# Image processing for Bioinformatics

## Laboratory
## Geometric Transformations

# 1 Affine transformations

## 1.1 Example with Matlab functions

Code

```
%% affine2d and imwarp
clc; clear; close all;
I = imread('cameraman.tif');
[m,n] = size(I);

% Reference 2-D image to world coordinates
RA = imref2d([m,n]);

% 2-D geometric transformation object.
% Traslation *** transpose matrix ***
t = [1 0 50; 0 1 100; 0 0 1];
tform = affine2d(t.');

% Apply the transformation to the image.
% RB  Spatial referencing information associated with the transformed image
[J,RB] = imwarp(I,RA,tform, 'nearest', 'FillValues',0);
% 'OutputView'  Size and location of output image in world coordinate system
[J2,RB2] = imwarp(I,RA,tform, 'nearest', 'FillValues',0,'OutputView',RA);
RB
RB2
```

Command window

```
RB =
imref2d with properties:

XWorldLimits: [50.5000 306.5000]
YWorldLimits: [100.5000 356.5000]
ImageSize: [256 256]
PixelExtentInWorldX: 1
PixelExtentInWorldY: 1
ImageExtentInWorldX: 256
ImageExtentInWorldY: 256
XIntrinsicLimits: [0.5000 256.5000]
YIntrinsicLimits: [0.5000 256.5000]

RB2 =
imref2d with properties:

XWorldLimits: [0.5000 256.5000]
YWorldLimits: [0.5000 256.5000]
ImageSize: [256 256]
PixelExtentInWorldX: 1
PixelExtentInWorldY: 1
ImageExtentInWorldX: 256
ImageExtentInWorldY: 256
XIntrinsicLimits: [0.5000 256.5000]
YIntrinsicLimits: [0.5000 256.5000]
```

Image

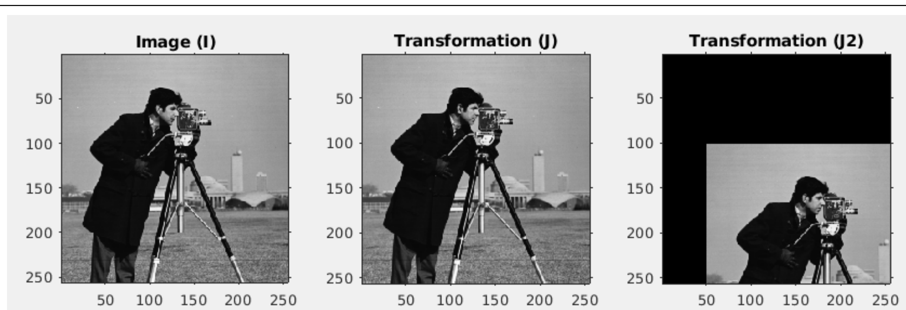



Table 1: Affine transformations

## 1.2 Assignment

- Change the *transformation object* and test others affine transformations.
- Combine and concatenate different sequences of transformations, are the output images the "same" if the order of transformations is different?
  To concatenate transformations, use the *output argument* **RB** (Spatial referencing information) of one transformation as *input argument* **RA** of the next transformation. Or combine the transformations multiplying them.
- Rotate an image around its center.

