Assignment 2 Report
Introduction to Computer Vision
Prof. Lee Kyoung Mu
TA: Gyeongsik Moo, Baik Sung Yong David

Nguyen Tuan Nghia
Student ID: 2018-21525
Email: nghianguyentuan@snu.ac.kr
Seoul National University

1 Homography Estimation

In this section, I discuss briefly how to estimate the Homography matrix H. The results will be shown in the next section.

1.1 Preprocessing

At first, I made sure that the working environment is clean. Then, let's set up the option for optimizing Homography matrix. In this work, I implemented two methods to obtain optimal H, including Levenberg-Marquadt cost minimization (LM) and Direct Linear Transform (DLT).

```
%% SETTING UP:
close all;
clear;
clc;

% Optimization options
% Please DO NOT change this
option_none = 0; % No optimization
option_lema = 1; % Levenberg-Marquadt
option_dlto = 2; % Direct Linear Transform

% Change this one
optimz_option = option_dlto;
```

The image list was specified and then loaded to the environment. I performed some preprocessing to the images, including resizing to 256×256 and converting them to gray images. The original color images would be kept for later use.

```
%% PREPROCESSING AND FEATURE EXTRACTION:
image_dir = '../../images/'; % Specify image set directory

% List of images to be loaded
image_list = {'nghia1.jpg';'nghia2.jpg';'nghia3.jpg';'nghia4.jpg';'nghia5.jpg'};
% image_list = {'img1.bmp';'img2.bmp';'img3.bmp';'img4.bmp';'img5.bmp'};

n_image = size(image_list,1); % Count the number of images

images_ori = cell(n_image,1); % Color images should be kept for later use
images = cell(n_image,1); % Gray images will be stored here

feature_f = cell(n_image,1);
feature_d = cell(n_image,1);

for i = 1:n_image
   images_ori{i} = imread(strcat(image_dir, image_list{i})); % Load image
   images_ori{i} = imresize(images_ori{i},[256 256]); % Resize image
   images{i} = single(rgb2gray(images_ori{i})); % Convert to gray image

   % Extract SIFT feature
   % ...
```

```
   % End Feature extraction

end

[h_ori, w_ori] = size(images{1}); % Get image size
```

1.2 Feature Extraction

In this work, SIFT feature was extracted from the images with the help of VL_feat toolbox, via 'vl_sift()' function. In order to get 200 to 300 feature points evenly distributed over each image, I set 'FirstOctave' option to -1. After the extraction, duplicated points would be discarded, then Non-maximal Suppression for SIFT, based on [1] and [2], would be applied to remove the feature points that cause uneven distribution. The number of feature points extracted is usually exactly 300. However, one should not expect feature points to appear uniformly on the region that contains plain texture such as (constant) blue sky or green grass field.

```
for i = 1:n_image

   % Preprocess image
   % ...
   % End preprocessing

   [f,d] = vl_sift(images{i},'FirstOctave',-1);

   % Remove duplicated points
   [~,sorted_idx] = sort(f(1,:));
   f = f(:,sorted_idx);
   d = d(:,sorted_idx);

   filtered_idx = 1;
   for j = 2:size(f,2)
      if ((f(1,j) - f(1,filtered_idx(end)) == 0) && (f(2,j) - f(2,filtered_idx(end)) == 0))
         continue;
      end
      filtered_idx = [filtered_idx, j];
   end

   feature_f{i} = f(:,filtered_idx);
   feature_d{i} = d(:,filtered_idx);

   % Apply Non-maximum Suppression to get better distribution
   fid = anms(feature_f{i},300); % Keep upto 300 points
   feature_f{i} = feature_f{i}(:,fid);
   feature_d{i} = feature_d{i}(:,fid);
end
```

