Assignment 6
Signal & Image Processing

Casper B. Hansen
University of Copenhagen
Department of Computer Science
fvx507@alumni.ku.dk

October 14, 2014

Abstract

In this assignment we look at transformations of images in the homogeneous coordinate space. Furthermore we look at how we can map from one point set over into another using techniques that will estimate such transformations.

Contents

1 Experiments on Translation

1.1 Translation matrix filter

The 2-dimensional homogeneous translation matrix is given by

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, \text{ if we let } w = 1. \tag{1}$$

For a translation matrix one pixel to the right, we let $t_x = 1$ and $t_y = 0$. And the corresponding filter kernel matrix would then be given by

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2}$$

It is easy to express $\widetilde{I}(x, y)$ by $I(x, y)$; each pixel in the transformed image $\widetilde{I}$ corresponds to the pixel immediately to the left of it of I. That is

$$\widetilde{I}(x, y) = I(x - 1, y) \tag{3}$$

An example of applying this simple 1-pixel translation kernel is given in appendix A.1.

1.2 Generalized translation

The code for this assignment is given in appendix A.2, and the generalized translation function is given in appendix B.1. An example rendition of its effects is shown below in figure 1.

Figure 1: Example translation

I've chosen to use a wrapping boundary condition. I found this to be the most desirable boundary condition for translation, although I could have simply ignored pixels outside the boundaries of the image.

1.3 Fourier transform translation

The code used to produce figure 2 below can be review in appendix A.3.

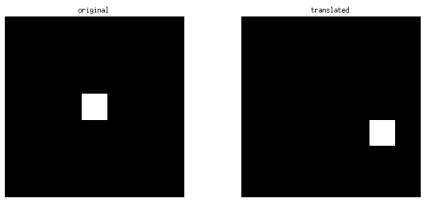

Figure 2: Fourier translation using $\vec{t} = (2, 1)^T$

As I couldn't do this using a more direct approach I sought some help on the internet, and so this solution code is largely inspired by an answer[1] on StackOverflow.

1.4 Non-integer fourier translation

The code used to produce figure 3 below can be review in appendix A.4.

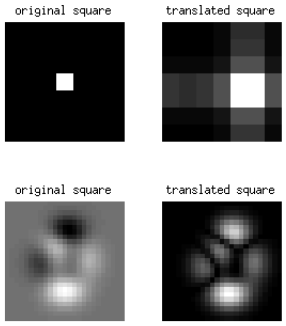

Figure 3: Fourier translation using $\vec{t} = (1.5, 0.5)^T$

As can be seen from the above figure, non-integer translations produces artifacts in the image, as apparent of the square image, and causes inversions as well, as evident of the `peak` produced image.

2 Procrustes Transformations

2.1 Free parameters and minimum N

I see no extraordinary conditions in the described problem, and so I would expect it to exhibit exactly the same free parameters as in the general case; namely 4 free parameters; α, γ, λ_1 and λ_2, with which we can express the entire transformation matrix by

$$T = \begin{bmatrix} \alpha & \gamma & \lambda_1 \\ -\gamma & \alpha & \lambda_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

which decomposes into three matrices (translation X, rotation R and scaling S);

$$X = \begin{bmatrix} 1 & 0 & \beta_x \\ 0 & 1 & \beta_y \\ 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad S = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where

$$\alpha = s\cos\theta \quad \gamma = s\sin\theta \quad \lambda_1 = s(\beta_x\cos\theta + \beta_y\sin\theta) \quad \lambda_2 = s(-\beta_x\sin\theta + \beta_y\cos\theta) \tag{6}$$

[1] http://www.stackoverflow.com/questions/25827916/matlab-shifting-an-image-using-fft

Because the minimization of the procrustes translation vector $\vec{t}$ is merely a negated average of the input vectors there is no dependence on N for this parameter. Likewise, so is the case of scaling; if we let $N = 1$, then s becomes 1, which produces an identity matrix, not transforming anything. I failed to find a mathematical argument as to the required N in order to produce the rotation matrix, can however state that I would imagine that since we must build the two eigenvectors U and V, we should need at least $N = 2$.

2.2 Finding procrustes parameters

In this assignment I've commented out the interactive code, which asks the user to pick out four sequential points for each of the images presented. These have been made static for a particular set of points that I've picked out myself — a rendition of the result is shown below. The code to produce this can be reviewed in appendix A.5.

