Faculty of Mechanical Engineering and Robotics

Department of Robotics and Mechatronics

Basics of AI and Deep Learning

Course for Mechatronic Engineering with English as instruction language

Instruction 7:

Image Processing: Image Segmentation

You will learn: what is image segmentation, what are basic segmentation algorithms for greyscale images, thresholding, how to implement multi-level segmentation using logical operations on images, how to segment color image in RGB and HSV color spaces, what are texture filters and how to apply them in segmentation, what is K-Means algorithm in image segmentation, what are morphological operations applied for binary images, what are objects in the binary image and how to find object features which are applied in object classification

Additional materials:

- Course lecture
- Image Database

Course supervisor:
Ziemowit Dworakowski, zdw@agh.edu.pl

Instruction author:
Krzysztof Holak, holak@agh.edu.pl

Image Segmentation

In the last class, you have seen a simple algorithm of image segmentation using thresholding operation. Let's start with reviewing the concept.

Thresholding converts a grayscale image into a black and white image. It is based on the binarization threshold. For example, pixels with intensity value less than the threshold are given value of 0, and pixels with a value greater than the threshold are given value of 1.

In MATLAB, binarization can be carried out by **`im2bw()`** function or inequality operator as discussed in the previous laboratory instruction.

Pixels of binary image can be understood as logical variable (0 and 1). Therefore, any logical operation can be performed on binary image matrices. For example:

1) Logical intersection - **`and(im1,im2);`**

2) Logical union - **`or(im1,im2);`**

3) Logical negation - **`not(im1)`**;

Arithmetic operations are also performed on images (binary and grayscale):

4) Addition - **`imadd(im1,im2);`**
5) Subtraction **– `imsubtract(im1, im2);`**
6) Multiplication **– `immultiply(im1,im2);`**

Here, there is a sample code for image binarization (with an application of two methods):

```
clc;
clear all;
close all;

imRGB = imread('Crow.jpg');
imGray = rgb2gray(imRGB);

thresh1 = graythresh(imRGB); % an automatic threshold computation, try also your own
chosen thresholds

thresh2 = floor(thresh1*255); % determine threshold value in the range 0-255

imBW1 = im2bw(imGray,thresh1);

imBW2 = imGray > thresh2;

figure
subplot(1,2,1);
imshow(imBW1);
subplot(1,2,2);
imshow(imBW2);
```

Morphological operation on binary images

Morphological processing uses set operations to make changes to images. They can be

applied for both binary and grayscale images (as was discussed in the previous laboratory exercises). In the general case, a morphological operation requires moving a mask, called a structural element, along the image and determining, for each position of the mask, the resulting value of the logical operation between the pixels of the mask and the pixels of the image lying directly under the mask. The effect of the most important morphological operations for binary images will be described below.

Remember that in MATLAB, **objects** have **white** pixels (logical ones) and the **background** pixels are **black** (logical zeros).

To use a morphological filter, you need to create a mask (a structuring element)

```
se1 = strel('type,size);
```

Where ' type ' denotes the type of the structuring element and size specifies its size. After creating the mask, you can apply one of the selected morphological filters. The basic filters are:

1) **Erosion** - filter removes noise in the form of objects with a size smaller than the size of a structuring element, reduces the surface area of all objects, disconnects objects connected by thin bridges

   ```
   bw_erode = imerode(im_bw,se1);
   ```

2) **Dilation** - filter removes noise in the form of "holes" in objects, increases the surface of all objects, connects objects that are close to each other

   ```
   bw_dilate = imdilate(im_bw,se1);
   ```

3) **Opening** - is a combination of successive erosion and dilatation. Deletes noise in the form of small objects, but does not change the surface area of large objects.

   ```
   bw_open = imopen(im_bw,se1);
   ```

4) **Closing** - the filter is a combination of successive dilatation and erosion. Deletes noise in the form of "holes" in objects, but does not increase their surface areas.

   ```
   bw_close = imclose(im_bw,se1);
   ```

In MATLAB, to apply morphological filters, you can also use the function **bwmorph()**

A sample code showing how to use these operations:

```
clc;
clear all;
close all;

imRGB = imread('Crow.jpg');
imGray = rgb2gray(imRGB);

thresh = graythresh(imRGB); % an automatic threshold computation, try also your own
chosen thresholds
imBW = im2bw(imGray,thresh);
```

```
se = strel('square',5);

imErode = imErode(imBW,se);
imDilate = imdilate(imBW,se);
imOpen = imopen(imBW,se);
imClose = imclose(imBW,se);


figure,
subplot(2,2,1)
imshow(imErode);
title('Erosion')

subplot(2,2,2)
imshow(imDilate);
title('Dilation')

subplot(2,2,3)
imshow(imOpen);
title('Opening')

subplot(2,2,4)
imshow(imClose);
title('Closing')
```

Now, let's practice what you know so far on practical exercises

Task 2.1 Write a script to carry out simple morphological operations on binary images. Carry out image thresholding to get binary image with objects present. Apply four morphological operations: dilation, erosion, opening and closing. Apply 3 different sizes of filter mask and 3 different types of masks

Image Analysis Basics – Object Detection

The main reason why we carry out an image segmentation is to make the image simpler for further image analysis. In the binary image the objects are separate from the background which simplify the computation of any features associated with the objects.

