L3. Binary image processing. Pattern recognition basics

A. Binary image processing

A1. Image labeling

Labeling process is performed on binary images and assigns a unique label to each individual object, by exploring the neighborhood relations between object pixels. Each pixel in the output image will have a value that corresponds to the object to which it belongs. MATLAB dispose of the predefined ***bwlabel*** function that performs the image labeling. An example of ***bwlabel*** function call is:

```
[L N] = bwlabel (IO, 8);
```

The first parameter corresponds to the binary image, and the second one to the neighborhood type (4 – for 4 neighborhood and 8 – for 8 neighborhood).

The output **L** is the image containing the labeled objects, along with the number of distinct objects **N**.

Display the labeling results:
Image **L** contains unique labels for each detected object. Though, the display of the result by just calling the ***imshow*** function is not relevant, since visually the labels are not properly differentiate. For a better visualization, MATLAB dispose of the ***label2rgb*** function that emphasize the color of each label:

```
IRGB = label2rgb (L, @jet, 'k', 'shuffle');
imshow (IRGB);
```

The first parameter is the labeled image **L**, the second one is a predefined color palette '**@jet'**, the third one is the background color '**k**' (- black), and the last one '**shuffle**' indicates the random choice of colors. The output of this call is a color image contains objects with different random colors.

A2. Morphologic Operations

The basic morphological operation are erosion and dilation. Both operations have as parameters the original image and a structuring element. The structuring element indicate the neighbors that will be duplicated or eliminated after applying morphological operations.

Dilation has an effect of object thickening".

```
ID = imdilate (IB, S);
```

S is the structuring element. It is a binary matrix that can be defined by the user, or they can be generated by calling ***strel*** function and create structuring elements with predefined shapes such as: *disk*, *square*, *rectangle*, *diamond* etc.

```
S = strel ('disk', 5);
```

The call of this particular function will generate a circular structuring element with radius 5.

Erosion has the opposite effect of dilation, reducing the dimension of the objects.

```
IE = imerode (IB, S);
```

Based on the basic morphological operations, other derivate operations can be defined, as for example opening and closing. Opening operation is defined as a dilation followed by an erosion. Its role is to eliminate small objects (or noise) remained as a result of thresholding operation. The closing, on the other hand, has the role of eliminating small holes in the objects. Both operations will keep the real dimensions of the objects.

```
IO = imopen (IB, S);
IC = imclose (IB, S);
```

A3. Simple geometrical properties of binary objects

Having as input a labeled image, it can be generated a binary image having non-zero pixels only in the position corresponding to an object. For example, we can create the binary image corresponding to the object having the label 1:

```
O1 = (L==1);
```

In this image, we can extract the pixel coordinates contained in the current object using the function ***find***.

```
[row column] = find (O1);
```

The mass center of object one can be computed as the mean value of the row and column coordinates.

```
r_centermass = mean (row);
c_centermass = mean (column);
```

The min and max coordinates of the object can be found as follows:

```
r_minim = min (row);
r_maxim = max (row);
c_minim = min (column);
c_maxim = max (column);
```

The object area, or the number of object pixels is directly computed as:

```
area = sum (O1(:));
```

A4: Practical work

Open the '*eight.bmp'* image. Apply all the necessary processing steps in order to compute the geometrical properties (area, center of mass, min/max coordinates) of each object. Print the results in the command line window.

Hint: the steps should be: thresholding (see L2), negative (I = ~I), labeling, morphological filtering (fill in small holes in the objects), geometrical properties computation, rint the values to the cmd line.

B. Basic pattern recognition with Matlab

B1. Preprocessing: thresholding and filtering

After the image is read from the disk and transformed in grayscale (see lab 2), the image is converted to grayscale and binarized using a threshold that favoures the best separation between the objects and the background. For the image bellow you can use a threshold of 250.

Fig. 1. Input image

```
Ig = rgb2gray (I);
Ibw=im2bw(Ig, 250/255); % threshold value should be between 0 .. 1
```

Fig. 2. Thresholded image '*cercuri-stele.jpg*' with th = 250

In Matlab, object pixels are considered the white ones so you nedd to invert the colors of th e image:

```
Ibw=~Ibw; % image negative
figure(1); imshow(Ibw); % show the result
```

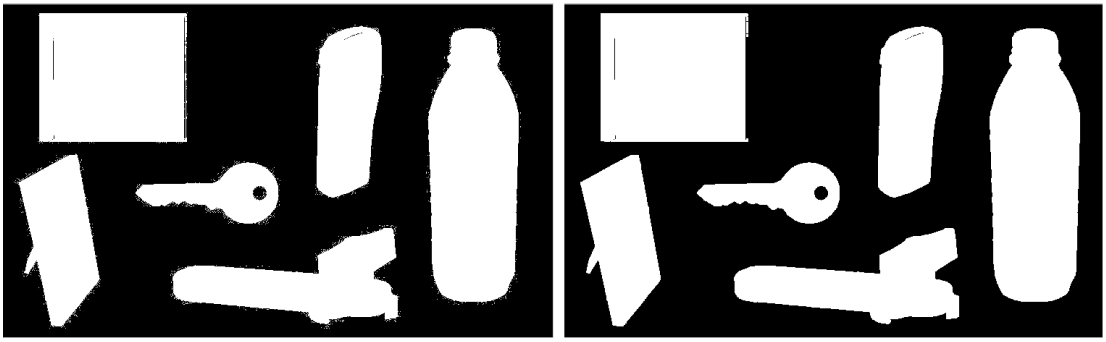
Fig. 3. Negative image from figure 2

In other situations you can use the automatic threhsold computation algorithm presented in the lecture (see *hist_threshold.m* in *ex_matlab_curs.zip*).