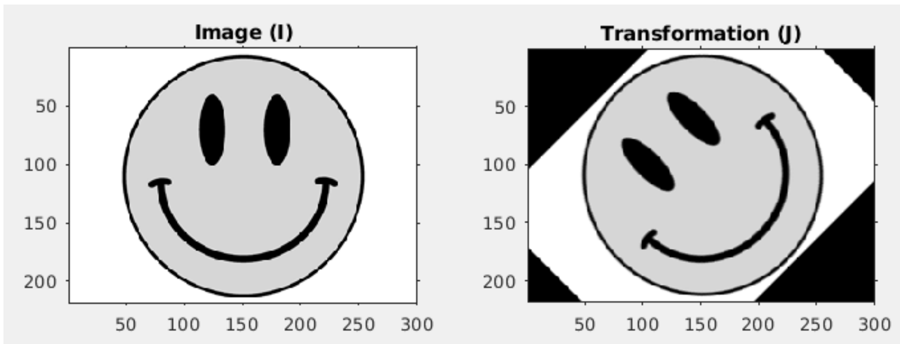
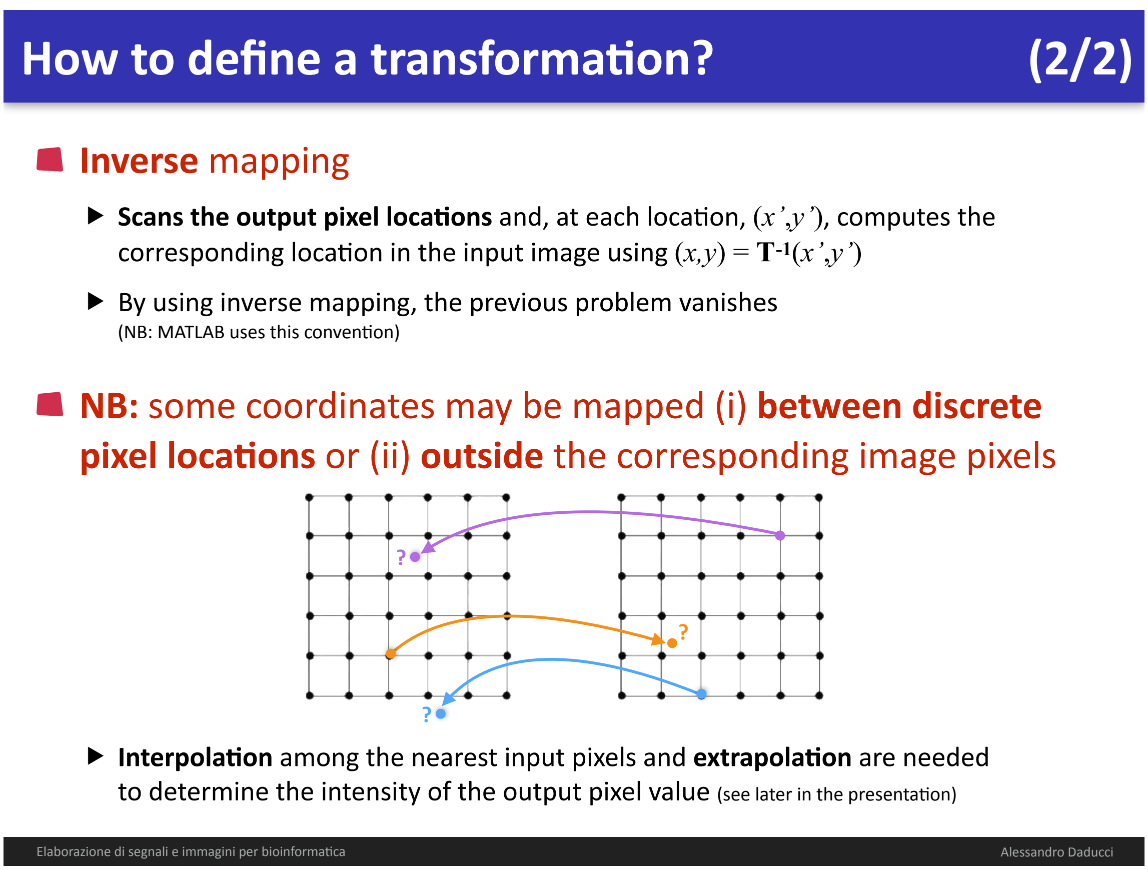



Figure 1: Rotate an image around its center

- Implement the function *imwarp* with the inverse mapping algorithm to perform a transformation:




To interpolate the values you can use the nearest neighbor.

- Find the sequence of transformations that generates the following output image $I2$.

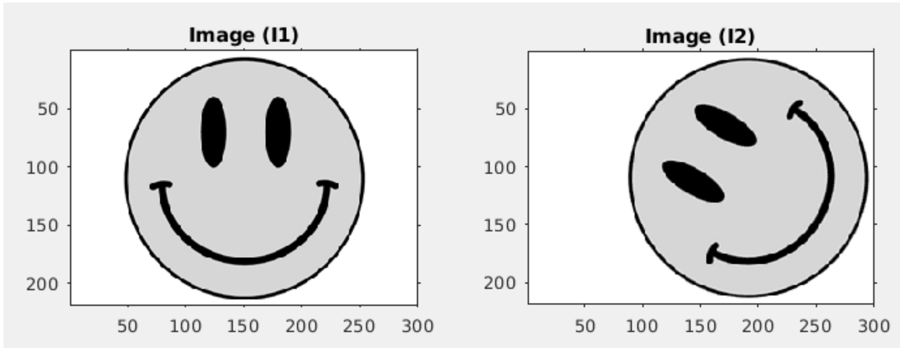



Figure 2: Find the affine transformation $T$ (1)

- Find the affine transformation $T$ using the corresponding points or two images.