1.3 Feature Matching

Feature matching was carried out with the help of VL_feat toolbox as well, via 'vl_ubcmatch()' function. I noticed that one feature point of the first image was sometimes matched with multiple points in the second image. Therefore, a simple duplication removal was performed to clean those matches.

```
%% FEATURE MATCHING:
match = cell(n_image-1,1); % Matches (index) will be stored here
H = cell(n_image-1,1); % Homography matrix will be stored here
d_sum = cell(n_image-1,1); % Distance measurement will be stored here
```

```
for i = 1:n_image-1
   fprintf('Processing Image Pair: %d and %d\n',i,i+1);

   fprintf('Matching Key Points\n');

   [match{i}, score] = vl_ubcmatch(feature_d{i},feature_d{i+1});

   % Remove one-point-multiple-match matches
   [score, sorted_idx] = sort(score);
   match{i} = match{i}(:,sorted_idx);
   for fil_dim = 1:2
      [~,sid] = sort(match{i}(fil_dim,:));
      match{i} = match{i}(:,sid);
      filtered_idx = 1;
      for j = 2:size(match{i},2)
         if match{i}(fil_dim,j) == match{i}(fil_dim,filtered_idx(end))
            continue;
         end
         filtered_idx = [filtered_idx j];
      end
      match{i} = match{i}(:,filtered_idx);
   end

   % Estimate H
   % ...
   % End estimation
end
```

1.4 Homography Estimation using RANSAC

After getting a list of putative correspondences between two images I_i and I_j, RANSAC and DLT method were used to estimate Homography matrix H_{ij}. I implemented a function, named 'HbyRANSAC()', that takes 'feature_i', 'feature_j', 'match_list' and 'optimize_option' as four arguments and returns 'H_best' and new 'match_list'. 'feature_i' and 'feature_j' are lists of feature points in images I_i and I_j respectively while 'match_list' contains the indices of all points that form matches. 'H_best' is the best estimated Homography matrix returned from this function, together with new 'match_list' after Guided Matching.

```
for i = 1:n_image-1
   % Match feature points
   % ...
   % End matching

   % Homography estimation using RANSAC and DLT
   fprintf('Estimating H using RANSAC\n');
   [H{i}, match{i}] = HbyRANSAC(feature_f{i}, feature_f{i+1},match{i}, optimz_option);

   % Output some results
   % ...
   % End Outputting
end
```

1.5 Constructing Panorama image

I implemented a simple warping function, named 'warping()', to project an image to the plane of another one by corresponding Homography matrix H. Instead of using 'maketform()' and 'imtransform()' functions which is not recommended by MATLAB, I used 'projective2d()' and 'imwarp()' functions to directly warp

color images to the plane of center image. Backward projection was applied to improve quality of the final image. The masks were used to blend respective warped images onto panorama plane. Blending could be done with the help of MATLAB Computer Vision package, however, I have written another manual blending script in case you do not use this package.

```
function [warped_img, mask] = warping(img_i, H, view)
tform = projective2d(H');
warped_img = imwarp(img_i, tform, 'OutputView',view);
mask = imwarp(true(size(img_i)), tform, 'OutputView', view);
```

1.6 Optimal Homography

In this part, I discuss three techniques to improve the quality of Homography matrix.

Minimizing ML cost function with Levenberg-Marquardt Algorithm:
The goal of this technique is to re-estimate Homography matrix from all inlier obtained after the basic estimation. I used 'lsqcurvefit()' function to minimize symmetric transfer error, denoted 'ydata'. The initial state of Homography matrix is the one obtained after using basic estimation. After running this minimization, a new Homography matrix would be created that fits the inliers better.

```
% Optimal Homography estimation
options = optimoptions('lsqcurvefit','Algorithm','levenberg-marquardt','Display','off');

while 1
   n_match = size(match_list,2);

   feature_f_match_1 = [feature_i(1:2,match_list(1,:)); ones(1,n_match)];
   feature_f_match_2 = [feature_j(1:2,match_list(2,:)); ones(1,n_match)];

   h = H_best';
   h = h(:);

   n_inlier = n_match;

   xdata = [feature_f_match_1;feature_f_match_2];
   ydata = zeros(1,n_inlier);

   % Minimize cost function using Levenberg-Marquardt Algorithm
   fprintf('--Levenberg-Marquardt Algorithm\n');
   H_LM = lsqcurvefit(@symmetric_transfer_dist,h,xdata,ydata,[],[],options);
   Ht = reshape(H_LM,[3,3]);
   Ht = Ht';

   % Find more matches using Guided Matching
   % ...
   % End Guided Matching
end
```

