Report for Lab session in

IMAGE SEGMENTATION

Daria Kruzhinskaia
email: dariakrukru@gmail.com

Prepared for: Advanced Color Image Processing
Course: COSI
Course instructor: Nuria Ortigosa

AIM

In this lab session we had to complete different activities to achieve the best performance for image segmentation. We thresholded and segmented different images (mostly gray scale) and we achieved the best result as possible trying not lose much information about the captured objects because in the real word it is critical to made an accurate analysis of them for majority of the image processing applications, especially in medical field.

LAB SESSION 1

Task: obtaining the suitable thresholds and the binarizing following images ('BB.tif', 'circuit.bmp', 'tools.bmp', 'chromoso.bmp').

In other words, for this lab session our goal is to binarize the images, trying to lose as less information as possible. This mean that objects should not significantly change their shape, they should be segmented from background and from each other. To achieve this we applied different methods for each image because they have their own characteristics that require different strategies.

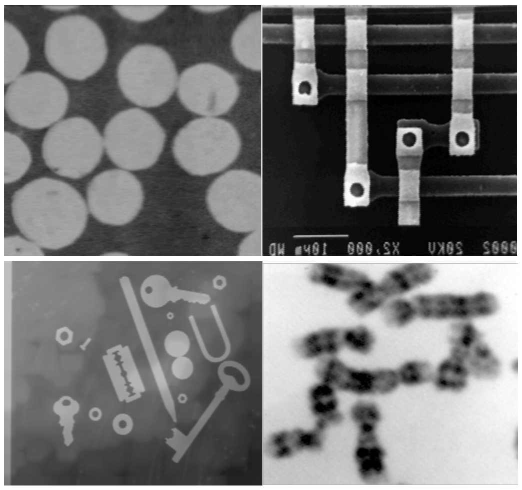


Figure 1. Original mages

BB.tif picture binarization

For the BB image, which contains several circles, the main problem was to split circles from each other, because they are very close and their borders touch. Firstly, Otsu thresholding was used to obtain initial black and white image. After that few circles were merge in one white component.

One approach to divide them is to use morphological operations (opening is our case) with quite big structuring element. But this approach transforms the shape a lot (Figure 2b). Therefore we applied the built-in function called *watershed()* that allows to detect the different circular components of the original image.

Function *watershed()* can be used as a complementary approach to segmenting objects that are in contact with each other. It finds "watershed ridge lines" in an image by treating it as a topological surface where light pixels represent high elevations and dark pixels represent low elevations. These rigid lines are used to separate the “catchment basins” following the logic that if there were water on the rigid line, it has equal chances of falling in either catchment basin. The catchment basins are local minima for the algorithm since they are low elevation points, and the region around these local minima is grown as if it was flooded with water. At the point where the regions of the basins merge, the algorithm constructs a dam that separates them.

The work of the algorithm is about why the transformation is applied not to the black & white image, but to negative result of *bwdist()* applied to inverted binarized image (Figure 2d). This image is complementary (background is white, circles are dark) and has only two values (0 or 1) to represent “landscape” or “high map” and watershed could be used. After the transform, the borders between objects have zero value. By adding these black borders to the initial binarized image we get each circle as a separate component, which could be analyzed further.

After that we subtract the result of the watershed function from the original image to have an image with separate components. We can see in the image from Figure 2f that the result of this subtraction is good enough but we decide to make an opening to have a better result, for this erosion we used and morphological element with disk shape (which is the most suitable for round objects) of 8 pixels size. Final result you could see at Figure 2c, highlighted with frame.