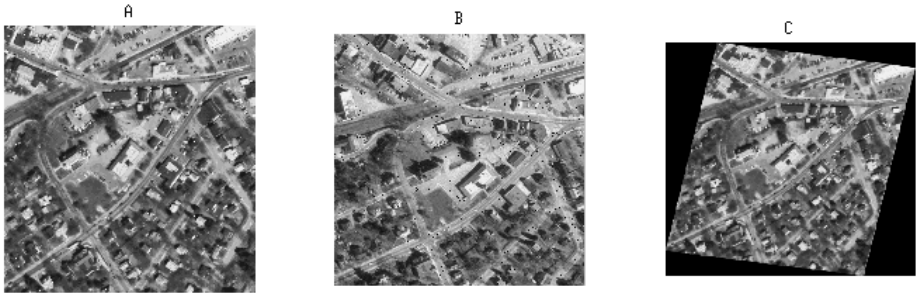

Figure 4: Rendition of the pre-selected point outcome

From the output of running `procrustes` on the input and target points I found that the scale s is 1.0977, whilst the translation vector $\vec{t}$ was $(-88.3279, 10.6171)^T$ and the rotation matrix was given by

$$\begin{bmatrix} 0.9820 & -0.1888 \\ 0.1888 & 0.9820 \end{bmatrix}$$

From these we can built the corresponding homogeneous matrices for translation X, rotation R and scaling S, and computing the product transformation matrix $T = SRX$ we can then read out the procrustes parameters one by one.

$$\begin{bmatrix} 1.0977 & 0 & 0 \\ 0 & 1.0977 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.9820 & -0.1888 & 0 \\ 0.1888 & 0.9820 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -88.3279 \\ 0 & 1 & 10.6171 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1.0780 & -0.2073 & -76.9113 \\ 0.2073 & 1.780 & -113.5232 \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

This gives us the full transformation matrix, and by equation (4) we can now read out the parameters from this matrix.

$$\alpha = 1.0780 \qquad \gamma = -0.2073 \qquad \lambda_1 = -76.9113 \qquad \lambda_2 = -113.5232 \quad (8)$$

4 Affine and Projective Alignments

4.1 Homogeneous coordinate space

To show why we prefer the homogeneous coordinate space over Euclidean coordinate space, we consider a homogeneous point p in n-nary space and a series of transformations $M_0, M_1, \ldots, M_m$ that we wish to apply to point p in that particular order.

Applying a transformation should be of the form $p_b = M_i p_a$, taking an input vector p_a producing an ouput vector p_b. This allows us to reason that since $M_i a$ produces a vector, we can apply the exact same form to the following transformation — that is, $p_c = M_{i+1} p_b$. By induction it then follows that $p_{final}(M_m(M_{m-1}(\ldots(M_0)\ldots)p_{original}$.

We can further argue that since $A(BC) = (AB)C$, we then have that the concatenated transformations can be expressed by $(M_m(M_{m-1}(\ldots(M_0)\ldots)p = (M_m M_{m-1}\ldots M_0)p$ and hence, we can precalculate the entire sequence of transformations into one single matrix, which can then be applied to any point, and thus eliviates many redundant matrix multiplications. Since the translation transformation cannot be carried out by matrix multiplication using Euclidean coordinate space, we cannot apply the same optimization technique, and hence homogeneous coordinate space is preferred.

4.2 Exact procrustes mapping

No, there is no such mapping because we're are limiting to the basic transformations; translation, scaling and rotation.

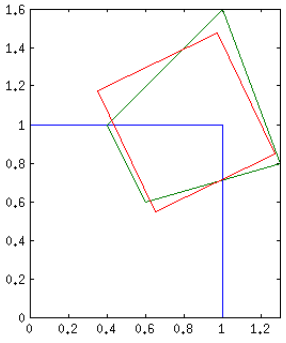

Figure 5: Attempted procrustes mapping from X to Y

Review the code to produce the figure above in appendix A.6.

4.3 Exact affine mapping

Affine gives us only one more transformation to work with; namely shearing. This, however, is still not enough to make an exact mapping.

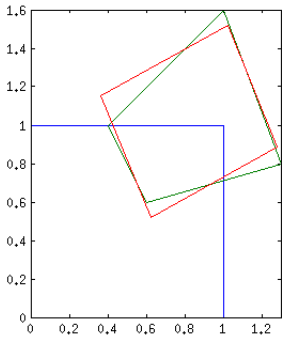

Figure 6: Attempted procrustes mapping from X to Y

Review the code to produce the figure above in appendix A.7.