Direct Linear Transform on all inliers:
This optimization method is similar to one that is carried out to estimate basic Homography matrix. During optimization, all inliers will be used to estimate Homography matrix instead of only 4 points as before.

```
% Optimal Homography estimation
while 1
   n_match = size(match_list,2);
```

```
  xi = [feature_i(1:2,match_list(1,:)); ones(1,n_match)];
  xj = [feature_j(1:2,match_list(2,:)); ones(1,n_match)];

  A = [];

  for i = 1:n_match
    A = [A; 0 0 0 (-xi(:,i)') (xj(2,i) * (xi(:,i)'))];
    A = [A; xi(:,i)' 0 0 0 (-xj(1,i) * (xi(:,i)'))];
  end

  fprintf('--DLT Algorithm\n');
  [~,~,V] = svd(A,0);
  h = V(:,end);
  Ht = reshape(h, [3 3]);
  Ht = Ht';

  % Find more matches using Guided Matching
  % ...
  % End Guided Matching
end
```

Guided Matching:
New Homography matrix obtained after applying optimization fits the current inlier set well. Using Guided Matching technique further increases the number of matches among the whole set of feature points. To determine if two points match with each other, I also used symmetric transfer error for the distance measurement.

```
while 1
  % Re-estimate H
  % ...
  % End Re-estimation

  % Find more matches using Guided Matching
  fprintf('--Guided Matching\n');

  alone_list_1 = [];
  alone_list_2 = [];

  for ii = 1:size(feature_i,2)
    if find(match_list(1,:)==ii)
      continue;
    else
      alone_list_1 = [alone_list_1 ii];
    end
  end

  alone_point_1 = feature_i(1:2,alone_list_1);
  alone_point_1 = [alone_point_1;ones(1,size(alone_point_1,2))];
  transformed_1 = Ht * alone_point_1;
  transformed_1 = [transformed_1(1,:) ./ transformed_1(3,:);transformed_1(2,:) ./
      transformed_1(3,:);transformed_1(3,:) ./ transformed_1(3,:)];

  for ii = 1:size(feature_j,2)
    if find(match_list(2,:)==ii)
      continue;
    else
      alone_list_2 = [alone_list_2 ii];
```

```
    end
  end

  alone_point_2 = feature_j(1:2,alone_list_2);
  alone_point_2 = [alone_point_2;ones(1,size(alone_point_2,2))];
  transformed_2 = Ht \ alone_point_2;
  transformed_2 = [transformed_2(1,:) ./ transformed_2(3,:);transformed_2(2,:) ./
      transformed_2(3,:);transformed_2(3,:) ./ transformed_2(3,:)];

  new_pair = [];
  for ii = 1:size(alone_point_1,2)
    d_best = inf;
    tmp_pair = [];
    for jj = 1:size(alone_point_2,2)
      xi = alone_point_1(:,ii);
      xj = alone_point_2(:,jj);
      xi_t = transformed_1(:,ii);
      xj_t = transformed_2(:,jj);

      d1 = sum((xi - xj_t) .^ 2);
      d2 = sum((xj - xi_t) .^ 2);
      d = d1 + d2;
      if (d < 1.25) && (d < d_best)
        d_best = d;
        tmp_pair = [alone_list_1(ii);alone_list_2(jj);d_best];
      end
    end
    new_pair = [new_pair tmp_pair];
  end

  H_best = Ht;

  if isempty(new_pair)
    fprintf('----No new matches...\n');
    break; % No new matches, so optimization is done
  end
  fprintf('----%d new matches...\n', size(new_pair,2));

  match_list = [match_list new_pair(1:2,:)];
end
fprintf('->Done Optimizing...\n');
```

If there are new matches, Homography matrix will be re-estimated from new inlier set by applying Levenberg-Marquardt minimization or DLT again. The loop ends when there is no new match.

2 Results and Discussion

My code was configured to output pairs of images, all feature points and matches, and the final panorama image. The console also outputs the Homography matrices and total symmetric transfer errors over all inliers.

2.1 Optimization: None

Without optimization, the constructed panorama image was inconsistent. It might look good sometimes, but it might also look pretty bad. All results are illustrated in Fig. 1, 2, 3, 4, 5.