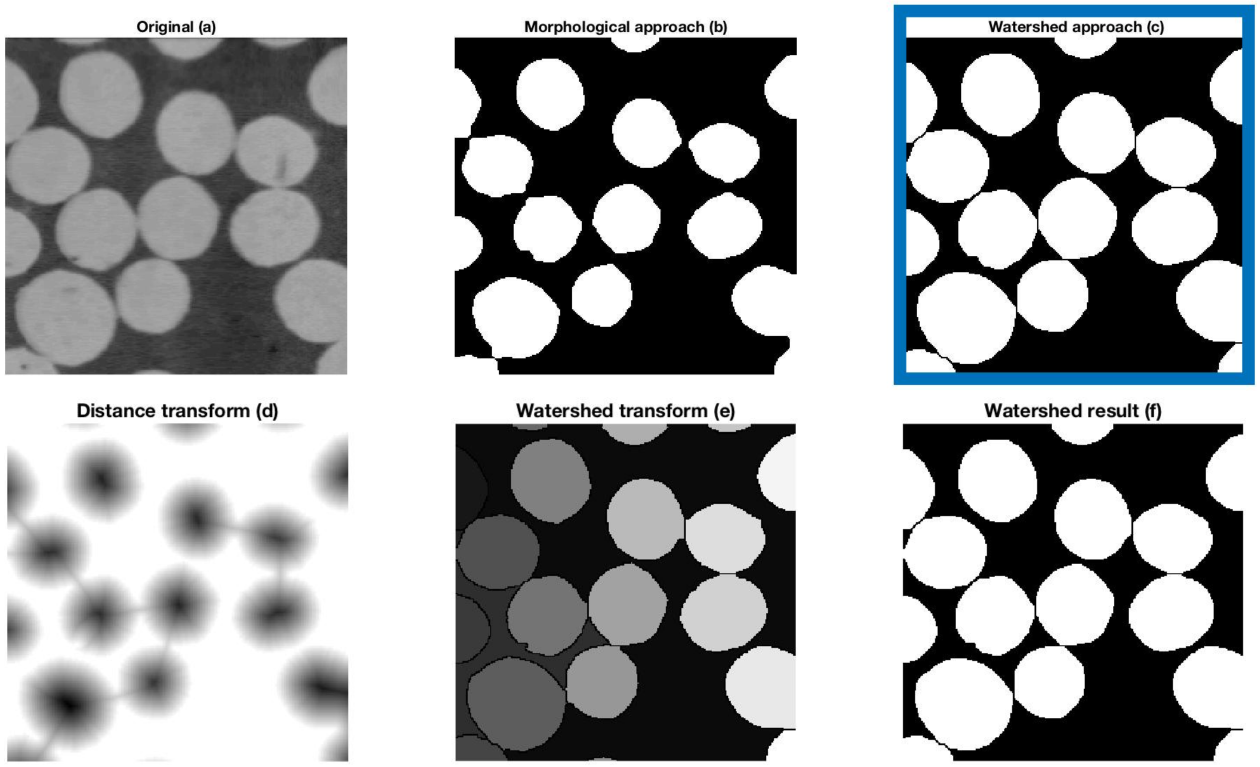


Figure 2: a) original image; b) morphological approach; c) final result with watershed approach; d) result of bwdist() function; e) result of watershed transform; f) image after watershed substracton.

It seems that the using of the watershed function could be really useful to segment images that have well defined geometrical components like in this case.

Circuit.bpm picture binarization

To binarize and segment the tools.tif image we applied several different methods, but the most sufficient result of thresholding was obtained by custom threshold (which is Otsu thresh by *graythresh()* function minus 0.09) after set of trials. Using this number almost all area of background becomes black and at the same time all main elements still safe.

Anyway there still a lot of unneeded horizontal lines and the simplest way to get rid of them is to apply erosion with vertical line structure element (line of length 6 px already gives a proper result).

Because thin horizontal lines do not contain enough pixels in vertical dimension – they just disappear after erosion. Finally, to delete small white dots function bwareaopen() was used. Final result is shown at Figure 3.

