3 = double(d3);
       d4 = double(d4);
       d5 = double(d5);


[matches12, scores12] = vl_ubcmatch(d1, d2, 1/thres);
[matches23, scores23] = vl_ubcmatch(d2, d3, 1/thres);
[matches43, scores43] = vl_ubcmatch(d4, d3, 1/thres);
[matches54, scores54] = vl_ubcmatch(d5, d4, 1/thres);

%% RANSAC-1

% number of correponding points between two image
n12 = size(matches12, 2); %image 1 and 2
n23 = size(matches23, 2); %image 2 and 3
n43 = size(matches43, 2); %image 3 and 4
n54 = size(matches54, 2); %image 4 and 5

save_inlier12 = cell(N,1);
save_inlier23 = cell(N,1);
save_inlier43 = cell(N,1);
save_inlier54 = cell(N,1);
save_outlier12 = cell(N,1);
save_outlier23 = cell(N,1);
save_outlier43 = cell(N,1);
save_outlier54 = cell(N,1);
save_H12 = cell(N,1);
save_H23 = cell(N,1);
save_H43 = cell(N,1);
save_H54 = cell(N,1);

save_num_inlier12 = zeros(N,1);
save_num_inlier23 = zeros(N,1);
save_num_inlier43 = zeros(N,1);
save_num_inlier54 = zeros(N,1);

% n randomly sampled correspondences
h_pts_12_1 = ones(3, n);
h_pts_12_2 = ones(3, n);
h_pts_23_2 = ones(3, n);
h_pts_23_3 = ones(3, n);
h_pts_43_4 = ones(3, n);
h_pts_43_3 = ones(3, n);
h_pts_54_5 = ones(3, n);
h_pts_54_4 = ones(3, n);

for j=1:N


cindx_n12 = randperm(n12);
indx_n12 = cindx_n12(1:n);
pts_12_1 = round(f1(1:2, matches12(1,indx_n12)));
pts_12_2 = round(f2(1:2, matches12(2,indx_n12)));
h_pts_12_1(1:2, :) = pts_12_1;
```

```
h_pts_12_2(1:2, :) = pts_12_2;

cindx_n23 = randperm(n23);
indx_n23 = cindx_n23(1:n);
pts_23_2 = round(f2(1:2, matches23(1,indx_n23)));
pts_23_3 = round(f3(1:2, matches23(2,indx_n23)));
h_pts_23_2(1:2, :) = pts_23_2;
h_pts_23_3(1:2, :) = pts_23_3;

cindx_n43 = randperm(n43);
indx_n43 = cindx_n43(1:n);
pts_43_4 = round(f4(1:2, matches43(1,indx_n43)));
pts_43_3 = round(f3(1:2, matches43(2,indx_n43)));
h_pts_43_4(1:2, :) = pts_43_4;
h_pts_43_3(1:2, :) = pts_43_3;

cindx_n54 = randperm(n54);
indx_n54 = cindx_n54(1:n);
pts_54_5 = round(f5(1:2, matches54(1,indx_n54)));
pts_54_4 = round(f4(1:2, matches54(2,indx_n54)));
h_pts_54_5(1:2, :) = pts_54_5;
h_pts_54_4(1:2, :) = pts_54_4;

% estimate homography
A12 = [];
A23 = [];
A43 = [];
A54 = [];
for i=1:n

A12 = [A12 ; 0 0 0 -h_pts_12_1(:,i)' h_pts_12_1(:,i)'*pts_12_2(2,
i) ; h_pts_12_1(:,i)' 0 0 0 -h_pts_12_1(:,i)'*pts_12_2(1, i)];
A23 = [A23 ; 0 0 0 -h_pts_23_2(:,i)' h_pts_23_2(:,i)'*pts_23_3(2,
i) ; h_pts_23_2(:,i)' 0 0 0 -h_pts_23_2(:,i)'*pts_23_3(1, i)];
A43 = [A43 ; 0 0 0 -h_pts_43_4(:,i)' h_pts_43_4(:,i)'*pts_43_3(2,
i) ; h_pts_43_4(:,i)' 0 0 0 -h_pts_43_4(:,i)'*pts_43_3(1, i)];
A54 = [A54 ; 0 0 0 -h_pts_54_5(:,i)' h_pts_54_5(:,i)'*pts_54_4(2,
i) ; h_pts_54_5(:,i)' 0 0 0 -h_pts_54_5(:,i)'*pts_54_4(1, i)];

