ME491(B)
Homework 4 - Image Alignment

Herman Kolstad Jakobsen
20196493

April 25, 2020

Contents

1 Introduction

Image alignment is the process of matching one image with another image. There exists two types of approaches. Direct (pixel-based) alignment searches for alignment where most pixels between the two images agree, while feature-based alignment searches for alignment where extracted features agree. There exists many applications for image alignment, such as tracking objects on video and motion analysis. Image alignment is much used in computer vision.

In this homework, direct alignment will be used to align two pairs of photos taken at KAIST. The procedure of image alignment will first be described, before presenting the results. The code used to solve the assignment is shown in the last section.

2 Direct image alignment

Direct image alignment has several similarities to calculating the projective matrix $\mathbf{P}$ used in camera calibration. Instead of estimating a projective matrix, a homography $\mathbf{H}$ between a pair of images is found. The homography is then used to warp one image onto the other.

The homography between two images can be estimated using the Direct Linear Transformation (DLT) algorithm, as done in the previous homework with the projective matrix. The transformation between two images is given by

$$\mathbf{x}_i' = \mathbf{H}\mathbf{x}_i \tag{1}$$

where $\mathbf{x}_i'$ and $\mathbf{x}_i$ are point correspondences between the two images. Note that the points are given in homogeneous coordinates. Applying the cross product on both sides, and then doing some rearranging, the equation can be expressed as

$$\begin{bmatrix} \mathbf{0}^T & -w_i'\mathbf{x}_i^T & y_i'\mathbf{x}_i^T \\ w_i'\mathbf{x}_i^T & \mathbf{0}^T & -x_i'\mathbf{x}_i^T \\ -y_i'\mathbf{x}_i^T & x_i'\mathbf{x}_i^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{bmatrix} = \mathbf{0} \tag{2}$$

with $\mathbf{h}^i$ being a row of the homography matrix and of size 3×1. Only two of the equations in eq. (2) are linearly independent. Each point correspondence will therefore yield two equations. The set of equations for one point correspondence can therefore be reduced to

$$\begin{bmatrix} \mathbf{0}^T & -w_i'\mathbf{x}_i^T & y_i'\mathbf{x}_i^T \\ w_i'\mathbf{x}_i^T & \mathbf{0}^T & -x_i'\mathbf{x}_i^T \end{bmatrix} \begin{bmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{bmatrix} = \mathbf{0} \tag{3}$$

which has the form $\mathbf{A}_i\mathbf{h} = \mathbf{0}$. For n point correspondences, the set of equations becomes $\mathbf{A}\mathbf{h} = \mathbf{0}$. Here, $\mathbf{A}$ is built from the matrix rows $\mathbf{A}_i$ contributed from each correspondence. It is worth noting that the data should be normalized before constructing the $\mathbf{A}$ matrix. Normalization of the data makes the equations invariant to scale and coordinate origin.

The homography matrix $\mathbf{H}$ has 8 degrees of freedom since it is only defined up to scale. Hence, it is necessary with four point correspondences to fully constrain $\mathbf{H}$. If only four points are specified, then an exact solution of the homography matrix is possible. Due to measurement noise when specifying the points, it is common to acquire more than four point correspondences. The homography matrix is then found by solving the following optimization problem

$$\min \|\mathbf{Ah}\| \quad \text{s.t.} \quad \|\mathbf{h}\| = 1 \tag{4}$$

The solution of the optimization problem is the unit singular vector corresponding to the smallest singular value of $\mathbf{A}$.

After estimating the homography matrix, the matrix can be used to warp one image onto the other. There are two types of image warping; forward warping and inverse warping.

In forward warping, each pixel x is mapped to its corresponding location $h(x)$ using the transformation h. This approach has the downside of mapped pixels being able to land "between" two pixels in the grid of the target image. As a result, some pixels in the grid of the target image may get no pixels mapped to its place, while other pixels will get several pixels mapped to its place. The problem is solved by the mapped pixel adding a "contribution" to several pixels in the neighbourhood of the pixel it is mapped to.

In inverse warping, each pixel in the target image grid is sampled from the original image using h^{-1}. A known problem for inverse warping is that a pixel may be sampled from "between" two pixels in the original image. Interpolation between the neighbouring pixels in the original image is used to solve this problem.

3 Results

Ten point correspondences were arbitrary chosen for each image pair. The chosen points are illustrated in fig. 1 and fig. 2.

