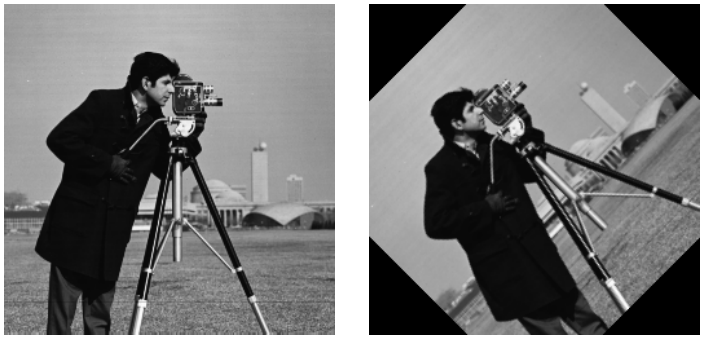
Image Rotation

in Matlab

Rein van den Boomgaard

September 2004

1 Introduction



(a) (b)

Figure 1: **Image Rotation.**

The goal of this lab exercise is to develop an algorithm in Matlab for image rotation. Given a grey value image as shown in Fig. 1.a the image should be rotated over α degrees and stored in a new image as shown in Fig. 1.b.

Your report should contain the following:

- The code of the `pv` function
- The plot of the image profile from $(100, 100)$ to $(120, 120)$ in the gray value version of the `flowers.tif` image. Both using linear and nearest neighbor interpolation and 100 points on the line.
- The code of the `imRotateOrigin` function
- The code of the `imRotate` function
- The `cameraman` image rotated over $\pi/3$ degrees around the origin

The code should be documented to illustrate your understanding of it. To include a plot in your report you can 'cut-and-paste' in case you write your report in Word. For those of you who use LaTeX you can do the following:

- Assuming the current figure shows the plot you want to include in your report use the following command to make an 'eps' file (eps stands for encapsulated postscript): `print -depsc2 filename.eps` (choose a filename yourself).
- In your LaTeX file write `\usepackage{graphicx}` in the preamble. The eps file (and thus the plot) is included in your document with the line `\includegraphics[height=4cm]{filename}` assuming the `filename.eps` file is in the same directory as the eps file.

2 Interpolation

A rotation algorithm is an example of a *geometrical transform*. In these algorithms there is a need to find the value in a point (x, y) that is *not* a sample point. We thus need an interpolation algorithm.

Write a function `pv` with signature `pv(im,x,y)` that calculates the value at location (x, y) using either nearest neighbor or bilinear interpolation. The Matlab function template is:

```
function r = pv( im, x, y, method )
% get a pixel value using interpolation
[M,N] = size(im);
if (x>=1) && (x<=M) && (y>=1) && (y<=N)
  % OK point (x,y) is within image
  if method=='linear'
    % do the bilinear interpolation
  else
    % do the nearest neighbor interpolation
  end
else
  % outside the image --> return a sensible value
  r = 0;
end
```

Remember that in Matlab a function returns a value by assigning a value to the 'return parameter' (in this case the parameter `r`). Also note that in Matlab the indices of an array start at 1.

It is easy to test your function `pv` by using it in a `profile` function that samples the image in N equidistant points along a line from $(x0, y0)$ to $(x1, y1)$. The profile function is:

```
function line = profile( image, x0, y0, x1, y1, N, method )
% profile of an image along straight line in N points
x = linspace(x0, x1, N); y = linspace(y0,y1,N);
for i = 1:length(x)
  line(i) = pv( image,x(i), y(i), method );
end
```

Test this function with:

```
a = imread( 'flowers.tif' );
a = im2double( rgb2gray(a) );
plot( profile( a, 100, 100, 120, 120, 100, 'linear' ) );
```

Experiment with a varying number of points on the profile line while keeping the begin and end point fixed. With many points on the line the difference between nearest neighbor interpolation and bilinear interpolation should become clearly visible.

HINT: in case your bilinear interpolation is not working correctly (or it takes too much time to get it right), implement the nearest neighbor interpolation and continue the exercise. Of course you should also ask the assistant for help or mail to `beeldbewerken@science.uva.nl`.

3 Rotation

The final goal is to rotate around the center of the image. But first let's start with a counter clockwise rotating around the origin $(0, 0)$. For a rotation over an angle ϕ a point (x, y) in the original image is mapped onto the point (x', y') in the resultant image. The relation between the points is:

$$x' = x\cos(\phi) - y\sin(\phi) \tag{1}$$
$$y' = x\sin(\phi) + y\cos(\phi) \tag{2}$$

In a geometric transform we need to express x and y as functions of x' and y'. To get from (x', y') to (x, y) we need to rotate over $-\phi$. Show that we thus obtain:

$$x = x'\cos(\phi) + y'\sin(\phi) \tag{3}$$
$$y = -x'\sin(\phi) + y'\cos(\phi) \tag{4}$$

The rotation algorithm then enumerates all points (x', y') in the resulting image, calculates the corresponding point (x, y) in the original image and uses an interpolation method to estimate the value of the original image in (x, y) and assigns that value to the point (x', y'). The template for the rotation function is:

```
function r = imRotateOrigin( im, angle, interpolationmethod )
sinangle = sin(angle);
cosangle = cos(angle); % precalculate the sine and cosine

[M,N] = size(im);      % the size of the image
r = zeros(M,N);        % allocate the rotated image

% loop over all points (xa,ya)  in the resulting image,
% calculate where the point came from (x,y)
% use interpolation (the function pv) to
% calculate the value at (x,y) in the original image
```

Note than angles in Matlab have to be given in radians. Implement the rotation algoritm and test it for small angles (why?). Can you see the difference between

In case the `imRotateOrigin` is implemented and working, change the function to rotate the image around the center of the image (call that function `imRotate`.