end

% Compute homography H using Linear Least Square to minimize ||Ah||
st ||h|| = 1;
[V12 D12] = eig(A12'*A12);
[V23 D23] = eig(A23'*A23);
[V43 D43] = eig(A43'*A43);
[V54 D54] = eig(A54'*A54);
H12 = [V12(1:3,1)' ; V12(4:6,1)' ; V12(7:9,1)'];
H23 = [V23(1:3,1)' ; V23(4:6,1)' ; V23(7:9,1)'];
H43 = [V43(1:3,1)' ; V43(4:6,1)' ; V43(7:9,1)'];
H54 = [V54(1:3,1)' ; V54(4:6,1)' ; V54(7:9,1)'];



% compute X'= HX
tr_pt12_2 = find_matching_pt_H(round(f1(1:2, matches12(1,:))), H12);
tr_pt23_3 = find_matching_pt_H(round(f2(1:2, matches23(1,:))), H23);
tr_pt43_3 = find_matching_pt_H(round(f4(1:2, matches43(1,:))), H43);
tr_pt54_4 = find_matching_pt_H(round(f5(1:2, matches54(1,:))), H54);
```

```matlab
% measured correponsding points in range using SIFT matvching
    ms_pt12_2 = round(f2(1:2, matches12(2,:)));
    ms_pt23_3 = round(f3(1:2, matches23(2,:)));
    ms_pt43_3 = round(f3(1:2, matches43(2,:)));
    ms_pt54_4 = round(f4(1:2, matches54(2,:)));

    err_12 = sqrt(sum((tr_pt12_2 - ms_pt12_2).^2, 1));
    err_23 = sqrt(sum((tr_pt23_3 - ms_pt23_3).^2, 1));
    err_43 = sqrt(sum((tr_pt43_3 - ms_pt43_3).^2, 1));
    err_54 = sqrt(sum((tr_pt54_4 - ms_pt54_4).^2, 1));

    inlier_index_pt12 = matches12(:,err_12 < thres_dist12);
    inlier_index_pt23 = matches23(:,err_23 < thres_dist23);
    inlier_index_pt43 = matches43(:,err_43 < thres_dist43);
    inlier_index_pt54 = matches54(:,err_54 < thres_dist54);

    outlier_index_pt12 = matches12(:,err_12 >= thres_dist12);
    outlier_index_pt23 = matches23(:,err_23 >= thres_dist23);
    outlier_index_pt43 = matches43(:,err_43 >= thres_dist43);
    outlier_index_pt54 = matches54(:,err_54 >= thres_dist54);

    save_inlier12{j} = inlier_index_pt12;
    save_inlier23{j} = inlier_index_pt23;
    save_inlier43{j} = inlier_index_pt43;
    save_inlier54{j} = inlier_index_pt54;

    save_outlier12{j} = outlier_index_pt12;
    save_outlier23{j} = outlier_index_pt23;
    save_outlier43{j} = outlier_index_pt43;
    save_outlier54{j} = outlier_index_pt54;

    save_num_inlier12(j) = size(inlier_index_pt12, 2);
    save_num_inlier23(j) = size(inlier_index_pt23, 2);
    save_num_inlier43(j) = size(inlier_index_pt43, 2);
    save_num_inlier54(j) = size(inlier_index_pt54, 2);

    save_H12{j} = H12;
    save_H23{j} = H23;
    save_H43{j} = H43;
    save_H54{j} = H54;


end

[~, max_inlier12_idx] = max(save_num_inlier12);
[~, max_inlier23_idx] = max(save_num_inlier23);
[~, max_inlier43_idx] = max(save_num_inlier43);
[~, max_inlier54_idx] = max(save_num_inlier54);

best_inlier12 = save_inlier12{max_inlier12_idx};
best_inlier23 = save_inlier23{max_inlier23_idx};
best_inlier43 = save_inlier43{max_inlier43_idx};
best_inlier54 = save_inlier54{max_inlier54_idx};

best_outlier12 = save_outlier12{max_inlier12_idx};
best_outlier23 = save_outlier23{max_inlier23_idx};
```

