Computer exercise 1
Introduction
Spatial statistics and image analysis, TMS016

1 Introduction

The purpose of this computer exercise is to give an introduction to some basic procedures in Matlab for working with images and spatial data. If you need a more thorough introduction to Matlab, see the links on the course homepage. When in doubt about how to use a specific function, use `help` and `doc` to get more information.

Before you begin, download the files for the exercise from the course homepage. The files contain the image `chalmersplatsen.jpg`, which you can use for the tasks below, but you can also try using some other images.

2 Loading and viewing images in Matlab

To load the image `chalmersplatsen.jpg`, write

```
>> x = imread('chalmersplatsen.jpg');
```

A digital color image is a collection of pixels, where each pixel is represented using three components: red, green, and blue (RGB). In Matlab, color images are represented using 3-dimensional matrices, where the third index denotes the colour components. To get the size of the image, write

```
>> [m,n,d] = size(x);
```

Here m and n are the number of pixels in each direction, and $d = 3$ is the number of color components. The color values for a single pixel (i, j) is given by `x(i,j,:)`, whereas a specific color component is given by `x(:,:,id)`.

Each color value is given by a number from 0 to 255, which is represented by an Integer value in matlab. For easier computations, it is often a good idea to start by transforming the values do double values in the interval $[0, 1]$:

```
>> x = double(x)/255;
```

We may now visualise the image using

```
>> imagesc(x);
>> axis image;
```

The second line is here needed to get the correct aspect ratio of the image. Alternatively, the `imshow` function can be used. For example, to view the red color component, write

```
>> imshow(x(:,:,1));
```

- Try obtaining the same image using `imagesc` by changing the colormap that is used for the image. See `doc colormap` for various colormaps that are available in matlab.

3 Color manipulations

- Test visualizing the image where some of the color components are switched. To for example change the order of the first two color components, you can write `x(:,:,[2 1 3])`.

We may often want to work with a grayscale version of the image. An easy way to obtain a grayscale version is to simply average over the three color components.

```
>> y1 = mean(x,3);
```

However, a slightly more sophisticated conversion can be obtained using

```
>> y2 = rgb2gray(x);
```

This function converts to grayscale by eliminating the hue and saturation information while retaining the luminance of the image.

- Plot the two grayscale images as well as their difference.

To get a feeling for the distribution of the pixel values in an image, the `imhist` function can be used. Try this for the grayscale image. Histograms like this will be important later on when we do segmentation of images. For segmentation, the relative amount of each color in a color image can also be of importance. For example, the relative amount of green in a pixels is G/(R+B+G).

- Generate an image where each pixels shows the relative color amount. To avoid using a for-loop to go through the pixels, use the `repmat` function.

Another way of obtaining an intensity-free representation is to use the AB components of the image when represented in LAB color space instead of RGB. Converting to LAB can be done using `rgb2lab`.

- Plot the three components of the LAB representation of the image. What do they represent? See https://en.wikipedia.org/wiki/Lab_color_space for the answer.

When we later do segmentation of images, a useful way of representing the result is as an $m \times n \times K$ matrix where the third component represent the probability of belonging to each class. Create a segmentation of the image, with $K = 4$, based on thresholds for the different color components

```
>> z(:,:,1) = xrel(:,:,2)>0.4;
>> z(:,:,2) = xrel(:,:,2)<0.4 & xrel(:,:,1)<0.3;
>> z(:,:,3) = xrel(:,:,2)<0.4 & xrel(:,:,1)>0.3 & xrel(:,:,3)<0.3;
>> z(:,:,4) = xrel(:,:,2)<0.4 & xrel(:,:,1)>0.3 & xrel(:,:,3)>0.3;
```

If we want to display an image of this classification, we have to convert this multispectral image to an RGB image. There are several ways in which this can be done. But using the fact that the third component represent probabilities, a natural way is to construct an image I with pixel values

$$I(i,j) = \sum_{k=1}^{K} z(i,j,k) RGB_k$$

where RGB_k is a vector with the RGB color we want for the kth class. For example $RGB_1 = [1\,0\,0]$ would give red as the color for the first class.

- Write a function

  ```
  function I = classification2rgb(z,RGB)
  ```

  that produces such an image, where RGB is a matrix where the kth row is the RGB_k vector. Use this function to display the classification z using some different choices of colors.