Fundamentals of Image Processing

Tomáš Mikolov and Valiantsina Hubeika, FIT VUT Brno

This practical exercise is dealing with fundamentals of image processing. All examples will be introduces using a grey-scale picture (processing in RGB requires to process each color component separatly).

1 Load and Display of Picture

```
I=imread('lena.gif');
imshow(I);
```

Do not forget to put semicolon after the commands (which return bitmaps)!

1.1 Basic Image Operation

The size of the picture can be determined by `size(I)`, which returns the size of the matrix containig the bitmap. To list the current variables use command `whos`. Here you can see the data type assigned to the varibles. The current image is of the type is `uint8`.

Rotation of the image:

```
Irot=imrotate(I, 30, 'bilinear');
imshow(Irot);
```

Storage to the disk:

```
imwrite (I, 'lenacopy.gif');
```

Exercize

- Determine the size of the image `lena.gif` before and after rotating it by 45 degree.

2 Fundamental Techniques in Image processing

There is a number of thechniques which can be realized by step-by-step processing of each pixel using a predefined function. Usually these operations can be realized in Matlab using ensembled functions. In this lab, all the operations will be done additionally "by hand" in order to learn how they work.

The values in the matrix I (the image is loaded into) lie in range 0-255. Each value determines the intensity of the brightness in the image. (0 corresponds to black, 255 corresponds to white). The values can be displayed using function `colorbar`. Position [0, 0] corresponds to the top left corner. Try:

```
Icopy=I;
Icopy(1:200, 1:400)=zeros(200, 400);
imshow(Icopy); colorbar;
```

2.1 Color Inversion

Inversion of the colors can be implemented as:

```
for (x=1:512)
  for (y=1:512)
    I2(x,y)=255-I(x,y);
  end;
end;
imshow(I2);
```

using a Matlab function:

```
I2 = 255 - I;
imshow(I2);
```

In this lab, most of the examples are implemented using a cycle although the cycle isn't optimal in Matlab. The purpose is to show how to implement the introduced operations in C, Java, etc.

2.2 Simple Edge Detection

Before complicated image processing it is useful to do edge detection for the purpose of detecting the location of the objects. Here, a very simple (primitive) edge detection algorithm is introduced. The idea is to compare the values of the neighboring pixels multiplied by a constant (in this case 10) to make the contrast more distinguishable:

```
for (x=1:511)
  for (y=1:512)
    I2(x,y)=(I(x,y)/2 - I(x+1,y)/2)*10;
  end;
end;
imshow(I2);
```

The difference in brightness of the neighboring pixels is usually significantly bigger than on the mono-color areas, therefore the resulting image will contain edges as white lines.

2.3 Thresholding

The objective of thresholding is to assign a new value to the pixel depending on the current value and the threshold (set, precomputed,...). In this example, first, the threshold is set and the pixels are set to be wither white or black.

```
prah=120;
for (x=1:512)
  for (y=1:512)
    if (I(x,y)<prah) I2(x,y)=0;
      else I2(x,y)=255;
    end;
  end;
end;
imshow(I2);
```

Exercise

- Try different values for the threshold. Which one is the best? Is it possible to precompute the value of the threshold?

3 Histogram

Histogram is a function which computes the number of pixels of a certain color. The matrix I contains 256 colors (0-255). The histogram of the Lena image looks like:

```
imhist(I);
```

Histogram estimation algorithm looks like:

```
Pocet=zeros(256);
for (x=1:512)
  for (y=1:512)
    Pocet(I(x,y))=Pocet(I(x,y))+1;
  end;
end;
plot(Pocet);
```

The histogram shows that some colors are not present in the image, whereas some other colors accure frequently. The histogram can be used for image analysis or/and image processing (correction). The equalization of the histogram looks like:

```
I2=histeq(I);
subplot(221); imshow(I); title ('original'); subplot(222); imhist(I);
subplot(223); imshow(I2); title ('equalized'); subplot(224); imhist(I2);
```

The subplot functions can be switched off by `subplot(111);` or `close all`

4 Linear Filters

In the previous example of edge detection, the detection was done by substraction of the neighboring pixels. Other method how do detect the edges in image is to implement a linear filter which is based on addition of the values of the neighboring pixels. Here are several examples where filters can be used.

4.1 Image Blur

First, try Matlab function `imfilter`:

```
h = ones(5,5) / 25
I2 = imfilter(I,h);
imshow(I2);
```

The matrix `h` represents a filter. Here, the current value of the pixel is substituted by the average value of the current and neighboring pixels:

```
h=ones(5,5)/25
I3=zeros(512,512);
for (x=3:509)
  for (y=3:509)
    for (x2=1:5)
      for (y2=1:5)
        I3(x, y)=I3(x, y)+h(x2, y2)*double(I(x+x2-3, y+y2-3));
      end;
    end;
  end;
end;
imshow(uint8(I3));
```

The averaging is done for each pixel of the image.