```
best_outlier43 = save_outlier43{max_inlier43_idx};
best_outlier54 = save_outlier54{max_inlier54_idx};

% draw line for corresponding points
figure(1)
drawimage = zeros(row1, col1+col2,3);
drawimage(:,1:col1,:) = rgbimg1;
drawimage(:,col1+1:col1+col2,:) = rgbimg2;
imshow(uint8(drawimage))
hold on
plot(f1(1,:), f1(2,:),'r.');
plot(f2(1,:)+col1, f2(2,:),'r.');
plot([f1(1,best_inlier12(1,:)) ;
f2(1,best_inlier12(2,:))+col1],[f1(2, best_inlier12(1,:)) ; f2(2,
best_inlier12(2,:))],'Color','b','LineWidth',1)
plot([f1(1,best_outlier12(1,:)) ;
f2(1,best_outlier12(2,:))+col1],[f1(2, best_outlier12(1,:)) ; f2(2,
best_outlier12(2,:))],'Color','g','LineWidth',1)

hold off

figure(2)
drawimage = zeros(row2, col2+col3,3);
drawimage(:,1:col2,:) = rgbimg2;
drawimage(:,col2+1:col2+col3,:) = rgbimg3;
imshow(uint8(drawimage))
hold on
plot(f2(1,:), f2(2,:),'r.');
plot(f3(1,:)+col2, f3(2,:),'r.');
plot([f2(1,best_inlier23(1,:)) ;
f3(1,best_inlier23(2,:))+col1],[f2(2, best_inlier23(1,:)) ; f3(2,
best_inlier23(2,:))],'Color','b','LineWidth',1)
plot([f2(1,best_outlier23(1,:)) ;
f3(1,best_outlier23(2,:))+col1],[f2(2, best_outlier23(1,:)) ; f3(2,
best_outlier23(2,:))],'Color','g','LineWidth',1)
hold off

figure(3)
drawimage = zeros(row3, col3+col4,3);
drawimage(:,1:col3,:) = rgbimg3;
drawimage(:,col3+1:col3+col4,:) = rgbimg4;
imshow(uint8(drawimage))
hold on
plot(f3(1,:), f3(2,:),'r.');
plot(f4(1,:)+col3, f4(2,:),'r.');
plot([f3(1,best_inlier43(2,:)) ;
f4(1,best_inlier43(1,:))+col1],[f3(2, best_inlier43(2,:)) ; f4(2,
best_inlier43(1,:))],'Color','b','LineWidth',1)
plot([f3(1,best_outlier43(2,:)) ;
f4(1,best_outlier43(1,:))+col1],[f3(2, best_outlier43(2,:)) ; f4(2,
best_outlier43(1,:))],'Color','g','LineWidth',1)
hold off

figure(4)
drawimage = zeros(row4, col4+col5,3);
drawimage(:,1:col4,:) = rgbimg4;
drawimage(:,col4+1:col4+col5,:) = rgbimg5;
imshow(uint8(drawimage))
hold on
```

```
plot(f4(1,:), f4(2,:),'r.');
plot(f5(1,:)+col4, f5(2,:),'r.');
plot([f4(1,best_inlier54(2,:)) ;
f5(1,best_inlier54(1,:))+col1],[f4(2, best_inlier54(2,:)) ; f5(2,
best_inlier54(1,:))],'Color','b','LineWidth',1)
plot([f4(1,best_outlier54(2,:)) ;
f5(1,best_outlier54(1,:))+col1],[f4(2, best_outlier54(2,:)) ; f5(2,
best_outlier54(1,:))],'Color','g','LineWidth',1)
hold off