Object recognition and classification are one of the most important tasks of computer vision. In the lab, you will learn how to compute image features which are need for these tasks based on a binary image. In order to analyze and detect object, the image has to be prepared first. Noise has to be removed and grayscale to binary conversion must be performed. Objects are represented by pixels of value 1 and background pixels have value 0.

First, all objects have to be labeled using **`bwlabel()`** function in MATLAB. This function simply gives numbers (a labels) to all disconnected objects in the image.

```
[imLabel, num] = bwlabel(imGray, nb);
```

Function takes binary image and type of neighborhood for object's pixels. 4 and 8 pixel neighborhoods can be used. The function returns a labelled image in which each object has assigned pixels' value equal to its numbers given.

Next, the geometric parameters describing shapes are computed using **`regionprops()`** function. The syntax of the function is as follows

```
features = regionprops(imLabel, properties);
```

where the variable **properties** is a string of characters describing which geometrical features you want to obtain. The simplest choice of properties are 'all' and 'basic'. The first one means that all features has to be calculated and the second one gives only the basic ones. The features are stored in a matrix of MATLAB structures which stores the numerical values of all features computed for all objects.

Here's there is a simple program that computes the basic features.

```
clc;
clear all;
close all;

imRGB = imread('Objects.jpeg');

figure(1)
subplot(1,2,1)
imshow(imRGB);
title('Original Image')

imGray = rgb2gray(imRGB);

thresh = graythresh(imGray);
imBW = im2bw(imGray,thresh);

figure(1)
subplot(1,2,2)
imshow(imBW);
title('Binary Image')

imLB = bwlabel(imBW,4);
features = regionprops(imLB,'basic');
```

Task 2.2 Run the code for binary objects analysis. Try it on images with different types of shapes provided by the instructor or made in a graphics software. First you have to binarize the image and label the objects present on it. See how the values of features changes with each object.

Task 2.3 Binarize the image given by the instructor in order to extract only one specific object from it. After binarization, only one white object must remain, all noise have to be removed. The object should be as close to the original one in shape and size as possible.

In case of most images, it is not enough to choose one binarization threshold. You may need to write a function that requires specifying two binarization thresholds and applying the logical operations.

Here is a small code that may be helpful in solving task 2.3

```
clc;
clear all;
close all;

imRGB = imread('YourImage.jpg');
imGray = rgb2gray(imRGB);

% Here, maybe you need to apply some preprocessing like filtering or morphological
operations applied to the grayscale image

% Here, you have to write your code for binarization, as described in the Task 2.1

% Here, you may apply additional filtering of binary image to get rid of the noise
```

```
% Diplay your results
figure,
subplot(1,2,1)
imshow(imGRay);
title('Original Image')

subplot(1,2,2)
imshow(myResult);
title('My Result')
```

Texture-based Segmentation

Threshold-based segmentation only works when the objects and background are approximately uniform in brightness or color. In the case of textured objects, this method is very difficult to implement in practice and may lead to objects with wrong shapes.
Texture filters can be used to detect areas that differ in texture. There are three basic texture filters in MATLAB:

```
entropyfilt();
stdfilt();
rangefilt();
```

Each filter calculates a certain function for each pixel in the image, with values depending on the brightness distribution of the pixels in its surroundings. By default, the pixel neighborhood is 3x3, but you can also be specified it as a parameter.

For example, the **`stdfilt()`** filter calculates the standard deviation of the pixels surrounding the analyzed pixel. It can be used to separate homogeneous areas from areas where brightness variability is large. Each of these filters simplifies the image so that its subsequent segmentation is possible using standard thresholding.

The images resulting from the texture filters may not have the pixel values in the standard range supported by image displaying and processing function. Therefore, before further processing, you may need to normalize the pixel values to the standard range.

Also, you can see the results of these filter application directly, if you use other function to display images than standard **`imshow()`**. For example, you can use **`image()`** or **`imagesc()`** commands that display matrix content as an image, using pseudocoloring method.

Task 2.4. Write a program to segment texture images provided by an instructor. Keep in mind that different filters may work best for different images and textures (that's why there are three of them to test).

K-Means image segmentation method

K-Means clustering is an unsupervised classification method. It partitions a given N-dimensional data into K clusters based on K centroids. The number of clusters K have to be known in advance and is provided by a user. The dimension of the feature space depends on the number of attributes computed for image pixels. In the standards color image segmentation, the feature vector is 3-dimensional and contains values of 3 color channels. The data is clustered into K clusters by iterative computation of the K clusters centroids

and computing membership of each data point to the cluster with the nearest centroid.