4.2 Focalization of the Image

The reverse to the blur is focalization function. The effect will be reached by subtracting the values of the neighboring pixels from the value of the current pixel.

```
h2=[-1 -1 -1; -1 9 -1; -1 -1 -1]
I2 = imfilter(I,h2);
imshow(I2);
```

4.3 Edge Detection

The filter [5 -5] corresponds to the earlier introduced edge detection method. The drowback of this filter is the noise as only one neighboring pixel is concidered. Moreover, such a filter is not capable of detecting horisontal edges. To improve detection usually larger filters are used and the image is passed through the filter twice: for detecting both, vertical and horisontal edges

```
h=[1 0 -1; 2 0 -2; 1 0 -1]
I2 = imfilter(I,h);
h2=h'
I3 = imfilter(I,h2);
imshow(I2/2+I3/2);
```

5 Compression à la JPEG using DCT

JPEG image compression is a well known technique based on DCT. This example explains the basic principal of the thechnique. First, the image is divided into blocks of 8x8 pixels and the 2D Discrete Cosine Transform (DCT) is applied on each of them. The transform decomposes the image into components with different frequency. The high frequency componnents are discarded as they carry least relevant information. When reconstructing the image, Inverse DCT is applied and the 8x8 blocks are reconstructed. Although, the high frequency information loss is evident (the edges are blured), the quality of the restored image is reasonable concedering that 58 out of 64 coefficients were discarded (which is equivalent to 90.625 % information loss).

```
Id = im2double(I);
T = dctmtx(8)                              % definuje matici DCT bazi
dct = @(x)T * x * T';                      % definuje funkci, kteou budeme
       % aplikovat na kazdy 8x8 blok
                                           % prima DCT
B = blkproc(Id,[8 8],dct);                 % tady to udelame
mask = [1   1   1   0   0   0   0   0  % maska, ktera vybere jen
        1   1   0   0   0   0   0   0  % prvnich par koeficientu ...
        1   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0];
B2 = blkproc(B,[8 8],@(x)mask.* x);    % vsechny bloky 8x8 se vymaskuji
invdct = @(x)T' * x * T;                   % definice zpetne DCT
I2 = blkproc(B2,[8 8],invdct);             % kterou zde aplikujeme.
imshow(I), figure, imshow(I2)
```

In reality, the JPEG compression is far more complicated. Less relevant information is not discarded but compressed (stored on fewer bits). Different color components are as well storred in different quality as the human eye persives colors with different sensibility.

Example http://www.mathworks.com/access/helpdesk/help/toolbox/images/f21-16366.html

Task

1. look at the columns and rows of the matrix T, which contains the DCT bases:

```
plot (1:8, T(1:3,:)')   % zobrazi prvni 3 radky
plot (1:8, T')          % zobrazi vsechny radky
```

```
plot (1:8, T(:,1:3))   % zobrazi prvni 3 sloupce
plot (1:8, T)          % zobrazi vsechny sloupce
```

Which frequencies correspond to the given bases?

2. Try to generate the following images:

   - all black.
   - all white.
   - horisontal stripes.
   - vertical stripes.
   - "chessboard".

   Pass the images to the above algorithm.

   - Display the matrix `B` and comment on it. An example:

     ```
     B (1:8, 1:8)
     B2 (1:8, 1:8)
     ```

   - Comment on the quality of the image.

3. The implementation using `blockproc` is very effective, but you don't see what is going on inside the the block. Try to implement the cycle substituting the function.

6 Error in Image

When experimenting with image processing, it is usefull to be able to compute the error (defect). As the measure of the commited error an average error per pixel can be calculated:

```
noise=0;
I2=im2uint8(I2);
for (x=1:512)
  for (y=1:512)
    noise=noise+double(abs(I(x,y)-I2(x,y)));
  end;
end;
noise=noise/512/512
```

7 Reference

http://www.mathworks.com/access/helpdesk/help/toolbox/images/f0-3373.html
http://www.fit.vutbr.cz/study/courses/ISS/public/pred/2d/aux.m

8 Solution

DCT "by hand":

```
Id = im2double(I);
% prima dct
T = dctmtx(8)
B = zeros (512,512);
for x=1:8:511,
  for y=1:8:511,
    vysek = Id (x:(x+7),y:(y+7));
    dctvyseku = T * vysek * T';
    B(x:(x+7),y:(y+7)) = dctvyseku;
  end
end
% maskovani
mask = [1   1   1   0   0   0   0   0  % maska, ktera vybere jen
        1   1   0   0   0   0   0   0  % prvnich par koeficientu ...
        1   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0];
B2 = zeros (512,512);
for x=1:8:511,
  for y=1:8:511,
    vysek = B (x:(x+7),y:(y+7));
    vymaskovano = vysek .* mask;
    B2(x:(x+7),y:(y+7)) = vymaskovano;
  end
end
% zpetna DCT
I2 = zeros (512,512);
for x=1:8:511,
  for y=1:8:511,
    vysek = B2 (x:(x+7),y:(y+7));
    zpet = T' * vysek * T;;
    I2(x:(x+7),y:(y+7)) = zpet;
  end
end
imshow(I), figure, imshow(I2)
```