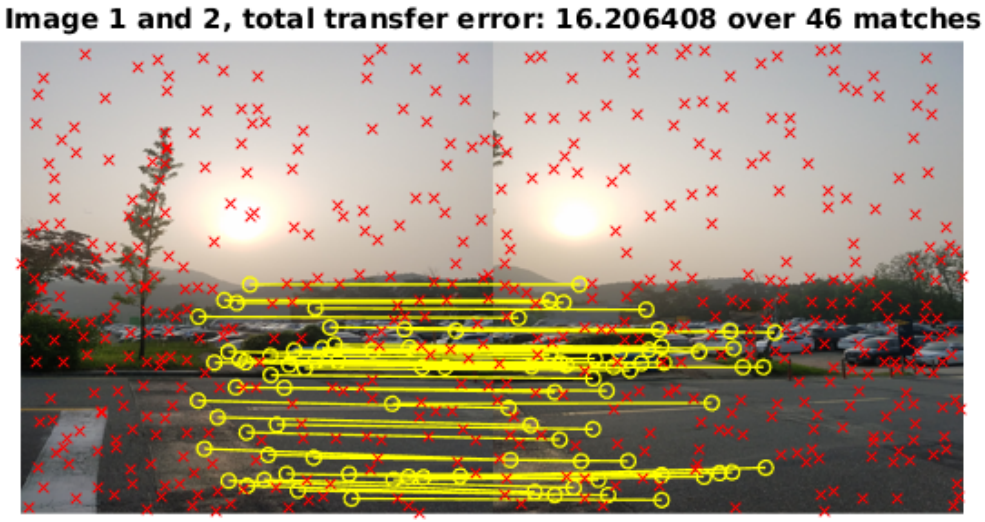

Figure 1: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

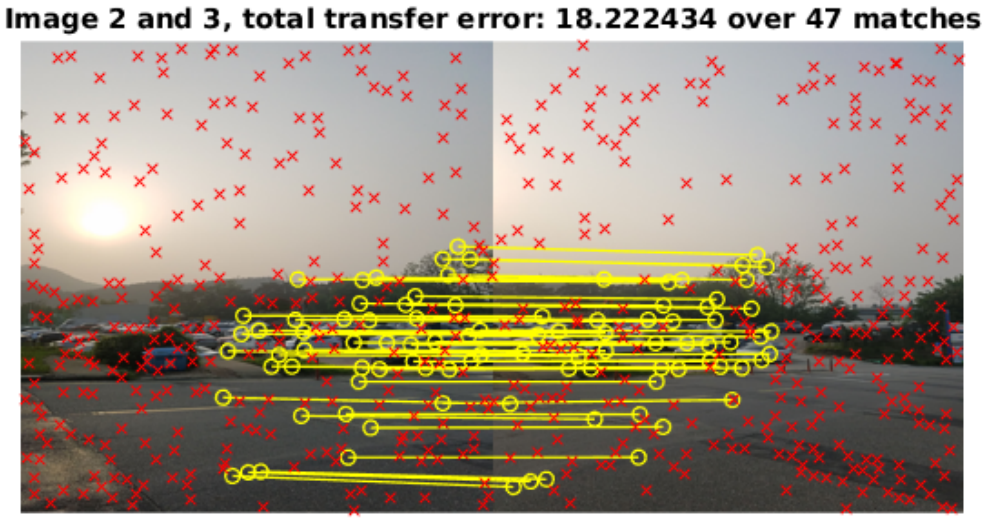

Figure 2: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

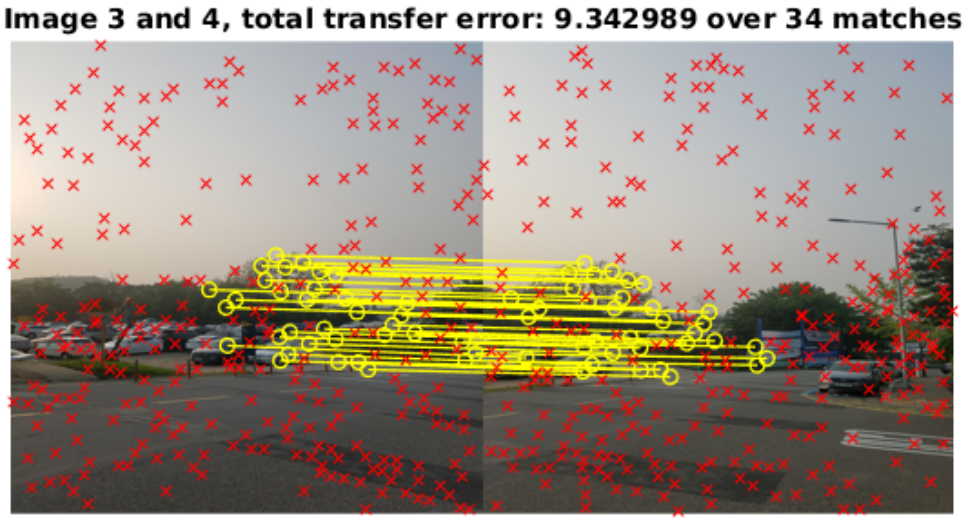

Figure 3: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

Figure 4: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

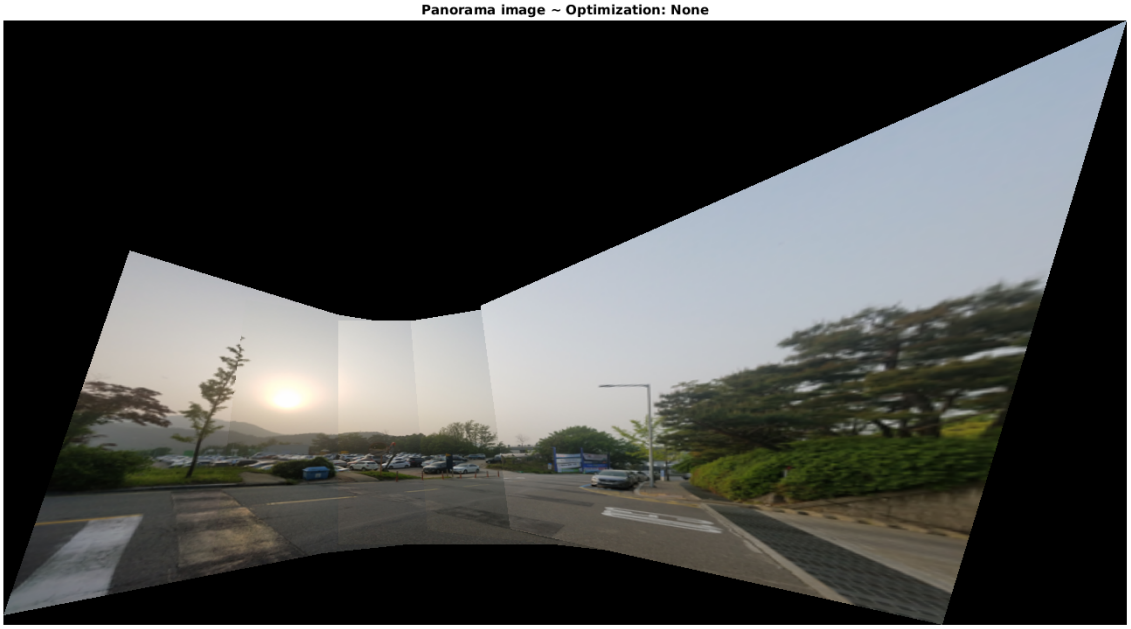

Figure 5: Panorama image without optimization

2.2 Optimization: Levenberg-Marquardt

Using Levenberg-Marquardt and Guided Matching made the panorama image more consistent. The number of putative correspondences also increased. All results are illustrated in Fig. 6, 7, 8, 9, 10.

2.3 Optimization: Direct Linear Transform

Using Direct Linear Transform on all inliers and Guided Matching made the panorama image more consistent. The number of putative correspondences also increased. The difference between using Levenberg-Marquardt minimization method and this one is unclear. All results are illustrated in Fig. 11, 12, 13, 14, 15.

2.4 Common issue

Moving Objects: Moving objects causes artifact in panorama image. It is due to the fact that the relative positions of those objects to others change between images. Even applying optimizations did not solve this problem. This problem is illustrated in Fig. 5, 10, 15 (notice the tall tree on the left).

References

[1] R. Song and J. Szymanski, "Well-distributed sift features," Electronics letters, vol. 45, no. 6, pp. 308–310, 2009.

[2] M. Brown, R. Szeliski, and S. Winder, "Multi-image matching using multi-scale oriented patches," in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, pp. 510–517, IEEE, 2005.

