Signal and image processing: Assignment 1

Mariuta Nicolae 04-09-2014

Exercise 1.1.

```
%1.1
A = rand([20 20]); %Generate random image
imagesc(A),colormap(gray);
xlabel('X');
ylabel('Y');

while (true)  %Infinite loop to read clicked pixel
    [xVector,yVector] = ginput(1);
    x = int64(xVector(1));  %coordinates of clicked pixel
    y = int64(yVector(1));
    disp(sprintf('X=%u', x));  %display coordinates in console
    disp(sprintf('Y=%u', y));
    A(y,x) = 0;                 %change clicked pixel color to black
    imagesc(A),title(sprintf('Clicked (X=%u,Y=%u)',x,y)); %show image again
with changed pixel color
    xlabel('X');
    ylabel('Y');
end
```

Fig. 1 Code for generating the random image and listen for clicked pixel.

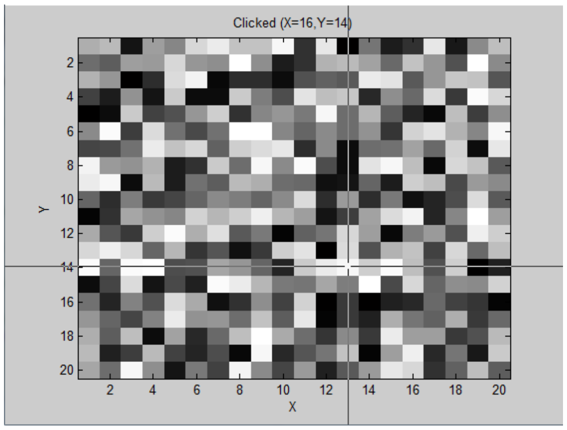

Fig. 2 Picture before clicking pixel

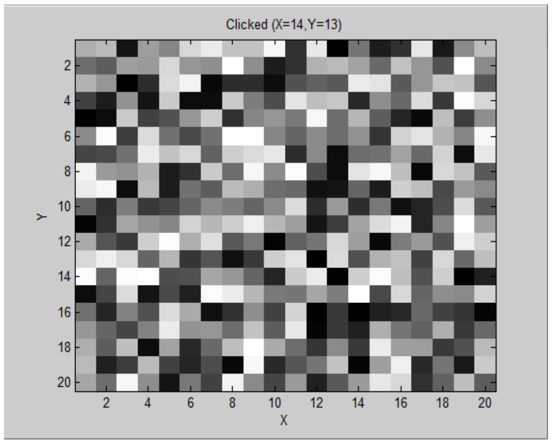

Fig. 3 Clicked pixel becomes black. Coordinates are shown in picture title

Exercise 1.2.

```
range = 1;
B = rand([100 100])*range;  %generate random image

subplot(1,3,1);imshow(B),title('imageshow'), colormap(gray);
subplot(1,3,2);image(B),title('image');
subplot(1,3,3);imagesc(B),title('imagesc');
```

Fig. 4 Code for generating random image and display using all 3 functions for comparison

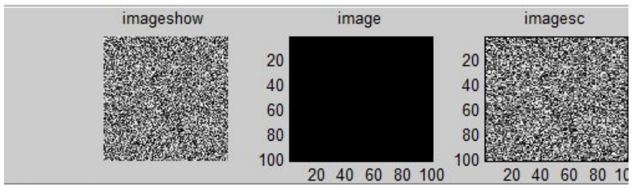

Fig. 5. Random image for range = 1

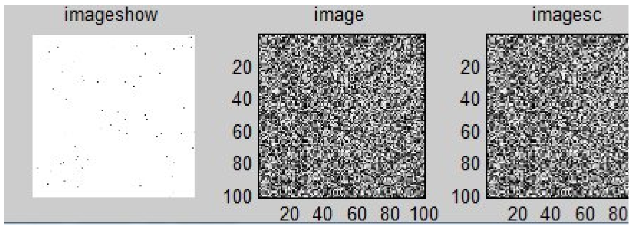

Fig. 6. Random image for range = 255

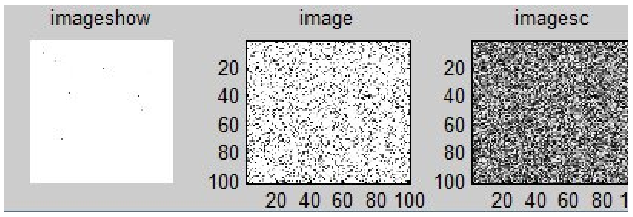

Fig. 7. Random image for range = 1000

By using different values for the variable "range" which defines the range for generating the random numbers, I have noticed the following:

- function imshow uses number between 0 and 1 to define the pixels intensities(0-black and 1-white). Pixels with value above 1 are white
- function image uses number between 0 and 255 to define the pixels intensities(0-black and 255-white). Pixels with value above 255 are white
- function imsagesc scales the pixel intensity for the interval of numbers are in the matrix(0-black and max value = white)
- imageshow keeps the image at its original size while image and imagesc scale the image to fit in the window

Exercise 1.3.

```
%1.3
A = imread('cameraman.tif');    %load image
subplot(5,2,1);imshow(A),axis off,title('original'); %display original image

for index = 1 : 8
    slice = bitget(A,index) %use bitget function to obtain the bit plane
    subplot(5,2,index+1);imshow(slice,
[]),colormap(gray),title(sprintf('bit%u',index));
end
```

Fig. 8. Code which loads a greyscale image, use the bitget function to get only one bit from all the pixels in the image and then displays all the bit planes

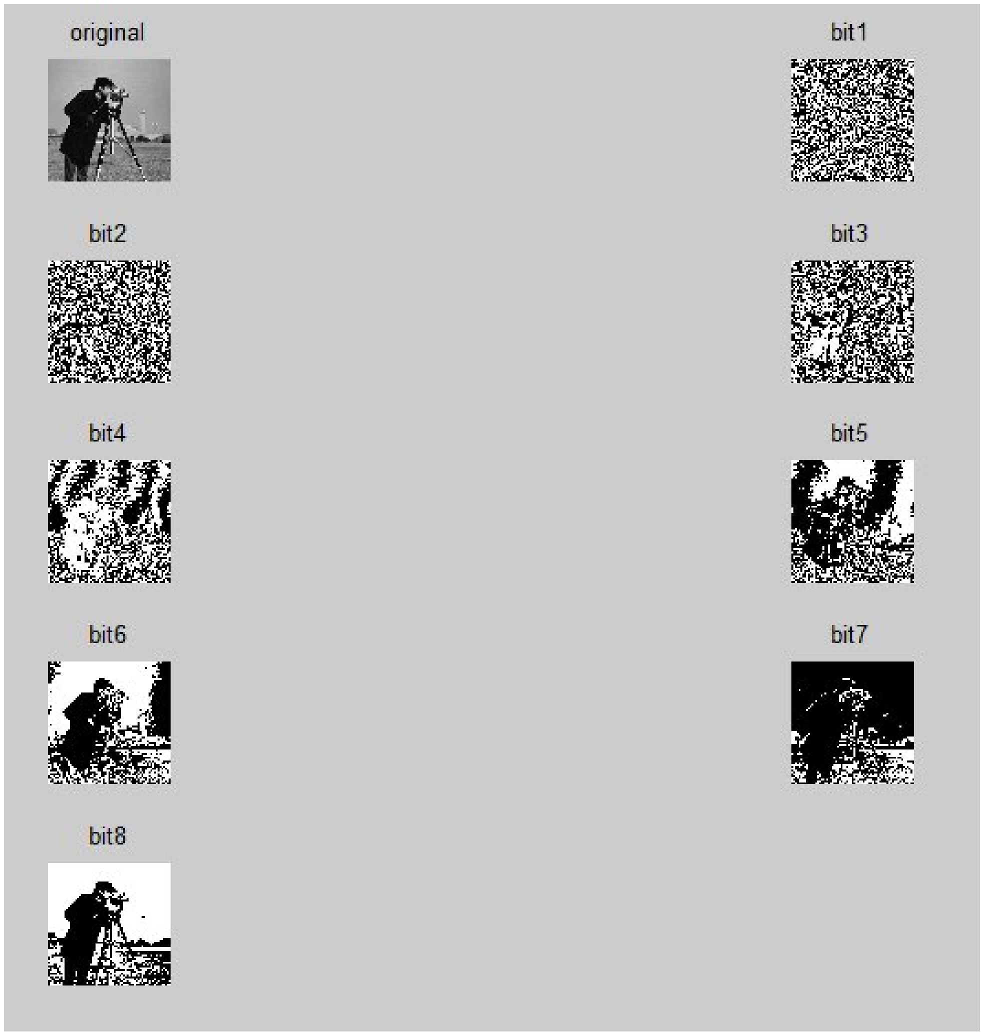

Fig. 9 Original image and results for applying the bit-plane slicing