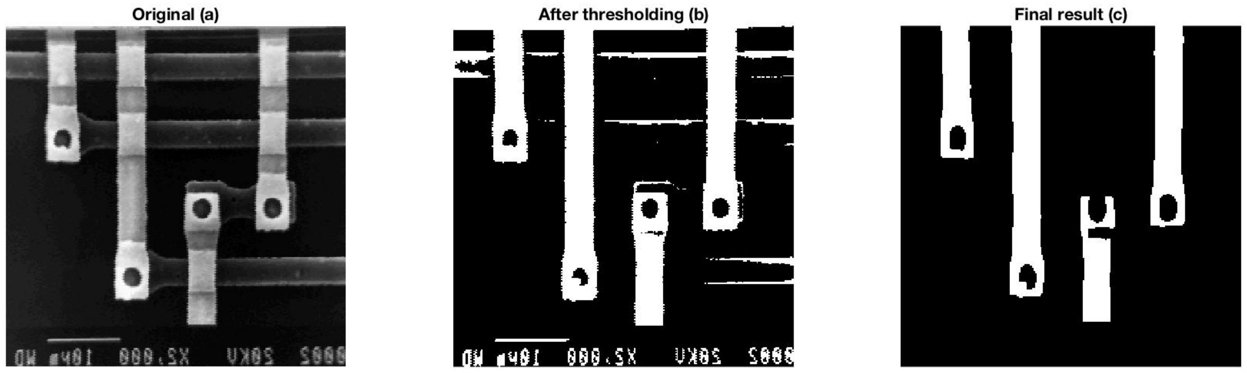


Figure 3: a) original image; b) binarization using custom threshold; c) after morphological operations.

Using this method we were able to preserve the holes in the middle of the components of the circuit and also it is the best way to obtain a better shape of the only element that goes from bottom to top.

Tools.bpm picture binarization

For binarization of the image tools.tif we had to apply a filter that can homogenize the illumination of the scene because the left top corner of the image has a different illumination than the rest of the image. After applying any direct binarization methods, like *imbinarize(),* to this image results are not satisfactory because the threshold should be different for the brighter and darker parts. There are two approaches to eliminate this problem.

First one – is to use array of thresholds instead of one. Matlab built-in function *adaptthresh()* do exactly this: calculate own threshold value for each pixel. But after applying such thresholding (Figure 4b) we got a lot of noise and white spots in the background. Some of them are bigger that small tools that we need to segment.

Second way to fix the illumination of the image is implementation of top-hat filtering. We performed operation *imtophat()* on grayscale image and obtained a mixture between top-hat and original one (to make transformed image more intense). After this we applied binarizing direct method. The Figure 4d correspond to a binarization after build a second images that is composed by the linear combination of original image and the top-hat filtered image (Figure 4c), the equation for this image is the next: *Original-image.*0.1 + Top Hat-image.*0.9*.