% compute homographies
f_H12 = save_H12{max_inlier12_idx};
f_H23 = save_H23{max_inlier23_idx};
f_H43 = save_H43{max_inlier43_idx};
f_H54 = save_H54{max_inlier54_idx};

f_H13 =  f_H12*f_H23;
f_H53 =  f_H54*f_H43;

% make mosaic image
temp1_13 = f_H13*[1 1 1]';
temp1_13 = temp1_13/temp1_13(3);
temp2_13 = f_H13*[col1 1 1]';
temp2_13 = temp2_13/temp2_13(3);
temp3_13 = f_H13*[1 row1 1]';
temp3_13 = temp3_13/temp3_13(3);
temp4_13 = f_H13*[col1 row1 1]';
temp4_13 = temp4_13/temp4_13(3);

temp1_53 = f_H53*[1 1 1]';
temp1_53 = temp1_53/temp1_53(3);
temp2_53 = f_H53*[col5 1 1]';
temp2_53 = temp2_53/temp2_53(3);
temp3_53 = f_H53*[1 row5 1]';
temp3_53 = temp3_53/temp3_53(3);
temp4_53 = f_H53*[col5 row5 1]';
temp4_53 = temp4_53/temp4_53(3);

max_x = max([temp1_13(1) temp2_13(1) temp3_13(1) temp4_13(1)
temp1_53(1) temp2_53(1) temp3_53(1) temp4_53(1)]);
max_y = max([temp1_13(2) temp2_13(2) temp3_13(2) temp4_13(2)
temp1_53(2) temp2_53(2) temp3_53(2) temp4_53(2)]);
min_x = min([temp1_13(1) temp2_13(1) temp3_13(1) temp4_13(1)
temp1_53(1) temp2_53(1) temp3_53(1) temp4_53(1)]);
min_y = min([temp1_13(2) temp2_13(2) temp3_13(2) temp4_13(2)
temp1_53(2) temp2_53(2) temp3_53(2) temp4_53(2)]);


row_mosaic = max_y-min_y+1;
col_mosaic =  max_x-min_x+1;
image_mosaic = zeros(row_mosaic, col_mosaic, 3); % image buffer to
be used to show output image
```

```
image_mosaic = make_image_mosaic(image_mosaic, rgbimg1, min_x,
min_y, f_H13, 1);
image_mosaic = make_image_mosaic(image_mosaic, rgbimg2, min_x,
min_y, f_H23, 2);
image_mosaic = make_image_mosaic(image_mosaic, rgbimg4, min_x,
min_y, f_H43, 3);
image_mosaic = make_image_mosaic(image_mosaic, rgbimg5, min_x,
min_y, f_H53, 4);
% image_mosaic = make_image_mosaic(image_mosaic, rgbimg3, min_x,
min_y, eye(3), 5);
image_mosaic(2-min_y:row5+1-min_y, 2-min_x:col5+1-min_x,:) =
rgbimg3;

figure(5)
imshow(image_mosaic)



% DogLEg

f_H12= dogleg(f_H12, f1(1,best_inlier12(1,:)),
f1(2,best_inlier12(1,:)), f2(1,best_inlier12(2,:)),
f2(2,best_inlier12(2,:)));
f_H23= dogleg(f_H23, f2(1,best_inlier23(1,:)),
f2(2,best_inlier23(1,:)), f3(1,best_inlier23(2,:)),
f3(2,best_inlier23(2,:)));
f_H43= dogleg(f_H43, f4(1,best_inlier43(1,:)),
f4(2,best_inlier43(1,:)), f3(1,best_inlier43(2,:)),
f3(2,best_inlier43(2,:)));
f_H54= dogleg(f_H54, f5(1,best_inlier54(1,:)),
f5(2,best_inlier54(1,:)), f4(1,best_inlier54(2,:)),
f4(2,best_inlier54(2,:)));



f_H13 =  f_H12*f_H23;
f_H53 =  f_H54*f_H43;