Exercise 1.4.

```
%1.4
A = imread('D:\master\image processing\examples\onion.png');

if(ndims(A) == 3) %check if image is in RBG format
   hsv_image = rgb2hsv(A)  % turn image to hsv
   Ahue = A(:,:,1);
   Asaturation = A(:,:,2);
   Avalue = A(:,:,3);
end

subplot(2,2,1); imshow(A); axis image;%display original image
subplot(2,2,2); imshow(Ahue); title('hue'); %display hue
subplot(2,2,3); imshow(Asaturation); title('saturation'); %display saturation
subplot(2,2,4); imshow(Avalue); title('value');%display value
```

Fig. 10. Matlab code that checks if image is rgb, turns it into hsv format and then splits into 3 images containing each of the hue, saturation and value. In the end it displays the results

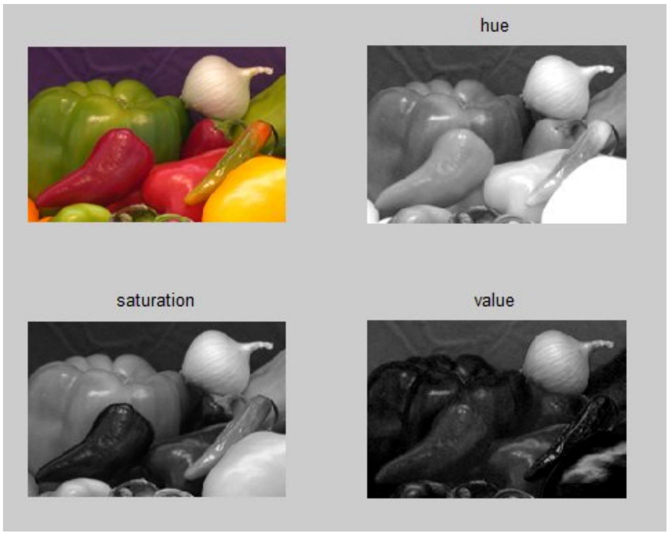

Fig. 11. Original image and each channel of HSV format.

Exercise 1.5.

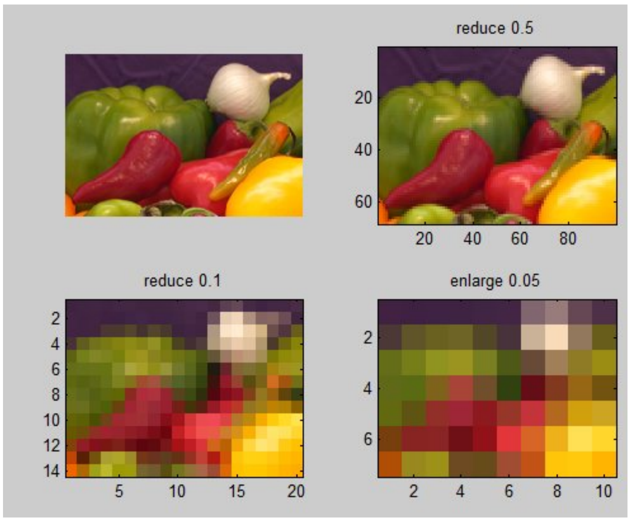

Fig. 12. Usage of the imresize function with different values

In Fig. 12 it can be observed that reducing the number of pixels deteriorates the quality of image until the objects can no longer be identified