Based on the point correspondences, the homography matrices were estimated to be

$$\mathbf{H}_1 = \begin{bmatrix} 0.6124 & 0.1319 & -1020.8 \\ 0.0013 & 0.6622 & -972.71 \\ 4.70 \cdot 10^{-5} & 6.98 \cdot 10^{-5} & 0.2875 \end{bmatrix} \tag{5}$$

$$\mathbf{H}_2 = \begin{bmatrix} 0.4515 & 0.2801 & -6.5012 \\ -0.3512 & 0.4203 & 1058.2 \\ -1.15 \cdot 10^{-5} & -2.91 \cdot 10^{-5} & 0.6483 \end{bmatrix} \tag{6}$$

The first image in each image pair was then warped using inverse warping. A comparison between the warped image and the desired image is shown in fig. 3 and fig. 4. Due to the warping, the resolution between the warped image and the desired image differs.

Overall, the image alignment can be concluded to be fairly good. For instance, in fig. 3 the balconies outside the windows and the rail on the roof are

Figure 1: Chosen points for first image pair.

Figure 2: Chosen points for second image pair.

(a) Warped image.

(b) Desired image

Figure 3: Comparison of warped image and desired image for the first image pair. The homography $\mathbf{H}_1$ was used in the warping.

(a) Warped image.

(b) Desired image.

Figure 4: Comparison of warped image and desired image for the second image pair. The homography $\mathbf{H}_2$ was used in the warping.

parallel to the balconies and rail in the desired image. This is also the case in fig. 4. In order to obtain a measurement of the accuracy, the euclidean distance between the ten point correspondences for the warped image and the desired image was summed up for each image pair. This gave the following result

$$e_1 = 11.4820 \quad \text{and} \quad e_2 = 55.5115 \tag{7}$$

The first warped image differs with a total of approximately 12 pixels, while the second warped image differs with about a total of 56 pixels.

An easy way to improve the accuracy is to include more point correspondences. The number of ten correspondences were arbitrary chosen, and it might happen that increasing this amount could yield a significant improvement in accuracy. A rule of thumb for estimating the projective matrix $\mathbf{P}$ in camera calibration is to have five times as many equations as unknowns. It could therefore be a good choice to choose 20 point correspondences for each image pair.

Other, more complex, ways to improve accuracy is for instance to use feature-based alignment. Gathering point correspondences manually is extremely prone to noise, and aligning images with the use of features can reduce this unwanted phenomenon. In addition, pixel-based alignment can then later be used to verify the result from the feature-based alignment.

Lastly, the estimated homography matrices can be further optimized. After the initial estimation, the matrices can be optimized to minimize the euclidean error between the warped and the desired image, ultimately yielding better accuracy.

4 Code

The main code for the image alignment implementation is shown in listing 1. The code uses three helper functions that normalizes data, constructs the **A** matrix and calculates the error, respectively. The helper functions are shown in listing 2, listing 3 and listing 4.