4.4 Projective transform

As far as I can see, we have all the parameters available that we need to make an exact mapping with projective, although the code I've written does not seem to work, and I have failed to find any bug in it (it can be reviewed in appendix A.8).

The above figure to the right shows the output of the code. It looks like it produces a very squeezed rectangle.

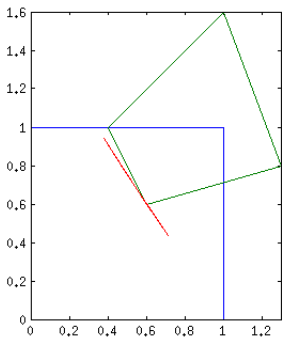

Figure 7: Faulty projective transformation mapping X to Y

A Assignments

A.1 Translation

```
6  %% assignment 1.1
7
8  clc, clear all;
9
10 H = [0 0 0;
11      1 0 0;
12      0 0 0];
13
14 I = double(imread('images/cameraman.jpg'));
15 [M,N] = size(I);
16 R = imfilter(I, H);
17
18 colormap(gray);
19 subplot(1,2,1), imagesc(I), axis image, axis off, title('original');
20 subplot(1,2,2), imagesc(R), axis image, axis off, title('filtered');
```

Figure 8: Solution code for 1.1 (../assignment1.m)

A.2 General Translation

```
22 %% assignment 1.2
23
24 clc, clear all;
25
26 N = 25;
27 C = ceil(N/2);
28 I = zeros([N N]);
29 I(C, C) = 1;
30 R = translate(I, [8 5]);
31
32 colormap(gray);
33 subplot(1,2,1), imagesc(I), axis image, axis off, title('original');
34 subplot(1,2,2), imagesc(R), axis image, axis off, title('translated');
```

Figure 9: Solution code for 1.2 (../assignment1.m)

A.3 Fourier transform translation

```
36 %% assignment 1.3
37
38 clc, clear all;
39
40 N = 7;
41 C = ceil(N/2);
42 I = zeros([N N]);
43 I(C, C) = 1;
44 [M,N] = size(I);
45
46 H = fftshift(fft2(I));
47 x = 2; y = 1;
48
49 % define shift in frequency domain
50 [u, v] = meshgrid((-M/2):(M/2-1),(-N/2):(N/2-1));
51
52 % perform the shift
53 H = H .* exp(-i*2*pi.*(u*x+v*y)/M);
54 IF = ifft2(ifftshift(H));
55
56 colormap(gray);
57 subplot(1,2,1), imagesc(I), axis image, axis off, title('original');
58 subplot(1,2,2), imagesc(abs(IF)), axis image, axis off, title('translated');
```

Figure 10: Solution code for 1.3 (../assignment1.m)

A.4 Non-integer fourier translation

```
60 %% assignment 1.4
61
62 clc, clear all;
63
64 N = 7;
65 C = ceil(N/2);
66 I1 = zeros([N N]);
67 I1(C, C) = 1;
68 I2 = peaks(25);
69 [M1,N1] = size(I1);
70 [M2,N2] = size(I2);
71
72 x = 1.5; y = 0.5;
73
74 % fourier transforms
75 H1 = fftshift(fft2(I1));
76 H2 = fftshift(fft2(I2));
77
78 % define shift in frequency domains
79 [u1, v1] = meshgrid((-M1/2):(M1/2-1),(-N1/2):(N1/2-1));
80 [u2, v2] = meshgrid((-M2/2):(M2/2-1),(-N2/2):(N2/2-1));
81
82 % perform the shift
83 H1 = H1 .* exp(-i*2*pi.*(u1*x+v1*y)/M1);
84 H2 = H2 .* exp(-i*2*pi.*(u2*x+v2*y)/M2);
85 IF1 = ifft2(ifftshift(H1));
86 IF2 = ifft2(ifftshift(H2));
87
88 colormap(gray);
89 subplot(2,2,1), imagesc(I1), axis image, axis off, title('original square');
90 subplot(2,2,2), imagesc(abs(IF1)), axis image, axis off, title('translated
       square');
91 subplot(2,2,3), imagesc(I2), axis image, axis off, title('original image');
92 subplot(2,2,4), imagesc(abs(IF2)), axis image, axis off, title('translated
       image');
```