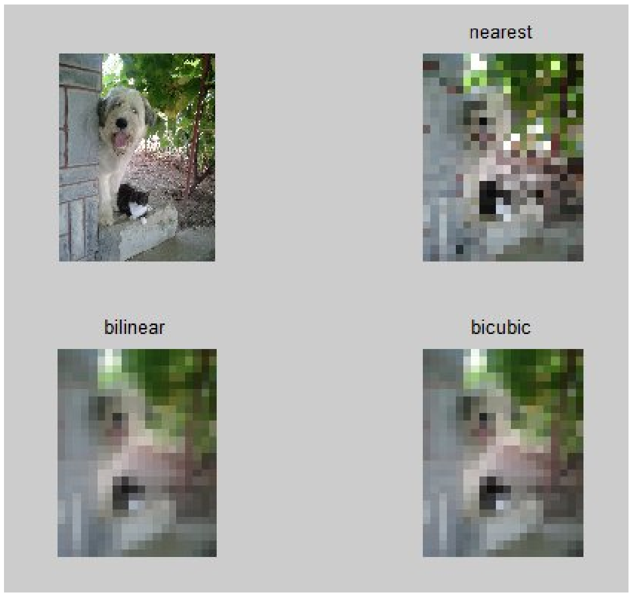

Fig. 13 Usage of different interpolation methods to resize a photo

Using the nearest interpolation method, the shape of objects is altered because it only considers one pixel in the location of the output pixel and the others are ignored, but it is the fastest method because it has a simple algorithm (0.016338 seconds).

For the other 2 methods, the image is blurred but objects can still be distinguished because it calculates the average of the pixels in the area where the output pixel is placed, using different size matrix of pixels(2x2 for bilinear and 4x4 for bicubic). But the computation times are larger (0.034950 seconds for bilinear and 0.051946 for trilinear).

Exercise 1.6.

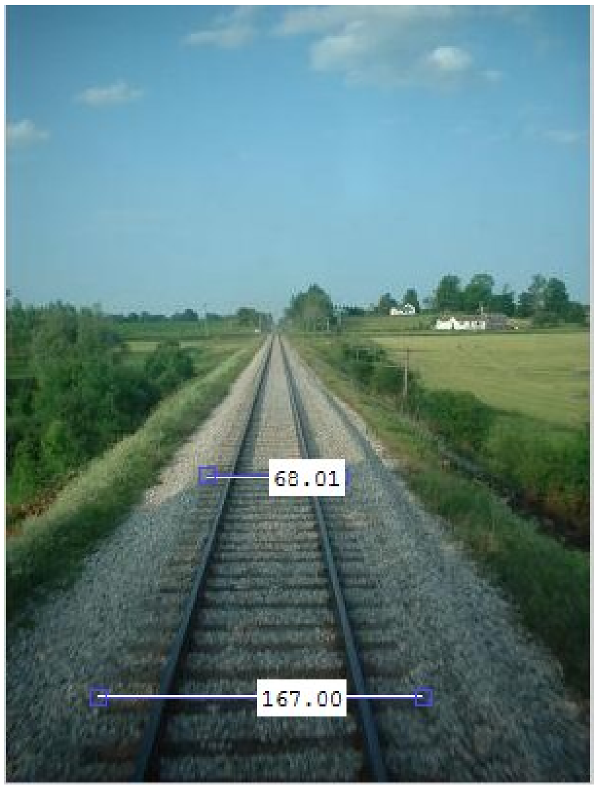

Fig. 14. Relative distance for background and foreground railway sleepers

If the focal length parameters is also known, using the perspective projection formulas, and the relative length of the railway sleepers we can determine the relative distance between the camera and the railway sleepers.

Exercise 1.7.