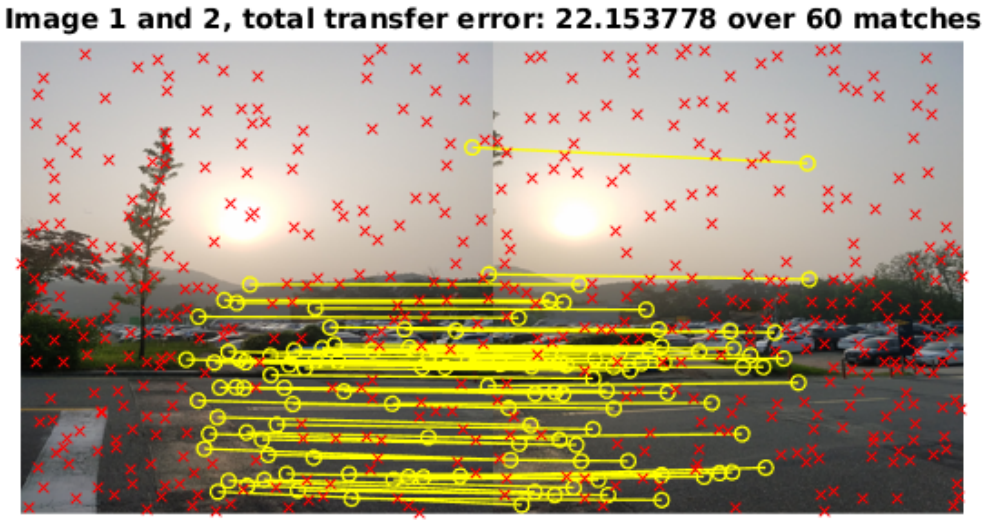

Figure 6: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

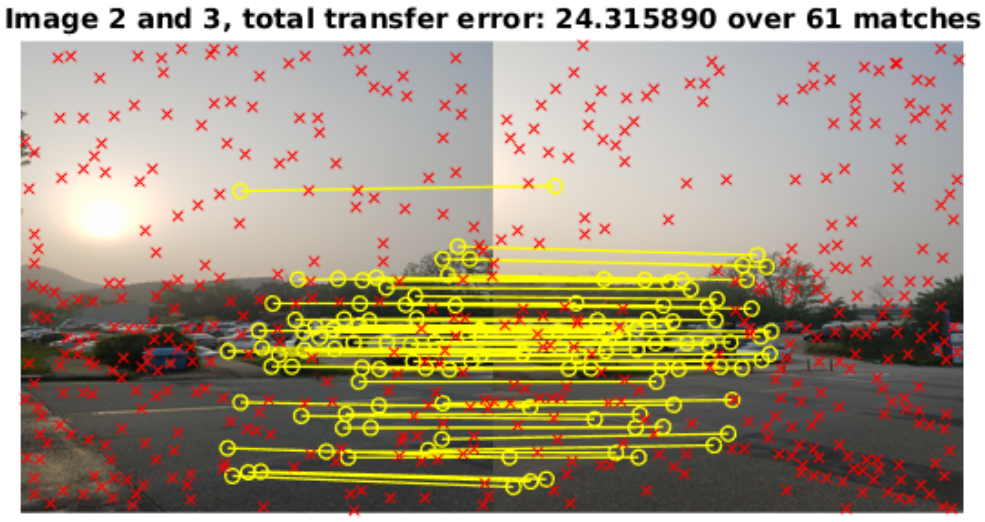

Figure 7: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

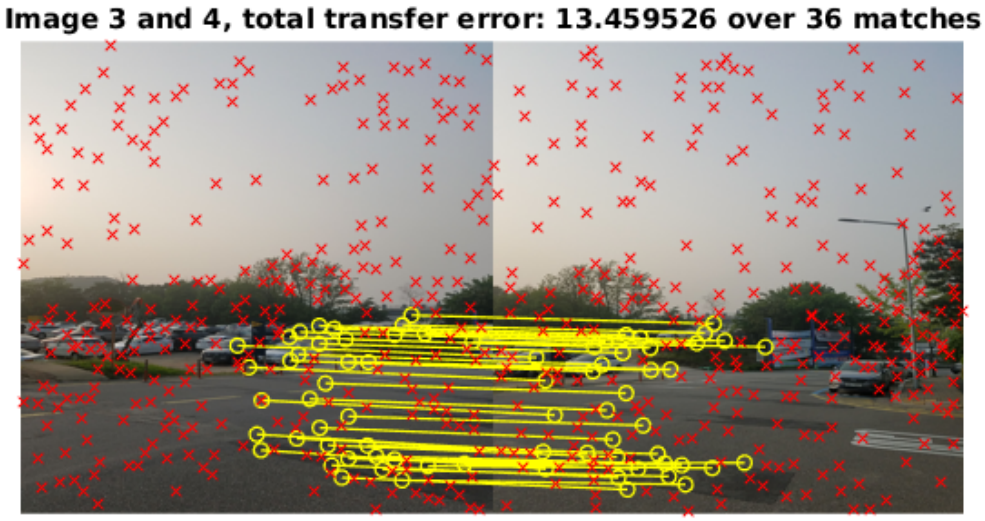

Figure 8: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

Figure 9: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

Figure 10: Panorama image with Levenberg-Marquardt optimization