---

**Algorithm 1** Find the affine transformation T

---

With Matlab function [x,y] = *ginput*(n) select at least $n = 3$ pairs of corresponding points in I1 and I2.
Create a linear equation system of the form $Ma = b$
Compute the pseudo-inverse of $M$
Compute the solution of the system $a = (M^T M)^{-1} M^T b$
With $a$ compute $T$
Apply $T$ to I1 to obtain J and compare it to I2

---

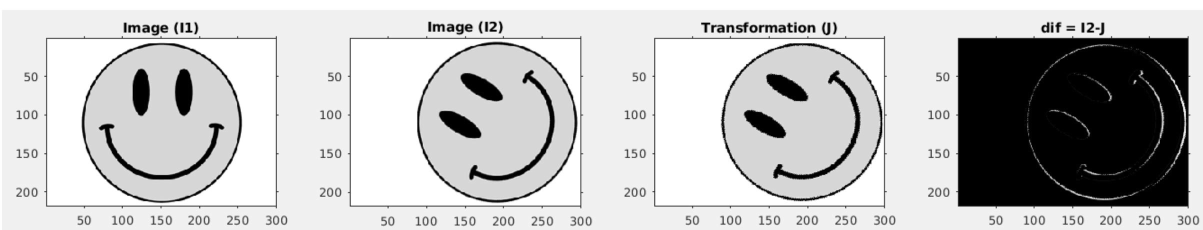



Figure 3: Find the affine transformation $T$ (2)

## 1.3 Solutions

- Change the *transformation object* and test others affine transformations.
- Combine and concatenate different sequences of transformations, are the output images the "same" if the order of transformations is different?
  To concatenate transformations, use the *output argument* **RB** (Spatial referencing information) of one transformation as *input argument* **RA** of the next transformation. Or combine the transformations multiplying them.
- Rotate an image around its center.

affine2d and imwarp Concatenation

```
%% affine2d and imwarp Concatenation
clc; clear; close all;
I = imread('faceBW.png');
[m,n] = size(I);

% Reference 2-D image to world coordinates
RA = imref2d([m,n]);

% Create a 2-D geometric transformation object.
% Traslation
t = [1 0 -n/2; 0 1 -m/2; 0 0 1];
tform = affine2d(t.');
[J,RB] = imwarp(I,RA,tform);
% Rotation
theta = -45;
t = [cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1];
tform = affine2d(t.');
[J2,RB2] = imwarp(J,RB,tform);
% Traslation
t = [1 0 n/2; 0 1 m/2; 0 0 1];
tform = affine2d(t.');
[J3,RB3] = imwarp(J2,RB2,tform,'OutputView',RA);

figure;
nr = 1;
nc = 2;
subplot(nr, nc, 1); imshow(I); title('Image (I)'); axis on;
subplot(nr, nc, 2); imshow(J3); title('Transformation (J3)'); axis on;
```

affine2d and imwarp Concatenation (multiplication)

```
%% affine2d and imwarp Concatenation multiplication
clc; clear; close all;
I = imread('faceBW.png');
[m,n] = size(I);

% Reference 2-D image to world coordinates
RA = imref2d([m,n]);

% Create a 2-D geometric transformation object.
% Traslation*Rotation*Traslation
theta = -45;
t = [1 0 n/2; 0 1 m/2; 0 0 1]*[cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1]*[1 0 -n/2;
    0 1 -m/2; 0 0 1];
tform = affine2d(t.');
[J,RB] = imwarp(I,RA,tform,'OutputView',RA);


figure;
nr = 1;
nc = 2;
subplot(nr, nc, 1); imshow(I); title('Image (I)'); axis on;
subplot(nr, nc, 2); imshow(J); title('Transformation (J)'); axis on;
```

