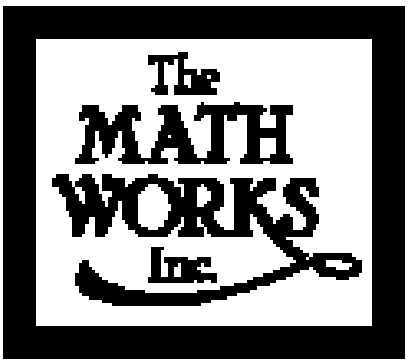
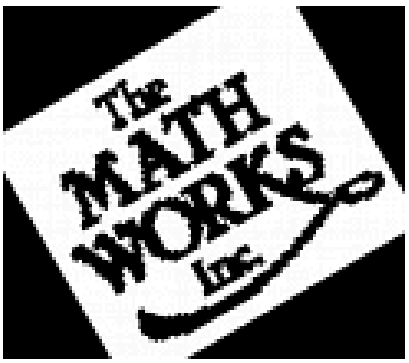
LAB 2:
Resampling – Image rotation

Computer Vision Laboratory
Linköping University, Sweden

Written January 2002 by
Qingfen Lin and Katarina Flood
Updated 2006 - 2011 by
Maria Magnusson, Erik Jonsson,
Johan Sunnegårdh and Fredrik Larsson
Updated 2017 - 2020 by Maria Magnusson





1 Preparations

Before coming to the computer session it is necessary to read through the course material on image resampling and this lab instruction.

Here you will find home exercises (marked with a pointing hand) to be answered before the session.

If you are unfamiliar with any of the MATLAB commands that are used in the lab, please use the

```
help
```

function for more information. Some MATLAB commands that might be useful in the lab are described below.

The command for displaying images, `imagesc`, can take an argument indicating the range of pixel values. For example,

```
imagesc(Im, [min max]);
```

displays the image with a linear scale between the values `min` and `max`. This command without scaling, i.e.

```
imagesc(Im);
```

automatically finds the minimum and maximum of the image and displays it with a linear scale in between. The command

```
colormap gray;
```

displays the image with grayscale values and the command

```
colormap jet;
```

displays the image with color values. The commands

```
axis image;
axis off;
title('rotated image');
colorbar;
```

are useful in connection to displaying an image. Try them and use the `help` function on them if needed.

2 Image shearing. A resampling example.

Put the files

```
rotateimage.m
shearimage.m
shearimagefast.m
bicubic16.txt
baboon.tif
logo.tif
```

to a suitable place at your own directory.
Create a file `Example1.m` with the following content:

```
Im = double(imread('logo.tif'));  % load image
T = [1 -1/3; 0 1];                % assign shear matrix
```

Start MATLAB and run `Example1.m`. This shear transformation of an image is defined by the affine transformation

$$\begin{bmatrix} x_g \\ y_g \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{3} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_f \\ y_f \end{bmatrix} \tag{1}$$

where (x_f, y_f) and (x_g, y_g) are the coordinates of the input and output images respectively. In your directory you will find the function `shearimage`, which takes the two parameters `Im` and `T` and returns `shearIm`.

```
1  function shearIm = shearimage(Im, T)
2
3  [rows, cols] = size(Im);
4  shearIm = zeros(rows, cols);
5  for xg = 1:cols
6    for yg = 1:rows
7      xyff = inv(T)*[xg;yg];
8      xff  = xyff(1);
9      yff  = xyff(2);
10     if (xff<=cols & yff<=rows & xff>=1 & yff>=1)
11       xf = round(xff);
12       yf = round(yff);
13       shearIm(yg,xg) = Im(yf,xf);
14     end
15   end
16 end
```

The program loops through the coordinates (x_g, y_g) of this output image of size $(cols, rows)$. It calculates the corresponding position in the input image on line 7,8,9 and checks whether it is inside the image range on line 10. If yes, nearest neighbour interpolation is carried out, otherwise the position is left as zero.
QUESTION 1: On line 7, we use the inverse of the matrix `T`, why?