% make mosaic image
temp1_13 = f_H13*[1 1 1]';
temp1_13 = temp1_13/temp1_13(3);
temp2_13 = f_H13*[col1 1 1]';
temp2_13 = temp2_13/temp2_13(3);
temp3_13 = f_H13*[1 row1 1]';
temp3_13 = temp3_13/temp3_13(3);
temp4_13 = f_H13*[col1 row1 1]';
temp4_13 = temp4_13/temp4_13(3);

temp1_53 = f_H53*[1 1 1]';
temp1_53 = temp1_53/temp1_53(3);
temp2_53 = f_H53*[col5 1 1]';
temp2_53 = temp2_53/temp2_53(3);
temp3_53 = f_H53*[1 row5 1]';
temp3_53 = temp3_53/temp3_53(3);
temp4_53 = f_H53*[col5 row5 1]';
temp4_53 = temp4_53/temp4_53(3);

max_x = max([temp1_13(1) temp2_13(1) temp3_13(1) temp4_13(1)
temp1_53(1) temp2_53(1) temp3_53(1) temp4_53(1)]);
max_y = max([temp1_13(2) temp2_13(2) temp3_13(2) temp4_13(2)
```

```
temp1_53(2) temp2_53(2) temp3_53(2) temp4_53(2)]);
min_x = min([temp1_13(1) temp2_13(1) temp3_13(1) temp4_13(1)
temp1_53(1) temp2_53(1) temp3_53(1) temp4_53(1)]);
min_y = min([temp1_13(2) temp2_13(2) temp3_13(2) temp4_13(2)
temp1_53(2) temp2_53(2) temp3_53(2) temp4_53(2)]);


row_mosaic = max_y-min_y+1;
col_mosaic =  max_x-min_x+1;
image_mosaic = zeros(row_mosaic, col_mosaic, 3); % image buffer to
be used to show output image



image_mosaic = make_image_mosaic(image_mosaic, rgbimg1, min_x,
min_y, f_H13, 1);
image_mosaic = make_image_mosaic(image_mosaic, rgbimg2, min_x,
min_y, f_H23, 2);
image_mosaic = make_image_mosaic(image_mosaic, rgbimg4, min_x,
min_y, f_H43, 3);
image_mosaic = make_image_mosaic(image_mosaic, rgbimg5, min_x,
min_y, f_H53, 4);
% image_mosaic = make_image_mosaic(image_mosaic, rgbimg3, min_x,
min_y, eye(3), 5);
image_mosaic(2-min_y:row5+1-min_y, 2-min_x:col5+1-min_x,:) =
rgbimg3;

figure(6)
imshow(image_mosaic)
```

Finding matching points

```
function [m_pt] = find_matching_pt_H(pt, H)

num_pt =size(pt, 2);
ori_pt = ones(3, num_pt);
ori_pt(1:2, :) = pt;

h_m_pt = H*ori_pt;
m_pt = [h_m_pt(1,:)./h_m_pt(3,:) ; h_m_pt(2,:)./h_m_pt(3,:)];

end
```

Generate Mosaic image

```
function [out_image_mosaic] = make_image_mosaic(image_mosaic,
ori_img, min_x, min_y, H, num)
num
image_mosaic = double(image_mosaic);
[row_c,col_c,~] = size(image_mosaic);
[row_a,col_a,~] = size(ori_img);
for i=1:row_c
    for j=1:col_c
        jj = j + min_x - 1 ;
        ii = i + min_y - 1;
        temp_pt = H\[jj ii 1]';
        temp_pt = temp_pt/temp_pt(3);
```

```
        % lower interger bound
        l_x = floor(temp_pt(1));
        l_y = floor(temp_pt(2));
        % upper interger bound
        u_x = ceil(temp_pt(1));
        u_y = ceil(temp_pt(2));

        ft_x = temp_pt(1) - l_x;
        ft_y = temp_pt(2) - l_y;


