Digital image processing and analysis
Laboratory 7: Image segmentation

Vedrana Baličević, Pavle Prentašić, Hrvoje Kalinić, Sven Lončarić

2014.

5.1 Introduction

5.1.1 Problems

1. What does the function *kmeans*() do and how?
2. What does the function *reshape* do?
3. Write a function that calculates energy of the signal without the DC component (the same way you wrote functions for moment and entropy calculation in the previous exercise).

5.2 Amplitude segmentation

Amplitude segmentation is the simplest way to segment an image. It is useful (efficient) when amplitude (the intensity of the pixel) defines the scene regions precise enough. To perform amplitude segmentation, we can use first order histogram.

5.2.1 Manual determination of treshold

In this part of the exercise, we will determine the segmentation threshold based on the histogram. If there are several regions with the more-or-less uniform color within each of them, then we expect histogram to be bimodal or multimodal. The interpretation of the histogram depends on the histogram itself and/or knowledge of the image content (e.g. number of regions and similarity of their colors).

An example of such histogram is given in the Fig. 5.1. In this histogram we see several groups, and we would choose a threshold value 130 for segmentation. An optimal choice of the threshold corresponds to the value between two maxima.

For amplitude segmentation we can use function *grayslice*() which will segment the input image with thresholds that are given as a second input parameter. Thresholds are given as a vector and have to be in [0,1] range. We can save the output image in the new variable. However, function *grayslice*() has a property that it displays the output immediately if the name of the output variable is not given. Working with *grayslice*() is explained in the following example.

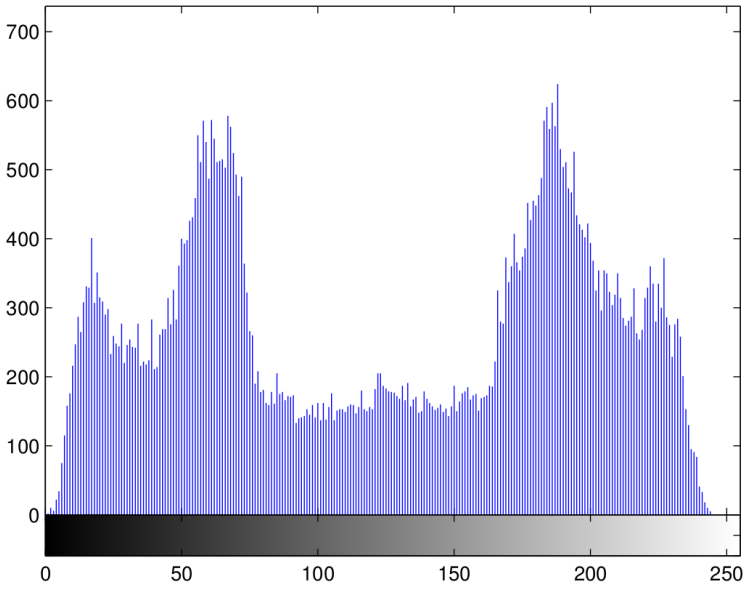

Figure 5.1: First-order histogram of an image.

An example

```
>> [img,map] = imread('testpat1.tif');       % read the image
>> imhist(img)                               % calculate and show histogram
%  If we interpret histogram as bimodal, then we select 1 threshold:
>> grayslice(im2double(img),130/255)         % segment and display
%  If we interpret histogram as multimodal (three-modal), then we select 2 thresholds
%  (here: 45/255 and 130/255):
>> grayslice(im2double(img),[45 130]/255)    % segment and display
```

5.2.2 Problems

1. Read the image $blood1.tif$, showing blood cells. We want to segment the image so that cells are represented with an annulus (two concentric circles) with as little noise as possible. In the segmented image, blood cells will have a cavity in the middle. Display the histogram, determine an optimal threshold t, and display the segmented image.

2. For the same image display the results if you select a threshold $t+0.2$ and $t-0.2$ instead of threshold t selected before. Describe what happened.

3. If we interpret the histogram of the same image as three-modal, where would you define the second threshold? Segment the input image again with the two thresholds. What happened?