To execute `shearimage` and plot the image before and after shearing side by side, add the following content to `Example1.m`:

```
figure(1); colormap gray;
shearIm = shearimage(Im,T);
subplot(121); imagesc(Im); axis image; colorbar;
subplot(122); imagesc(shearIm); axis image; colorbar;
```

QUESTION 2: Change the value $-\frac{1}{3}$ in T to a different value, i.e. -1. Around which point is the image sheared?

QUESTION 3: How can you change line 7 in the code so that the image is sheared around the center of the image `(cx,cy)=(cols/2,rows/2)` instead?

Comments There are a number of points which you should be aware of in this example, namely,

- In general, affine transformations changes the size of the image, as the extensions in x and y are affected. For simplicity, the image size in our example is kept constant after shearing although certain parts of the image are cropped (line 8).
- In MATLAB , the matrix coordinate system origin is located at the upper-left corner, the i-axis is vertical and is numbered from top to bottom, the j-axis is horizontal and is numbered from left to right. In this example we use (i, j) as (y, x). Since y is numbered from top to bottom, we do not have a Cartesian grid in its right sense, but rather a vertically flipped version of it.

MATLAB is very slow at running through loops. It is, however, very efficient at performing matrix calculations. There are many specialized commands for this purpose which might take a while to get used to. We here include alternative code using so called *vectorization* instead of loops for the function `shearimage`. The reason is that we want to show how to write efficient MATLAB code.

```
1  function shearIm = shearimageFast(Im, T)
2
3  T=inv(T);
4  [rows, cols] = size(Im);
5  shearIm = zeros(rows, cols);
6  [xgGrid, ygGrid] = meshgrid(1:cols, 1:rows);
7  xfGrid = round(xgGrid * T(1,1) + ygGrid * T(1,2));
```

```
8   yfGrid = round(xgGrid * T(2,1) + ygGrid * T(2,2));
9   ind1 = find(xfGrid>0 & xfGrid<=cols & yfGrid>0 & yfGrid<=rows);
10  ind2 = sub2ind([rows, cols],  yfGrid(ind1), xfGrid(ind1));
11  shearIm(ind1)=Im(ind2);
```

QUESTION 4: Compare the computational speed of the two codes for shearing, i.e. perform `shearimageFast(im, T)` and `shearimage(im, T)`. Did you notice any time difference between the two? Tip: Use the commands `tic` and `toc`.

From now on, we do not recommend you to use vectorization code in the lab since it is much more difficult to program.

3 Image rotation

Your task is to write a MATLAB function that rotates an image around its center using three different interpolation methods. (We recommend you again to use loops as in `shearimage` if you are not very familiar with vectorization commands.) The main structure of your MATLAB function `RotateIm.m` may look like:

```
function RotateIm = rotateimage(Im, theta, intpol)
...
switch intpol
  case 'nearest'
   % rotation code with nearest neighbour interpolation
  case 'bilinear'
   % rotation code with bilinear interpolation
  case 'bicubic4'
   % rotation code with bicubic4 interpolation
  case 'bicubic16'
   % rotation code with bicubic16 interpolation
  otherwise
   error('Unknown interpolation method');
end
```

QUESTION 5: Write an equation, similar to (1), for the affine transformation equation for rotation an angle θ around the point (x_t, y_t).

3.1 Nearest neighbour interpolation

a. Implementation exercise

The formula for nearest neighbour interpolation can be given as a kind of convolution, namely

$$g(x_g, y_g) = \hat{f}(x_f, y_f) = \sum_{\alpha,\beta} f(\alpha, \beta) \cdot \mathbf{\Pi}(x_f - \alpha, y_f - \beta) = \sum_{\alpha,\beta} f(\alpha, \beta) \cdot \Pi(x_f - \alpha) \cdot \Pi(y_f - \beta) \quad (2)$$

where we have assumed that α and β are integers and image sample points and that the sample distance $\Delta = 1$. The output image sample points (x_g, y_g) (integers here) are transformed to input coordinates (x_f, y_f) which are normally not integers.