Listing 1: Code for image alignment.

```
clear variables;
%% Corresponding image points
% Origin (0,0) is top left corner.

% Image points for homography 1
img_h1_1 = [
    1695    1960    1;
    2213    2024    1;
    2195    2377    1;
    1681    2325    1;
    1644    3223    1;
    1584    3198    1;
    1635    3472    1;
    1576    3433    1;
    2769    2781    1;
    2788    3235    1;
    ]';

img_h1_2 = [
    547     649     1;
    1127    695     1;
    1146    1085    1;
    597     1078    1;
    697     1972    1;
    635     1960    1;
    722     2190    1;
    660     2167    1;
    1702    1427    1;
    1728    1820    1;
    ]';

% Image points for homography 2
img_h2_1 = [
    475     783     1;
    930     1024    1;
    833     1215    1;
    376     999     1;
    1080    2774    1;
    1586    2833    1;
    1444    3046    1;
    929     3009    1;
    342     2200    1;
    33      1350    1;
    ]';

img_h2_2 = [
```

```
47      696     1971    1;
48      1152    1912    1;
49      1170    2113    1;
50      722     2190    1;
51      2265    3324    1;
52      2750    3089    1;
53      2759    3374    1;
54      2277    3632    1;
55      1331    3209    1;
56      625     2651    1;
57      ]';
58
59
60  %% Estimate homography matrix (DLT algorithm)
61  % Normalization is performed in order to make DLT algorithm ...
        invariant to
62  % scale and coordinate origin.
63  [norm_img_h1_1, T_img_h1_1] = data_normalization(img_h1_1);
64  [norm_img_h1_2, T_img_h1_2] = data_normalization(img_h1_2);
65  [norm_img_h2_1, T_img_h2_1] = data_normalization(img_h2_1);
66  [norm_img_h2_2, T_img_h2_2] = data_normalization(img_h2_2);
67
68  A1 = construct_A_matrix(norm_img_h1_1, norm_img_h1_2);
69  A2 = construct_A_matrix(norm_img_h2_1, norm_img_h2_2);
70
71  [¬,¬,V1] = svd(A1);
72  [¬,¬,V2] = svd(A2);
73
74  norm_h1 = V1(:,9);   % Vector containing homography matrix elements
75  norm_h2 = V2(:,9);
76
77  norm_H1 = reshape(norm_h1,3,3)';   % Homography matrix with ...
        normalized elements
78  norm_H2 = reshape(norm_h2,3,3)';
79
80  H1 = T_img_h1_2\norm_H1*T_img_h1_1; % Backward transform to get ...
        original data
81  H2 = T_img_h2_2\norm_H2*T_img_h2_1;
82
83
84  %% Checking result
85  % Can compare pers1 to img_h1_2 and pers2 to img_h2_2.
86  pers1 = H1*img_h1_1;
87  for i = 1:size(pers1')
88     pers1(:,i) = pers1(:,i)/pers1(3,i);
89  end
90
91  pers2 = H2*img_h2_1;
92  for i = 1:size(pers2')
93     pers2(:,i) = pers2(:,i)/pers2(3,i);
94  end
95
96  % Calculates sum(sqrt((truth_xi-align_xi)^2 +(truth_yi-align_yi)^2))
97  error1 = error_measurement(H1, img_h1_1, img_h1_2);
98  error2 = error_measurement(H2, img_h2_1, img_h2_2);
99
100
```

```
%% Warping
% Homography 1
img1 = imread('hw04_h1_img1.jpg');
tform1 = projective2d(H1');
warped1 = imwarp(img1, tform1);
imshow(warped1);

% Homography 2
img2 = imread('hw04_h2_img1.jpg');
tform2 = projective2d(H2');
warped2 = imwarp(img2, tform2);
imshow(warped2);


%% Save images
imwrite(warped1, 'warp1.jpg');
imwrite(warped2, 'warp2.jpg');
```

Listing 2: Code for normalizing data.

```
function [norm_data, T_data] = data_normalization(data) % ...
    Isotropic scaling
    % Asuming data (image points) are given as [x; y; w].
    [~,number_points] = size(data);
    mean_data = mean(data,2); % Mean of each row (coordinate)
    dist_sum = 0;
    for i = 1:number_points
        dist_sum = dist_sum+norm(data(:,i)-mean_data)^2;
    end
    scale_2 = sqrt(dist_sum/(2*number_points));
    T_data = [
        1/scale_2, 0, -1/scale_2*mean_data(1);
        0, 1/scale_2, -1/scale_2*mean_data(2);
        0, 0, 1
        ];
    norm_data = T_data*data;
end
```

Listing 3: Code for constructing **A** matrix.

```
function A = construct_A_matrix(img, pers_img)
    [~,number_points] = size(img);
    A = zeros(2*number_points, 9);
    for i = 1:number_points
        % First equation for each pair of points
        A((i-1)*2+1,:) = [
            0, 0, 0, ...
            (-pers_img(3,i)*img(:,i))', ...
            (pers_img(2,i)*img(:,i))'
        ];
        % Second equation for each pair of points
        A((i-1)*2+2,:) = [
            (pers_img(3,i)*img(:,i))', ...
            0, 0, 0, ...
```

```
            (-pers_img(1,i)*img(:,i))'
        ];
    end
end
```

Listing 4: Code for calculating error between warped image and desired image.

```
function error = error_measurement(H, img, truth_img)
    % Calculates sum(sqrt((truth_xi-align_xi)^2 ...
        +(truth_yi-align_yi)^2))
    error = 0;
    [~,img_size] = size(img);
    for i = 1:img_size
        align_pt = H*img(:,i);
        align_pt = align_pt/align_pt(3);
        truth_pt = truth_img(:,i);
        diff = ...
            sqrt((truth_pt(1)-align_pt(1))^2+(truth_pt(2)-align_pt(2))^2);
        error = error + diff;
    end
end
```