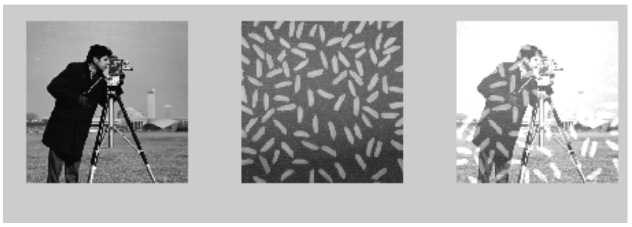

Fig. 15. Resulting image for using the + operator for adding the images

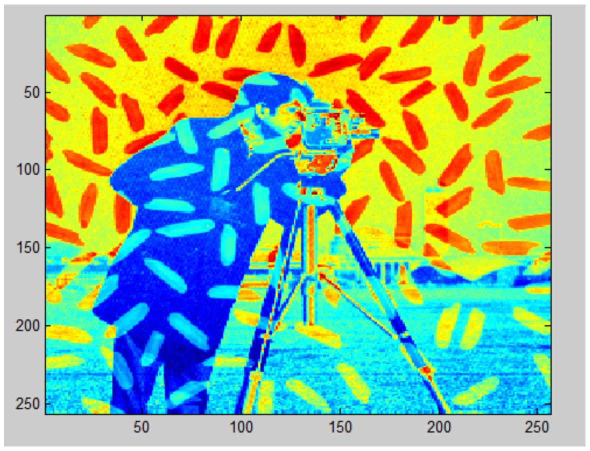

Fig. 16. Resulting image using the imadd function and adding the parameter 'uint16'

Using the + operator, there will be many pixels with value higher than 255 and that is why many of the pixels became white. But forcing the result into a 16bit value for each pixels, the resulting pixels has 2 channels of color: red and blue(8 bits for each color), and that is why the picture is colored.

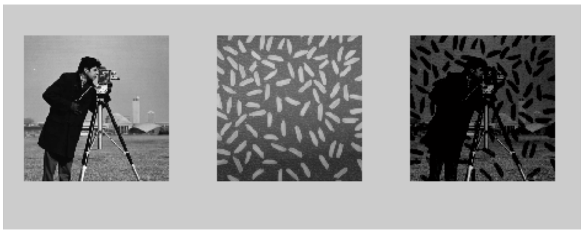

Fig. 17 Resulting image after using the – operator to subtract the images

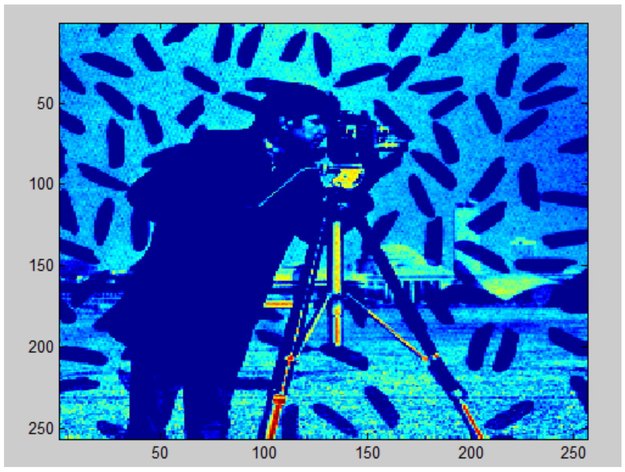

Fig. 18 Result for using the imsubtract() function

Exercise 1.8.

```
%1.8
names = [
    'D:\master\image processing\examples\AT3_1m4_01.tif'
        'D:\master\image processing\examples\AT3_1m4_02.tif'
        'D:\master\image processing\examples\AT3_1m4_03.tif'
        'D:\master\image processing\examples\AT3_1m4_04.tif'
        'D:\master\image processing\examples\AT3_1m4_05.tif'
        'D:\master\image processing\examples\AT3_1m4_06.tif'
        'D:\master\image processing\examples\AT3_1m4_07.tif'
        'D:\master\image processing\examples\AT3_1m4_08.tif'
        'D:\master\image processing\examples\AT3_1m4_09.tif'
        'D:\master\image processing\examples\AT3_1m4_10.tif'
        ];  %array with paths to images

prev = imread(names(1,:));   %store the first image

for R = 2 : size(names)
   im= imread(names(R,:));      %read next image
   mosaic = imabsdiff(im,prev); % calculate abs diff
   subplot(3,3,R-1),imshow(mosaic); %display image
   prev = im;                   % store previous image
end
```