QUESTION 6: Write an alternative equation to (2) including the MATLAB command `round` instead of the sum and the rectangular function $\mathbf{\Pi}$.

Now implement a complete MATLAB code for rotation using nearest neighbour interpolation. Use a MATLAB script `Example2.m` (similar to `Example1.m`) that calls the function `RotateIm.m`. Try to rotate the image with some different angles, i.e. $\pi/6$ and $\pi/3$.

b. Image quality check

To check how the interpolation in the rotation affects the image, first rotate an image with θ, then rotate it back with $-\theta$. Extend `Example2.m` with:

```
rotIm = rotateimage(Im, pi/6, 'nearest');
nIm = rotateimage(rotIm, -pi/6, 'nearest');
```

The image obtained after the second rotation will have the same orientation as the original one, but with missing corners as shown in Figure 1. In order to compare the first and the last images in the previous figure, it is a good idea to aviod the problem with the missing corners by applying a circular mask to the original image before any rotation. Extend `Example2.m` with:

```
[Ny, Nx] = size(Im);
N = min(min(Nx,Ny));
[x,y] = meshgrid(-ceil((Nx-1)/2):floor((Nx-1)/2), ...
 -ceil((Ny-1)/2):floor((Ny-1)/2));
mask = (x.^2 + y.^2)<((N-1)/2)^2;
Im = Im.* mask;
```

The result should look like as shown in Figure 2.

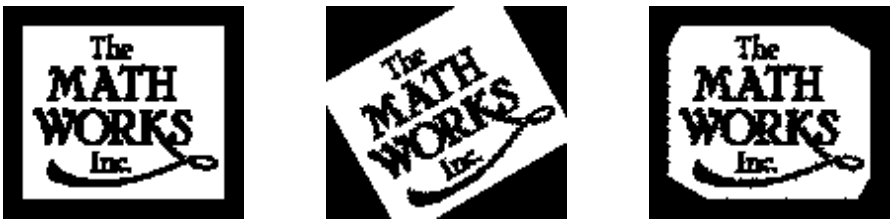


Figure 1: The original (left), rotated (middle), and back-rotated image (right).

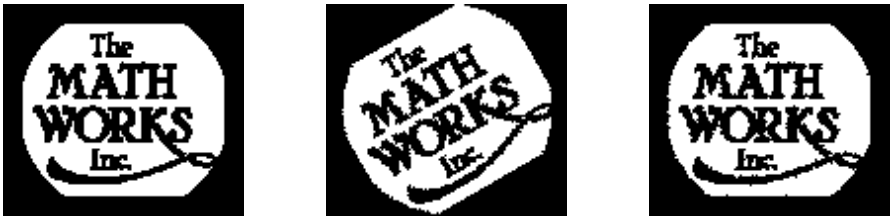


Figure 2: The original (left), rotated (middle), and back-rotated image (right).

Now apply the mask and rotate the image forth and back. Display the images and check the result. Also, look at the difference-image by using, for example, the command

```
imagesc(nIm-Im); axis image; axis off; colorbar;
```

QUESTION 7: How does the difference-image look like? How large and where are the errors?

A more objective measure is to calculate the error energy, i.e. the sum of the squared difference-image:

```
sum(sum((nIm-Im).*(nIm-Im)))
```

QUESTION 8: What is the error energy in the spatial domain when using nearest neighbour interpolation?

Note: You should get the answer 134 here. In case you have not, ask the teacher to help you with the code. Small variations in rotation center and rotation angle may change the value. If all students have the same value here, it will be easier to check the following exercises.

c. Quality check in the Fourier domain

Calculate the Fourier transform of `Im` and `nIm` by extending `Example2.m` with:

```
fIm = fftshift(fft2(ifftshift(Im)));
fnIm = fftshift(fft2(ifftshift(nIm)));
```

Comment If you display the Fourier amplitude spectrum $|F(u,v)|$ directly, you will find that the middle part (including the DC-component) is shown as a very bright spot, while the rest is almost zero. To overcome this problem, display $^{10}\log(1+|F(u,v)|)$ instead. The logarithm function performs the desired compression of the amplitudes. Note that we add 1 in the expression $^{10}\log(1+|F(u,v)|)$ to avoid $\log(0) = -\infty$.