Figure 11: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

Figure 12: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

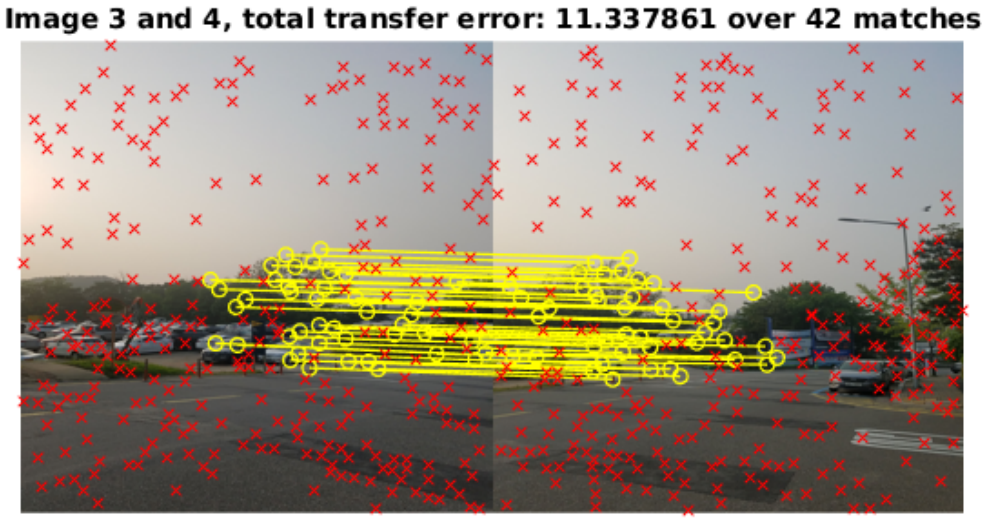

Figure 13: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

Figure 14: Matched feature points (yellow 'o' points) and outliers (red 'x' points)

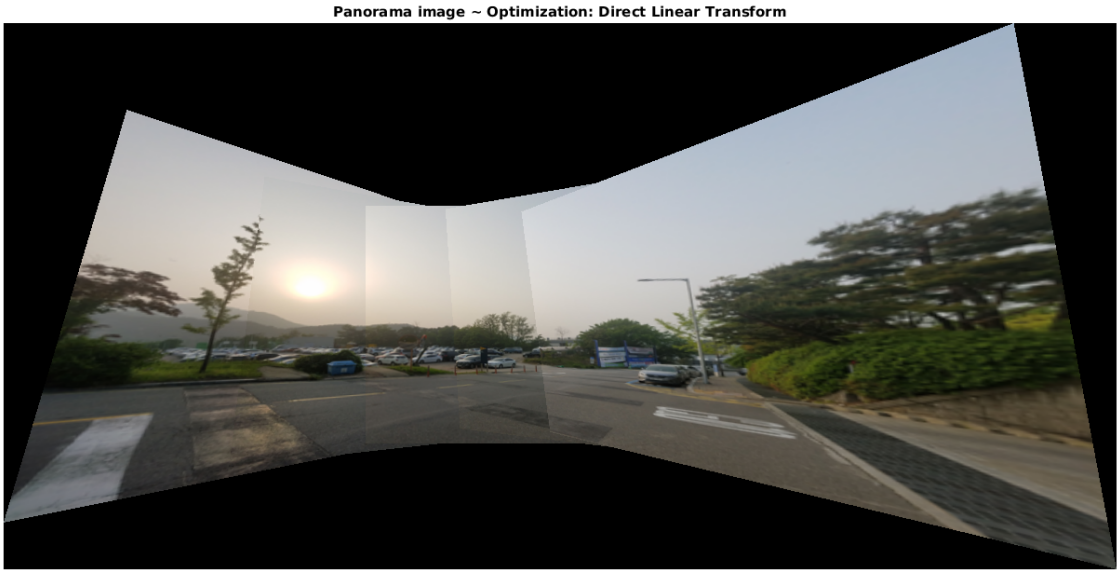

Figure 15: Panorama image with Direct Linear Transform optimization