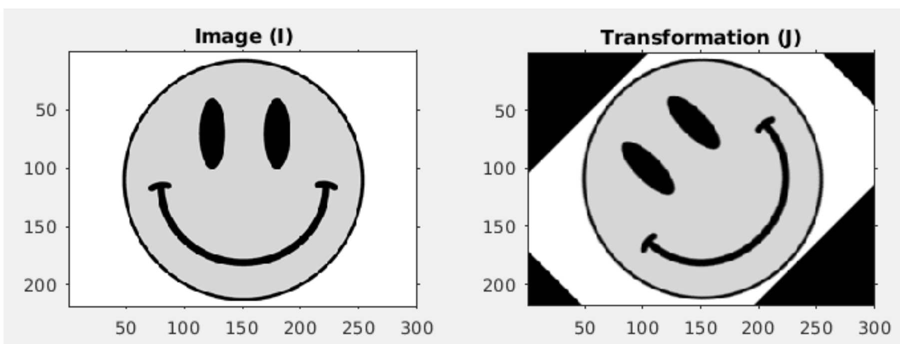


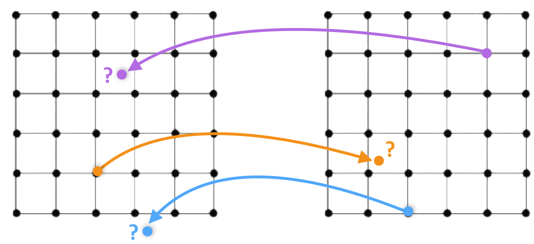

Figure 4: Rotate an image around its center

- Implement the function *imwarp* with the inverse mapping algorithm to perform a transformation:

# How to define a transformation? (2/2)

- **Inverse** mapping
  - **Scans the output pixel locations** and, at each location, $(x',y')$, computes the corresponding location in the input image using $(x,y) = \mathbf{T}^{-1}(x',y')$
  - By using inverse mapping, the previous problem vanishes
    (NB: MATLAB uses this convention)

- **NB:** some coordinates may be mapped (i) **between discrete pixel locations** or (ii) **outside** the corresponding image pixels




  - **Interpolation** among the nearest input pixels and **extrapolation** are needed to determine the intensity of the output pixel value (see later in the presentation)



To interpolate the values you can use the nearest neighbor.

Function myImwarp

```
% The X, Y axes of the image must correspond to those of the world,
% the first component of the vector of coordinates must be columns (j) and the second rows (i).
function J=myImwarp(I, t)

  [m,n] = size(I);

  J = zeros([m,n]);
  for i = 1:m % Rows: Y axis
    for j = 1:n % Cols: X axis
      aux=fix(t\[j;i;1]); % vector of coordinates [X;Y;1]
      u=aux(2); % Y component
      v=aux(1); % X component
      if 0<u && u<=m && 0<v && v<=n
        J(i,j) = I(u,v);
      end
    end
  end
end
```

- Find the sequence of transformations that generates the following output image $I2$.

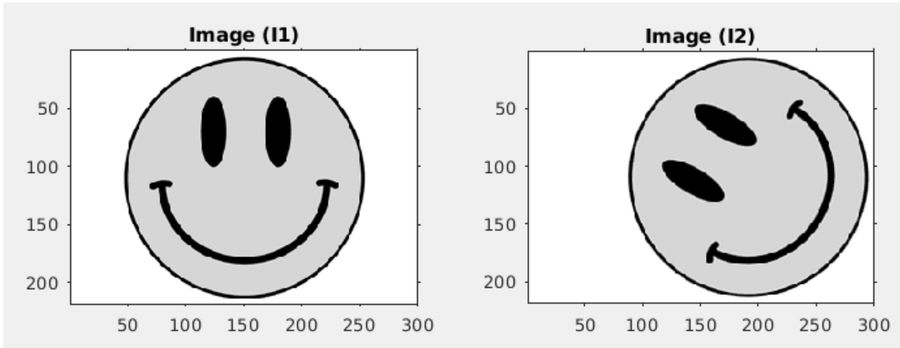



Figure 5: Find the affine transformation $T$ (1)

Function myImwarp - Example

```
%% myImwarp - test
clc; clear; close all;
I = im2double(imread('faceBW.png'));
[m,n] = size(I);

% Reference 2-D image to world coordinates
RA = imref2d([m,n]);

% Rotate an image around its center
% Create a 2-D geometric transformation object.
theta = -45;

% Sequence of transformations that generates I2
t = [1 0 39; 0 1 0; 0 0 1]*[1 0 n/2; 0 1 m/2; 0 0 1]*[cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1]*[1 0 -n/2; 0 1 -m/2; 0 0 1];

%myImwarp
J=myImwarp(I, t);

% Matlab transformation *** transpose matrix ***
tform = affine2d(t.');
% Apply the transformation to the image.
[J2,RB2] = imwarp(I,RA,tform, 'nearest', 'FillValues',0,'OutputView',RA);

dif = J-J2;
sum(sum(dif))

% figure;
nr = 1; nc = 4;
subplot(nr, nc, 1); imshow(I); title('Image I1'); axis on;
subplot(nr, nc, 2); imshow(J); title({'Image J','(myImwarp)'}); axis on;
subplot(nr, nc, 3); imshow(J2); title({'Image J','(Matlab imwarp)'}); axis on;
subplot(nr, nc, 4); imshow(J-J2); title({'Image J-J2','(Matlab imwarp)'}); axis on;
```

- Find the affine transformation $T$ using the corresponding points or two images.