Display the Fourier transform of the original (`Im`), the forward-rotated (`rotIm`), and the forward and back-rotated image (`nIm`). Use the `abs` and `log` commands to obtain $^{10}\log(1+|F(u,v)|)$ as descibed above.

QUESTION 9: Do the Fouriertransform of the original and the forward and back-rotated image look similar?

QUESTION 10: Also, look at the Fourier transform of the difference image (`nIm-Im`). How are the errors distributed in relation to the frequences?

Compute the error energy in the Fourier domain, i.e.

```
N = size(fIm)
sum(sum((fIm-fnIm).*conj(fIm-fnIm)))/(N(1)*N(2))
```

QUESTION 11: What is the error energy in the Fourier domain when using nearest neighbour interpolation?

QUESTION 12: Compare the error energy in the spatial and the Fourier domain. Which theorem relates these two measures and do your own measurements agree with the theorem?

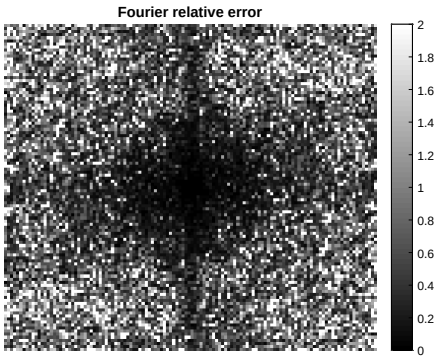
QUESTION 13: Do your own measurements agree with the theorem?

Previously, we used the logarithm of the absolute error, $^{10}\log(1+|F(u,v)-G(u,v)|)$ to compare two Fourier transforms. Now, define a *relative* error for the Fourier components instead, i.e. $|F(u,v)-G(u,v)|/|F(u,v)|$ (log is no longer needed).

QUESTION 14: Why can this be a better measure? Remember that in most images, the low frequencies have ...?... amplitude and the high frequencies have ...?... amplitude.

QUESTION 15: Compute and display the relative error image for the Fourier components. How are the errors distributed in relation to the frequences? A contrast interval between 0 and 2 is recommended here, i.e. use `[0 2]` as a parameter to `imagesc`.

The image should look like as shown below.



3.2 Bilinear Interpolation

a. Implementation exercise

The formula for bilinear interpolation can be given as a kind of convolution, namely

$$g(x_g, y_g) = \hat{f}(x_f, y_f) = \sum_{\alpha,\beta} f(\alpha,\beta) \cdot \mathbf{\Lambda}(x_f - \alpha, y_f - \beta) = \sum_{\alpha,\beta} f(\alpha,\beta) \cdot \Lambda(x_f - \alpha) \cdot \Lambda(y_f - \beta) \quad (3)$$

where we have assumed that α and β are integers and image sample points and that the sample distance $\Delta = 1$. The output image sample points

(x_g, y_g) (integers here) are transformed to input coordinates (x_f, y_f) which are normally not integers.

QUESTION 16: The procedure for calculating bilinear interpolation was given in Lecture 4. Which slide(s) are you going to use to support your implemention of bilinear interpolation?

Now add the code for bilinear interpolation to `rotateimage.m` and execute it on the same image as before. Plot the result.

b. Image quality check

QUESTION 17: To check how the interpolation in the rotation affects the image, rotate and back-rotate the image as you did in section 3.1b. How does the difference-image look like? How large and where are the errors?

QUESTION 18: What is the error energy in the spatial domain when using bilinear interpolation? Compare it with the result for nearest neighbour interpolation.

---

c. Quality check in Fourier domain

Display the Fourier transform of the original and the forward and back-rotated image. Use the `abs` and `log` commands to obtain $^{10}\log(1+|F(u,v)|)$ as descibed above.