Depending on the thresholding algorithms' result, a morphological filtering operation might be necessary in order to eliminate imperfections and identify properly the objects.

If there are points (noise) around the objects an opening operation might be required.

```
se = strel('square',3); % create a structurring element of size 3x3
IC = imopen (Ibw, se);
```

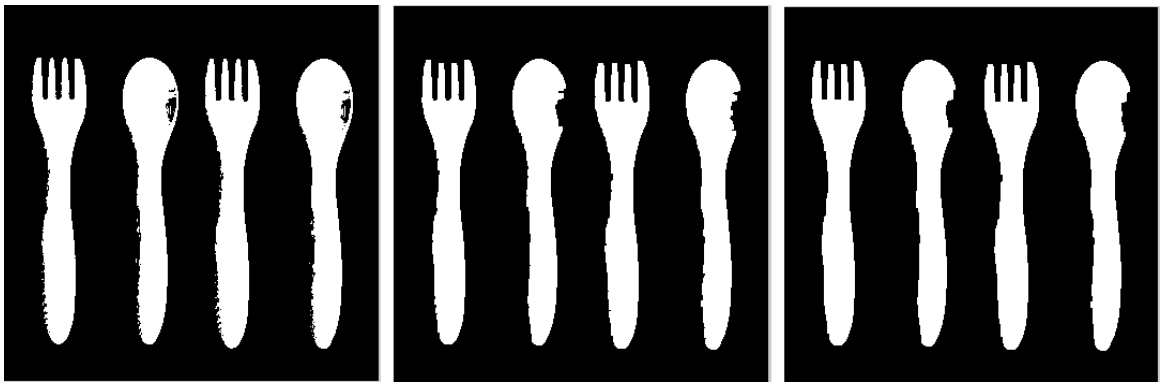


(a) (b)

Fig. 4. (a) Thresholding (*th* = 200) + negative of image '*obiecte-diferite.jpg*'; (b) result after opening on Fig.4.a.

If there are too many holes in the objects, a closing operation is required:

```
se = strel('square',5); % create a structuring element of size 5x5
IC = imopen (Ibw, se);
IC = imclose (IC, se);
```



(a) (b) (c)

Fig. 5. (a) Thresholding (*th* = 230) + negative of image ' *furculita-lingura.jpg*; (b) result after opening on Fig. 5.a; (c) Result after closing on Fig. 5.b.

B2. Labeling

Labeling example (including display code):

```
[L N] = bwlabel (IC, 8);
IRGB1 = label2rgb (L, @jet, 'k', 'shuffle');
imshow (IRGB1);
```

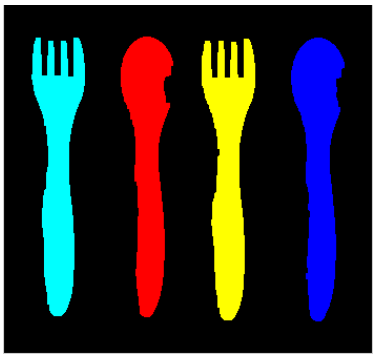


Fig. 6. Result of the labeling operation on the filtered image (figure 5. - Right)

B3. Geometrical properties

In order to recognize objects, relevant geometric properties are computed to discriminate between several types of objects. In our case we will compute the area, the perimeter and the eccentricity:

- the *area* is the total number of object's pixels;
- the *perimeter* is the total number of pixels from the object's frontie;
- the *eccentricity* is the deviation of the object's envelope relative to a circular one (i.e. 0 – circle; 1 – line).

```
objprop = regionprops(L, 'Area', 'Perimeter', 'Eccentricity');
```

B4. Usage of the geometrical properties for classification

One element that can differentiate between objects types is the ratio between the area and the perimeter.

$$T = 4\pi\left(\frac{A}{P^2}\right)$$

As the borders of the objects are rougher and its shape is more concave, the ratio is smaller. The maximum value of T is 1 (circle)

A list containing these ratios (for all the detected objects) is created:

```
T = [];
for i=1:N
    T = [T 4*3.14*objprop(i).Area/(objprop(i).Perimeter*
                                    objprop(i).Perimeter)];
end
```

Example:
Perform the above mentioned operations (*th* = 250, opening with *se* of size 3 and closing with *se* of size 5) on image '*cercuri-stele.jpg*' to obtain a result similar to the one in figure 6:

(a) (b) (c)

Fig. 7. (a) Result after opening on figure 3 with *se* of size 3; (b) Result after closing on Fig. 6.a image with *se* of size 9; (c) Result after labeling on Fig. 6.b.