5.3 Automatic determination of the threshold

In this part of the exercise, we will demonstrate the success of $K-means$ algorithm in automatic determination of the threshold. We will use MATLAB's function $kmeans()$. Output variable c of the function

contains the calculated centers of the classes in which we group the image pixel intensities.

An example

```
>> [img,map]=imread('testpat1.tif');              % read the image
>> [idx, c] = kmeans(im2double(img(:)), 3);       % perform the segmentation
>> figure; imagesc(reshape(idx,size(img)));       % display the segmentation result
```

Iterative calling of the function gives differences in centers c. This happens because we didn't define the initial centers as an input parameter. If we define the initial centers, ie. $(\frac{1}{6}, \frac{1}{2}, \frac{5}{6})$, the centers will remain constant. The last two parameters are used for speeding the algorithm up (but on the cost of the precision). However, you can call the function with this parameters til the end of this exercise.

```
>> [idx, c] = kmeans(im2double(img(:)),3, 'start', ...
[0.5:3]'/3, 'onlinephase', 'off');
```

5.3.1 Problems

1. Read the image *blood1.tif*. Try the segmentation using the k-means algorithm. Compare the threshold values obtained with the manual approach.

5.4 Extraction of edges

For finding the edges in the image, we can use function *edge*(), which supports several methods for finding the edges. The function gives a binary image (same dimensions as the input image), which contains only the detected edges from the input image. Supported methods are: *sobel*, *prewitt*, *roberts*, *zerocross*, *log* and *canny*.

Methods based on the first order derivative calculate the derivative estimation for each pixel. Amplitude segmentation is then applied to this image. In function *edge*, threshold is defined internally.

Methods based on the second order derivative detect zero-crossing of the intensity values. In this case, the threshold corresponds to the steepness of the edge.

An example

```
>> [img,map]=imread('saturn.tif'); % read the image
>> rub=edge(img,'sobel');          % apply Sobel method
>> imshow(rub)                     % display the edges image
>> rub=edge(img,'sobel',0.02);     % we can also define the threshold
>> imshow(rub)                     % display these edges too
>> rub=edge(img,'log',[],4);       % apply LoG method with std. dev. 4
>> imshow(rub)                     % display these edges too
```

5.4.1 Problems

1. Read the image *4.2.07.tiff* from the USC-SIPI database. Run the *edge*() function on the image, using the 'Sobel' operator method, with different threshold values. Which threshold value gives the best edge detection?

2. Run the *edge*() function on the image *4.2.07.tiff*, using the a) 'Sobel', b)'zerocross' or 'log' and c) 'Canny' method, with automatic choice of threshold. Which method works the best?

3. Is the automatic threshold for 'Sobel' different than yours? Which one seems to give better result?

4. Add the noise to the original image using the *imnoise*() function and repeat the edge detection with the 3 operators. How does the noise affect the edge detection for these cases?

5.5 Texture segmentation

We will segment the textures based on selected features. First, for each pixel in the image, we will calculate the selected feature on the defined neighboorhod of the pixel Image containing the feature values is the image we will segment.

In the previous exercise we used the function *colfilt()* to calculate the texture features. Here, the output of this function will serve as an input for a function *kmeans*() to perform automatic amplitude segmentation.

An example

```
>> [img,map] = imread('texture.tif');
>> p = [2,3];                                % define the shift
>> fun = @(x) inertia(x,p);                  % function pointer (function handle)
>> znacajke = nlfilter(img,[10 10],fun);     % process the image
>> figure; imagesc(znacajke); colormap(gray)
>> [idx, c] = kmeans(im2double(znacajke(:)), 5, 'start', ...
[0.5:5]'/5, 'onlinephase', 'off');
>> figure; imagesc(reshape(idx,size(img)));  % display the segmentation result
```

1. Read the image *texture.tif*. Display the image. How many textures are there in the image? Describe them.

2. Select several features and calculate them on the blocks of size 12×12. Display the calculated features and estimate which ones can be used to segment given structure. For the selected images apply the K-means method and comment on the results.

3. Calculate the spectra energy (without the DC component) feature on the *texture.tif* image, on the blocks of size 12×12. Is this feature good for segmentation of the textures on this image? Segment the energy image using the K-means method and comment on the results.