QUESTION 19: Why are the high frequencies more attenuated in the forward and back-rotated image?

QUESTION 20: Compute and display the relative error image for the Fourier components. Compare it with the previous one when you used nearest neighbour interpolation. How are the errors distributed in relation to the frequences? A contrast interval between 0 and 2 is recommended here, i.e. `[0 2]`.

3.3 Bicubic4 interpolation

a. Implementation exercise

The formula for bicubic4 interpolation can be given as a kind of convolution, namely

$$g(x_g, y_g) = \hat{f}(x_f, y_f) = \sum_{\alpha,\beta} f(\alpha, \beta) \cdot \mathbf{h}(x_f - \alpha, y_f - \beta) = \sum_{\alpha,\beta} f(\alpha, \beta) \cdot h(x_f - \alpha) \cdot h(y_f - \beta) \tag{4}$$

where we have assumed that α and β are integers and image sample points and that the sample distance $\Delta = 1$. The output image sample points (x_g, y_g) (integers here) are transformed to input coordinates (x_f, y_f) which are normally not integers. The equation for $h(x)$ is

$$h(x) = \begin{cases} 2 \cdot |x|^3 - 3 \cdot x^2 + 1, & |x| \leq 1, \\ 0, & \text{elsewhere.} \end{cases} \tag{5}$$

Now add the code for bicubic4 interpolation to `rotateimage.m` and execute it on the same image as before. Plot the result.

Hint: Compared to bilinear interpolation, the triangular functions $\Lambda(x)$ and $\Lambda(y)$, should be replaced with $h(x)$ and $h(y)$.

b. Image quality check

To check how the interpolation in the rotation affects the image, rotate and back-rotate the image as you did in section 3.1b.

QUESTION 21: What is the error energy in the spatial domain when using bicubic4 interpolation? Compare it with the result for bilinear and nearest neighbour interpolation.

---

c. Quality check in Fourier domain

QUESTION 22: Compute and display the relative error image for the Fourier components. Compare it with the previous one when you used nearest neighbor interpolation and bilinear interpolation. How are the errors distributed in relation to the frequences? A contrast interval between 0 and 2 is recommended here, i.e. `[0 2]`.

3.4 Bicubic16 interpolation

a. Implementation exercise

The formula for bicubic16 interpolation can also be given as in equation (4), but now the equation for $h(x)$ is

$$h(x) = \begin{cases} 1.5|x|^3 - 2.5x^2 + 1, & |x| \leq 1, \\ -0.5|x|^3 + 2.5x^2 - 4|x| + 2, & 1 \leq |x| \leq 2, \\ 0, & \text{elsewhere.} \end{cases}$$

Most of the code for bicubic16 interpolation is given in `bicubic16.txt`. Complete it and add it to `rotateimage.m` and execute it on the same image as before. Plot the result.

QUESTION 23: Why do you get values below 0 and values above 1? (To get a nicer look you can supply the range `[0 1])` to `imagesc`.)

---

b. Image quality check

To check how the interpolation in the rotation affects the image, rotate and back-rotate the image as you did in section 3.1b.

QUESTION 24: What is the error energy in the spatial domain when using bicubic16 interpolation? Compare it with the result for bicubic4, bilinear and nearest neighbour interpolation.

---

c. Quality check in Fourier domain

QUESTION 25: Compute and display the relative error image for the Fourier components. Compare it with the previous one when you used nearest neighbor interpolation and bilinear interpolation. How are the errors distributed in relation to the frequences? A contrast interval between 0 and 2 is recommended here, i.e. `[0 2]`.

3.5 Image quality for a natural image

Apply three interpolation methods on a natural image, `baboon.tif`. (You may skip bicubic4.) Compare the image quality after several forward rotations (to get a large effect). For example, you can rotate $\pi/6.1$ radians 11 times. (Obs! Do not use $\pi/6$!)