Fig. 19. Code that calculates absolute difference between successive image in an array of cell images

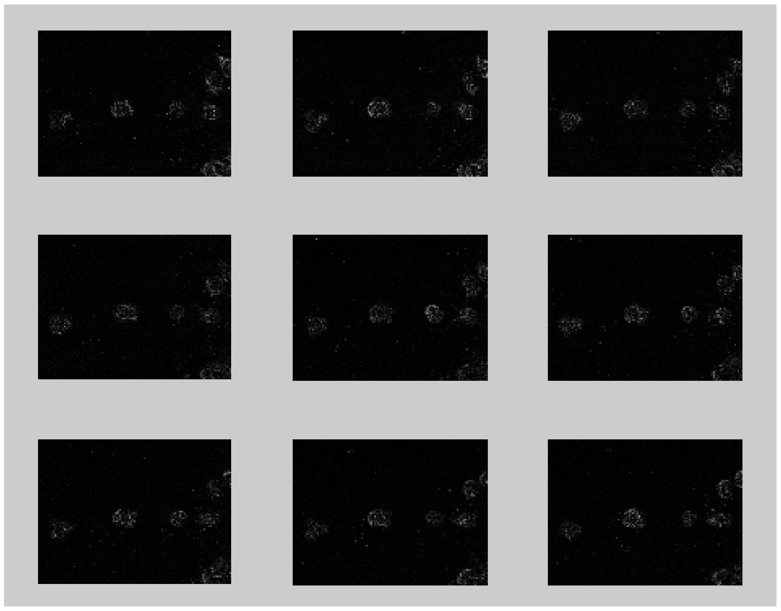

Fig. 20 Resulting images.

The results are showing the movement of the cells. This could be useful for tracking the movement of different object: cells, cars, people.

Exercise 1.8.

```
function O = blend( A,w_A,B,w_B )
    O1 = immultiply(A,w_A);
    O2 = immultiply(B,w_B);
    O = imadd(O1,O2);
end
```

Fig. 21 Implementation of blending function, where the output image is the result of C = w_A . A + w_B . B operation, where A,B are images and w_A and w_B are weights

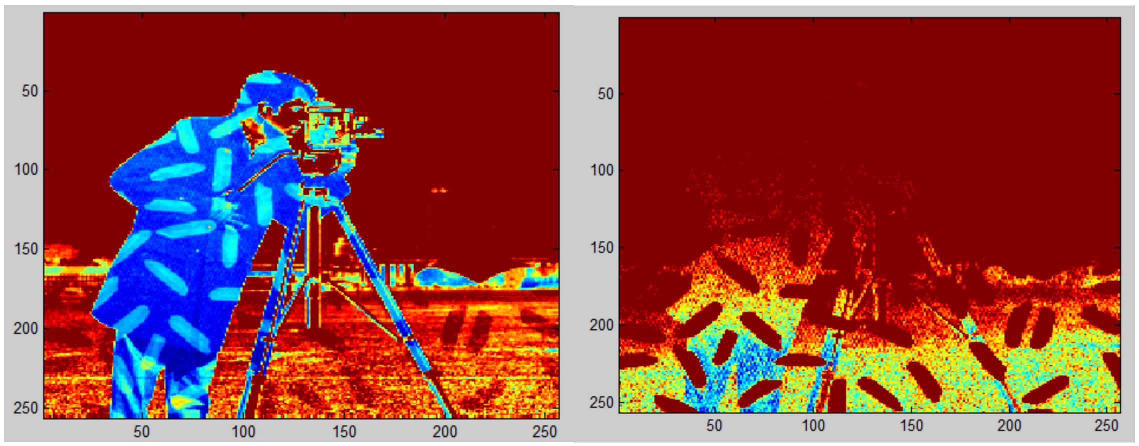

Fig. 22 Results for blending two images with different weight values