Let's segment a color image using K-Means clustering. After loading the RGB color image into the MATLAB workspace, decide how many channels you want to use for segmentation.

`im_channels = im1(:,:,1:3);` - in this example, all 3 color channels are used. Later you can change the number of channels and see the results of the segmentation. It would be more helpful in the case of other color spaces in which color information is separated from the intensity information.

To perform K-Means clustering, write a code

```
features = im_channels;
[labels, cmap] = imsegkmeans(features,numClusters);
im_label = label2rgb(labels);
```

The matrix feature in this example is the same as our channels of color image, but later it will contain more pixel attributes as features.

Note that we use a function **`label2rgb()`** to convert segmented and labelled image to RGB for the sake of better visibility of our regions.

Task 2.5 Write a code for K-Means color image segmentation. Next, change the number of color channels (reduce the feature space dimension) and see the result of the segmentation. Try to segment different images, each time choosing a number of clusters based on the image content. Next convert the image in RGB into other color space (HSV, CIE L*a*b*) and repeat the exercise.

Next, we add more features to our feature space. First we know that pixels belonging to the same object should be spatially closer to each other. Therefore, the position of the pixel in the image (X,Y coordinates) may be used as a new attribute. First, let us prepare a two new matrices which for each pixel store the X and Y coordinate of the pixel, respectively.

```
nrows = size(im1,1);
ncols = size(im1,2);
[X,Y] = meshgrid(1:ncols,1:nrows);
```

And augment our feature matrix

```
features = cat(3,im_channels, X,Y);
```

Another useful attribute of the pixel, especially when dealing with natural images, are features describing image textures. One of the most common filter used in texture analysis is Gabor filter, named after Dennis Gabor (1900-1979), who first proposed it for 1D signal processing. The filter response shows as whether there is a specific frequency content in the specified direction in a localized region around the analyzed pixel. Mathematically Gabor filter's impulse response is a sinusoidal wave multiplied by a Gaussian function. For the purpose of image analysis, we may design a bank of Gabor filters, each tuned to detecting a particular spatial frequency in a specified image direction.

First, Gabor filter is defined for grayscale images, and therefore we must prepare for it a

grayscale version of our image.

```
imGray = rgb2gray(imRGB);
```

Then, we have to make a bank of Gabor filters. In the example, the following values of spatial frequencies and filter orientations are used:

```
waveStep = 1;
orientStep = 45;
wavelength = 2.^(0:waveStep:5) * 3;
orientation = 0:orientStep:135;
g = gabor(wavelength,orientation);
```

Finally, we filter the image using all filters available in the filter bank

```
gaborResults = imgaborfilt(imGray,g);
```

It is useful to remove noise before further processing

```
for i = 1:length(g)
     sigma = 0.5*g(i).Wavelength;
gaborResults(:,:,i) = imgaussfilt(gaborResults(:,:,i),3*sigma);
end
```

In our example, 24 new image have been created. Each represents the response to a given Gabor filter instance. All of them are used in the next step to enhance the feature space. After changing the feature space, perform clustering once again.

```
features = cat(3,im_channels,X,Y,gaborResults);
[labels, cmap] = imsegkmeans(features,numClusters);
im_label = label2rgb(labels);
```

You can also use only Gabor filter responses as elements of feature space. Try to test different possibilities. The choice of feature space elements depends on the image you want to analyze and the types of object you want to segment out of the background. As in the case of previous segmentation algorithms, the results of the segmentation may be processed in order to find geometric features of objects as you will see in the following parts of the lab.

Task 2.6 Perform K-Means image segmentation of image with a texture provided by an instructor. After filtering images by Gabor filters, visualize the results for all filters. Try to chose the filters that provided the best response that can be used in the image segmentation. Try to discard the filters that gave redundant information. This should reduce the feature space and computational time.

Additional tasks

Task 2.7. Write a program that will enable segmentation of a color image and carry out a segmentation of an image given by the instructor, not using K-Means method.
Remember that in addition to the standard color representation, i.e. RGB, MATLAB offers other ones that may be better suited for efficient segmentation (of course, it depends on the

image type). The choices are, for example, the HSV or CIELab color space. Before you start writing code, find information on how color is represented in these spaces.

Task 2.8 Knowing how morphological filter work, write a script that detects the edges of binary objects using only morphological operations and simple logical operation. This approach is called a morphological gradient.

Task 2.9 Try to propose some other features which could be helpful in the K-Means image segmentation and add them to the feature space. Use the above approach to find the best segmentation for a few images provided by the instructor. For each image, the complexity me be different. Try to find the best number of clusters and the best set of features that gives you the best segmentation result for the given image.