**Algorithm 2** Find the affine transformation T

With Matlab function [x,y] = $ginput$(n) select at least $n = 3$ pairs of corresponding points in I1 and I2.
Create a linear equation system of the form $Ma = b$
Compute the pseudo-inverse of $M$
Compute the solution of the system $a = (M^T M)^{-1} M^T b$
With $a$ compute $T$
Apply $T$ to I1 to obtain J and compare it to I2

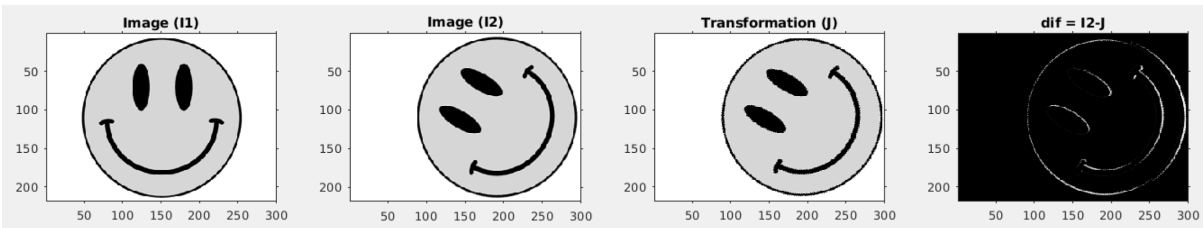



Figure 6: Find the affine transformation $T$ (2)

Find T - Example 1 - two sets of linear equations

```
%% Find T − two sets of linear equations
clc; clear; close all;
I1 = im2double( imread('faceBW.png') );
I2 = im2double( imread('faceRTBW.png') );
[m,n] = size(I1);
np = 4;
% Select np points
imshow(I1);
[x1,y1] = ginput(np);
imshow(I2);
[x2,y2] = ginput(np);

vecOnes = ones(np,1);
M2 = [x1,y1,vecOnes];
bx=x2;
by=y2;
ax = inv(M2.'*M2)*M2.'*bx;
ay = inv(M2.'*M2)*M2.'*by;

T = [ax.';ay.';[0,0,1]];

% Reference 2−D image to world coordinates
RA = imref2d([m,n]);
% Create a 2−D geometric transformation object.
tform = affine2d(T.');
% Apply the transformation to the image.
[J,RB] = imwarp(I1,RA,tform, 'nearest', 'FillValues',1,'OutputView',RA);

dif = I2−J;

figure; nr = 1; nc = 4;
subplot(nr, nc, 1); imshow(I1); title('Image (I1)'); axis on;
subplot(nr, nc, 2); imshow(I2); title('Image (I2)'); axis on;
subplot(nr, nc, 3); imshow(J); title('Transformation (J)'); axis on;
subplot(nr, nc, 4); imshow(dif); title('dif = I2−J'); axis on;
```

Find T Example 2 - x' and y' in the same matrix

```matlab
%% Find T - x' and y' in the same matrix
clc; clear; close all;
I1 = im2double( imread('faceBW.png') );
I2 = im2double( imread('faceRTBW.png') );
[m,n] = size(I1);
np = 4;

% Select np points
imshow(I1);
[x1,y1] = ginput(np);

imshow(I2);
[x2,y2] = ginput(np);

vecOnes = ones(np,1);
vecZeros = zeros(np,1);
M = [[x1;vecZeros],[y1;vecZeros],[vecOnes;vecZeros],[vecZeros;x1],[vecZeros;y1],[vecZeros;vecOnes]];
b = [x2;y2];
a = inv(M.'*M)*M.'*b;

T = [a(1:3).';a(4:6).';[0,0,1]];

% Reference 2-D image to world coordinates
RA = imref2d([m,n]);

% Create a 2-D geometric transformation object.
tform = affine2d(T.');

% Apply the transformation to the image.
[J,RB] = imwarp(I1,RA,tform, 'nearest', 'FillValues',1,'OutputView',RA);

dif = I2-J;

figure;
nr = 1;
nc = 4;
subplot(nr, nc, 1); imshow(I1); title('Image (I1)'); axis on;
subplot(nr, nc, 2); imshow(I2); title('Image (I2)'); axis on;
subplot(nr, nc, 3); imshow(J); title('Transformation (J)'); axis on;
subplot(nr, nc, 4); imshow(dif); title('dif = I2-J'); axis on;
```