List T will have the following contents (for the above mentioned settings):

```
T = [  0.85514   0.32029   0.88804]
```

As it can be seen, the star-shape object has a ratio much smaller than the other objects. That way we can automatically differentiate the star shape from the others.

To differentiate between the ‘circle’ and the ‘shield’ objects we can used the eccentricity property (modify the above code to display the list of eccentricity attributes of the objects). You should obtain something similar to this:

```
E = [    0.0842    0.1137    0.5355]
```

For the circle we obtain the smallest eccentricity. You can also observe that the star has a low eccentricity due to the fact that its envelope is closed to a circular shape).

Based on the 2 properties we can a simple object classifier (similar to a simple ‘decision tree’), valid for object segmentation from example / figure 7:

```
CL = [];     %class list , initially empty

for i=1:N
    if T(i) < 0.5 %star
        CL = [CL 2];
        fprintf('Object %d: star \n', i);
     else
        if objprop(i).Eccentricity > 0.3
            CL = [CL 3]; % shield
            fprintf('Object %d: shield \n', i);
        else
            CL = [CL 1]; %circle
            fprintf('Object %d: circle \n', i);
        end
    end
end

CL
```

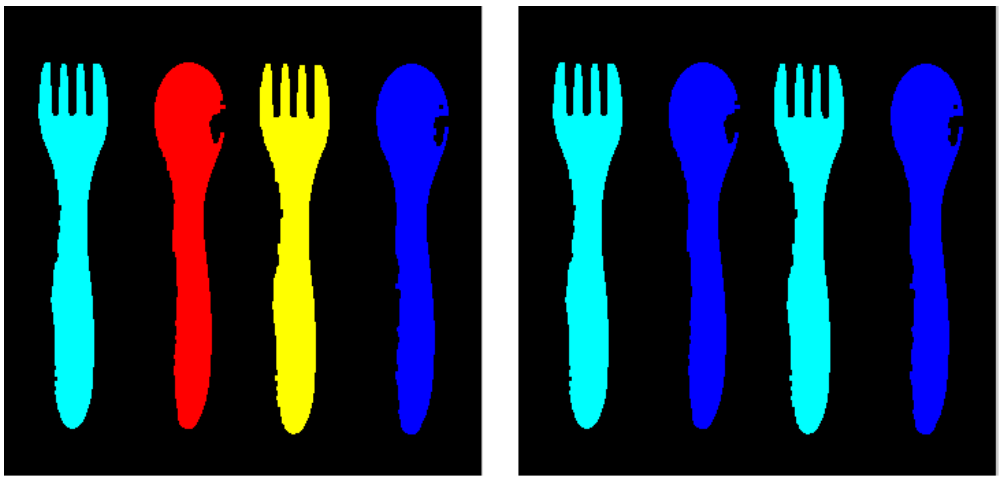
And the obtained results should look like:

```
Object 1: circle
Object 2: star
Object 3: shield

CL = [  1     2     3 ]
```

B5. Graphical display of the segmentation results

In order to illustrate the classification we can use a displaying method similar to the one used for labelling. Its advantage will be more obvious when there are more objects of the same type/class (fig. 8), where 4 objects must be classified in 2 classes: ‘fork’ and ‘spoon’. The left image is the result of the labelling, while the image on the right is the result of the classification.



(a) (b)

Fig. 7. (a) Image after labelling; (b) Object classes highlighted in different colors.

In the first step we need to create a class image (LC) of the same size with the labels image/matrix initialized with null values.

```
LC = L * 0;
```

Then, for each object, we will add to the class image its corresponding points/pixels with the associated class value:

```
for i=1:N
    LC = LC +(CL(i) * (L==i));
end
```

The result of the classification can be displayed like this:

```
IRGB2 = label2rgb (LC, @jet, 'k', 'shuffle');
figure, imshow (IRGB2);
```

And the obtained results should look like:

```
T = [0.16960   0.33714   0.16722   0.33733]
Object 1: fork
Object 2: spoon
Object 3: fork
Object 4: spoon
CL = [ 1     2     1     2 ]
```

B6. Practical work

1. Implement the presented steps as a single program. Display all the intermediate results (rgb2gray, thresholding, labelling, geometric properties, classes). The first test image will be *'cercuri-stele.jpg'*.
2. Write another program able to classify the objects from image *'furculita-lingura.jpg'*. Hint – adjust the threshold, size of the morphological filters and the thresholds and rules of the classifier.
3. Write another program able to classify the objects from image *'cleste-surubelnita.jpg'*. Hint – adjust the threshold, size of the morphological filters and the thresholds and rules of the classifier.

4. Write another program able to identify the key and the CD case from the image '*obiecte-diferite.jpg'*. *Hint:* use E o T to distinguish between the CD case and other objects and the 'EulerNumber' (EulerNumeber = 1-nr_holes) property to distinguish between the key and other objects (if the segmentation is properly performed the key should be the only object with a hole.

Note: All the test images can be found in the archive 'obiecte.zip'. Implement each classifier in separate m files.