        % bi-linear interpolation
        if(temp_pt(2)<=row_a && temp_pt(2)>=1 && temp_pt(1)<=col_a &&
temp_pt(1)>=1)
            image_mosaic(i, j,:) = (1-ft_x)*(1-ft_y)*ori_img(l_y,
l_x, :) + (1-ft_x)*ft_y*ori_img(u_y, l_x, :) + ft_x*(1-
ft_y)*ori_img(l_y, u_x, :) + ft_x*ft_y*ori_img(u_y, u_x, :);
        end

    end
end

out_image_mosaic = uint8(image_mosaic);

end
```

DogLeg

```
function  final_hk= dogleg(H, ori_x, ori_y, matched_x, matched_y)

% vectorize matrix H to p(=hk)
h0 = [H(1,:)' ; H(2,:)' ; H(3,:)'];
N = size(matched_x,2);
uk = 1;
rk = 5;
hk = h0;
rowdl = 1;

syms h11 h12 h13 h21 h22 h23 h31 h32 h33 x y xhat yhat
fi = jacobian([(h11*x + h12*y + h13)/(h31*x + h32*y + h33), (h21*x +
h22*y + h23)/(h31*x + h32*y + h33)], [h11, h12, h13, h21, h22, h23,
h31, h32, h33]);
err = [xhat - (h11*x + h12*y + h13)/(h31*x + h32*y + h33); yhat -
(h21*x + h22*y + h23)/(h31*x + h32*y + h33)];
k = 1;
while( rowdl>0 )% termination condition
mat_J_f = [];
mat_err = [];
for i=1:N % making MATRIX J_f, err_p
    mat_J_f = [mat_J_f; subs(fi, [h11 h12 h13 h21 h22 h23 h31 h32 h33
x y], [hk' ori_x(i) ori_y(i)])];
    mat_err = [mat_err; subs(err, [h11 h12 h13 h21 h22 h23 h31 h32
h33 x y xhat yhat], [hk' ori_x(i) ori_y(i) matched_x(i)
matched_y(i)])];
```

```
end

delta_gd =
(norm(mat_J_f'*mat_err)/norm(mat_J_f*mat_J_f'*mat_err))*mat_J_f'*mat
_err;
delta_gn = (mat_J_f'*mat_J_f + uk*eye(9))\mat_J_f'*mat_err;

% compute delta
if(norm(delta_gn)<rk)
    delta = delta_gn;
elseif( (norm(delta_gd)<rk) && (rk<norm(delta_gn)) )
    aa = (delta_gn-delta_gd)'*(delta_gn-delta_gd);
    bb = delta_gd'*(delta_gn-delta_gd);
    cc = delta_gd'*delta_gd - rk^2;
    beta = (-bb + sqrt(bb^2-aa*cc)) / (aa);
    delta = delta_gd + beta*(delta_gn - delta_gd);
else
    delta = rk*delta_gd/norm(delta_gd);
end

%update hk
Cpk = mat_err'*mat_err;
hk = hk + delta;

mat_J_f_k_1 = [];
mat_err_k_1 = [];
for i=1:N
    mat_J_f_k_1 = [mat_J_f; subs(fi, [h11 h12 h13 h21 h22 h23 h31 h32
h33 x y], [hk' ori_x(i) ori_y(i)])];
    mat_err_k_1 = [mat_err; subs(err, [h11 h12 h13 h21 h22 h23 h31
h32 h33 x y xhat yhat], [hk' ori_x(i) ori_y(i) matched_x(i)
matched_y(i)])];
end
Cpk_1 = mat_err_k_1'*mat_err_k_1;

rowdl = (Cpk - Cpk_1)/(2*delta'*mat_J_f'*mat_err -
delta'*mat_J_f'*mat_J_f*delta);

% update rk
if(rowdl<0.25)
    rk = rk/4;
elseif(rowdl<=0.75)
    rk = rk;
else
    rk = 2*rk;
end



end

final_hk = [hk(1:3)' ; hk(4:6)' ; hk(7:9)'];

end
```