Figure 11: Solution code for 1.4 (../assignment1.m)

A.5 Finding procrustes parameters

```
6  %% assignment 2.2
7
8  clc, clear all;
9
10 A = imread('images/imtransform3.gif');
11 B = imread('images/imtransform4.gif');
12
13 %figure (1), imshow(A), colormap(gray);
14 %[x,y] = ginput(4); input = [x,y];
15 input = [153 212; 187 167; 211 186; 176 230];
16
17 %figure (2), imshow(B), colormap(gray);
18 %[x,y] = ginput(4); base = [x,y];
19 base = [181 223; 221 188; 237 207; 198 241];
20
21 % close(1); close(2);
22
23 figure (3), colormap(gray);
24 t = cp2tform(input, base, 'affine');
25 [D,Z,T] = procrustes(input, base);
26 C = imtransform(A, t);
27 T.c, T.b, T.T % print procrustes information
28
29 X = [1 0 T.c(1); 0 1 T.c(2); 0 0 1];
30 R = [T.T(1,1) T.T(1,2) 0; T.T(2,1) T.T(2,2) 0; 0 0 1];
31 S = [T.b 0 0; 0 T.b 0; 0 0 1];
```

Figure 12: Solution code for 2.2 (../assignment2.m)

A.6 Exact procrustes mapping

```
6  %% assignment 4.2
7
8  clc, clear all;
9
10 X = [0.0 0.0; 1.0 0.0; 1.0 1.0; 0.0 1.0; 0.0 0.0];
11 Y = [0.6 0.6; 1.3 0.8; 1.0 1.6; 0.4 1.0; 0.6 0.6];
12
13 [D, Z, transform] = procrustes(Y, X);
14
15 figure (4-2);
16 plot(X(:,1),X(:,2),Y(:,1),Y(:,2),Z(:,1),Z(:,2)), axis image;
```

Figure 13: Solution code for 4.2 (../assignment4.m)

A.7 Exact affine mapping

```
%% assignment 4.3

clc, clear all;

X = [0.0 0.0; 1.0 0.0; 1.0 1.0; 0.0 1.0; 0.0 0.0];
Y = [0.6 0.6; 1.3 0.8; 1.0 1.6; 0.4 1.0; 0.6 0.6];

[M,N] = size(X);
t = fitgeotrans(X, Y, 'affine');

X = X';
Z = zeros(N,M);
for i = 1:M
    s = X(:,i);
    s = [s(1) s(2) 1]';
    u = (t.T') * s;
    Z(:,i) = [u(1) u(2)]';
end

X=X', Y, Z=Z' % print for comparison
plot(X(:,1),X(:,2),Y(:,1),Y(:,2),Z(:,1),Z(:,2)), axis image;
```

Figure 14: Solution code for 4.3 (../assignment4.m)

A.8 Projective transform

```
%% assignment 4.4

clc, clear all;

X = [0.0 0.0; 1.0 0.0; 1.0 1.0; 0.0 1.0; 0.0 0.0];
Y = [0.6 0.6; 1.3 0.8; 1.0 1.6; 0.4 1.0; 0.6 0.6];

[M,N] = size(X);
t = fitgeotrans(X, Y, 'projective');

X = X';
Z = zeros(N,M);
for i = 1:M
    s = X(:,i);
    s = [s(1) s(2) 1]';
    u = (t.T') * s;
    Z(:,i) = [u(1) u(2)]';
end

X=X', Y, Z=Z' % print for comparison
plot(X(:,1),X(:,2),Y(:,1),Y(:,2),Z(:,1),Z(:,2)), axis image;
```

Figure 15: Solution code for 4.4 (../assignment4.m)

B Functions

B.1 Translate Function

```
function [ out ] = translate( I, t )
    [M,N] = size(I);
    Tt = [1 0 t(1);
          0 1 t(2);
          0 0 1  ];

    for x = 1:M
        for y = 1:N
            v = Tt * [x y 1]';
            out(mod(v(1),M)+1, mod(v(2),N)+1) = I(x,y);
        end
    end
end
```

Figure 16: Generalized translation function (../translate.m)

References

[1] Solomon & Breckon, Fundamentals of Digital Image Processing - A Practical Approach with Examples in Matlab, 2011