QUESTION 26: Here, it should be clearly visible that bicubic16 interpolation gives the best image quality - right? Also show the images to the teacher! Is the teacher content?

3.6 Resampling and interpolation with interp2.m in MATLAB

MATLAB contains a resampling and interpolation function `interp2.m`, which works with *inverse mapping*, see Figure 3. Specify the following:

- a matrix with the input x-coordinates, e.g. `X1`
- a matrix with the input y-coordinates, e.g. `Y1`
- the input image, e.g. `im`
- a matrix with the input x-coordinates `X1interp` expressed in the output coordinates `X2`
- a matrix with the input y-coordinates `Y1interp` expressed in the output coordinates `Y2`
- the interpolation method, e.g. `'linear'`

The following script utilizes `interp2.m` to perform rotation with nearest neighbor:

```
1  Im = double(imread('logo.tif'));
2  [Ny, Nx] = size(Im);
3  N = min(min(Nx,Ny));
```

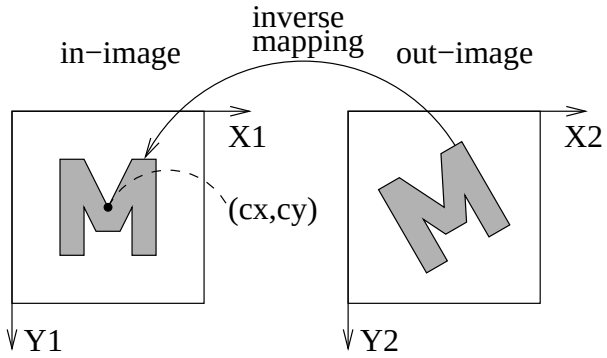


Figure 3: Illustration how interp2.m works in the case of rotation.

```
4   [x,y] = meshgrid(-ceil((Nx-1)/2):floor((Nx-1)/2), ...
5                    -ceil((Ny-1)/2):floor((Ny-1)/2));
6   mask = (x.^2 + y.^2)<((N-1)/2)^2;
7   Im = Im.* mask;
8   [rows, cols] = size(Im);
9   cx = cols/2;
10  cy = rows/2;
11  [X1, Y1] = meshgrid(1:cols, 1:rows); % inimage grid
12  [X2, Y2] = meshgrid(1:cols, 1:rows); % outimage grid
13  phi = pi/6;
14  X1interp = cos(phi)*(X2-cx) - sin(phi)*(Y2-cy) + cx;
15  Y1interp = sin(phi)*(X2-cx) + cos(phi)*(Y2-cy) + cy;
16  X1interpback = cos(-phi)*(X2-cx) - sin(-phi)*(Y2-cy) + cx;
17  Y1interpback = sin(-phi)*(X2-cx) + cos(-phi)*(Y2-cy) + cy;
18  rotIm = interp2(X1, Y1, Im, X1interp, Y1interp, 'nearest');
19  backrotIm = interp2(X1, Y1, rotIm, X1interpback, Y1interpback, 'nearest');
20  figure(1); colormap gray;
21  subplot(221); imagesc(Im,[0 1]);
22  axis image; colorbar; title('orig Im')
23  subplot(222); imagesc(rotIm,[0 1]);
24  axis image; colorbar; title('rotated Im')
25  subplot(223); imagesc(backrotIm,[0 1]);
26  axis image; colorbar; title('backrotated Im')
27  nansum(nansum((backrotIm-Im).*(backrotIm-Im)))
```

Note that `nansum`, which disregards undefined values, is used instead of `sum`.

QUESTION 27: Execute the script and check the error energy in the spatial domain. Is it equal to what you got before for nearest neighbor interpolation in section 3.1?

QUESTION 28: How can you change the script so that it performs bilinear interpolation instead? Do so and then check the error energy in the spatial domain. Is it equal to what you got before for bilinear interpolation in section 3.2?

QUESTION 29: How can you change the script so that it performs bicubic16 interpolation instead? Do so and then check the error energy in the spatial domain. Is it equal to what you got before for bicubic16 interpolation in section 3.4?