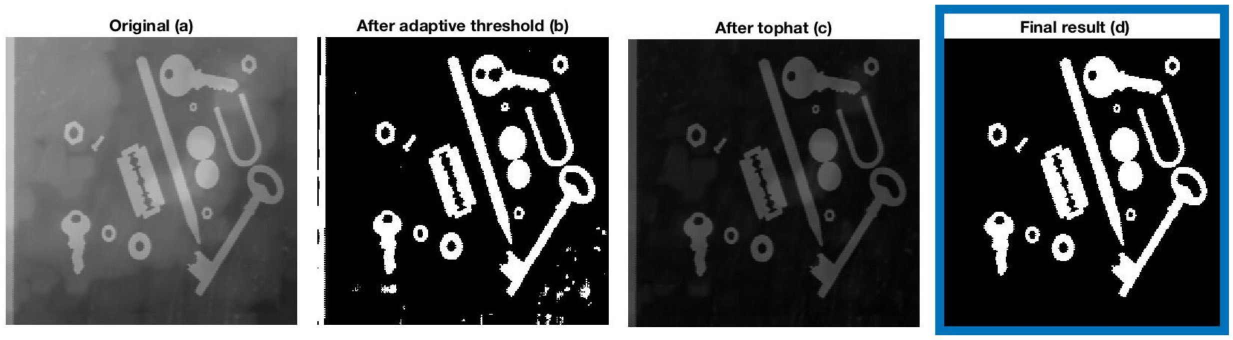
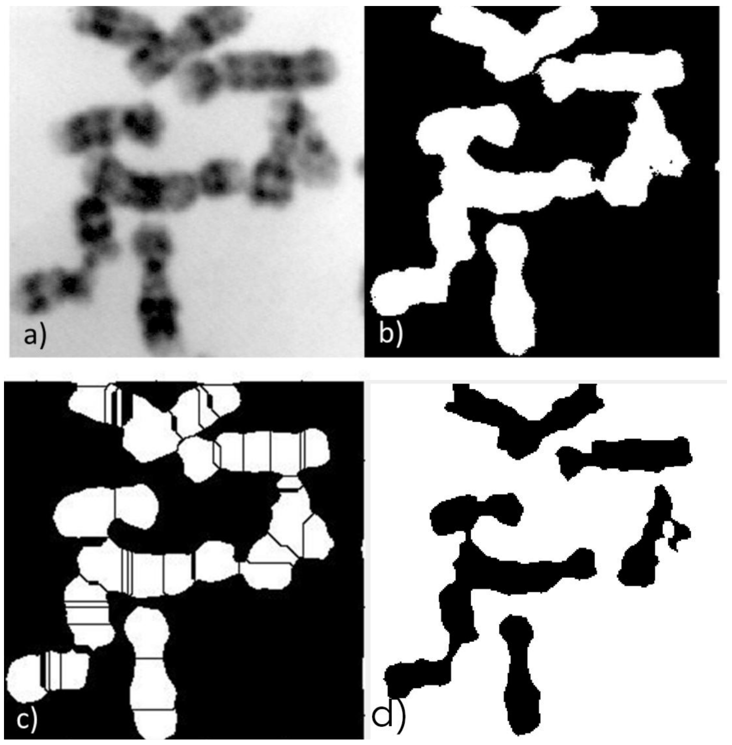


Figure 4: a) original image; b) result with adaptive threshold; c) top-hat filter; d) result of binarization after top-hat.

Chromoso.bmp picture binarization

The Chromosomes images represent a real challenge to binarize because not all the parts of the chromosome are well defined in the image and also these cells have a certain shape that we want to preserve because for some medical application is really important to preserve the shape of the objects because some of the diagnosis are bases in shape and size of a cell, calcification or tumor.



For this case we tried do to apply 3 different methods. In the first and third methods we applied graythresh and imbinarize build functions but in for the first case we have to get the complementary image of the resulting image because as we can observe in figure 5 a) and d) the binarization of the chromoso image give as a result an image where the chromosomes are black and the background is white, we know that Matlab consider only white parts of a binary image as objects so this is why we decided to take the complementary image of our resulting image to get b).

Figure 5: a) original image, b) binarization using imbinarize with global adjustment using the complimentary image, c) binarization using watershed, d) binarization using imbinarize and morphological operations.

To obtain the b) image we used the global method of the imbinarize image. To get image d) we used just graythresh and imbinarize using only the threshold generated by graythresh and than apply morphology.

For figure 6c we used watershed() and bwareaopen() inbuilt functions, the first one to detect the shape of the chromosomes and the second one to erode some of the elements (small ones) of the images to separate some of the cells in the picture, to get a better result of the process we applied an average filter and get the complementary image of the binarization of the original image.

In this case we determined that the method used to get the images display in Figure 5c is the best to get a good shape of the chromosomes and to separate them without losing a lot of information. The method presented to get Figure 5b is also good but we were not able to perceive some of the inner lines of the cell and the method using watershed allow us to do this.

Conclusion of Lab session 1

The common geometrical forms are more easier to recognise. Methods where we used watershed function were the best in binarization and segmentation of these images without compromising a lot of information contained in the image during the process. Also it is quite challenging to segment contacting objects.

LAB SESSION 2

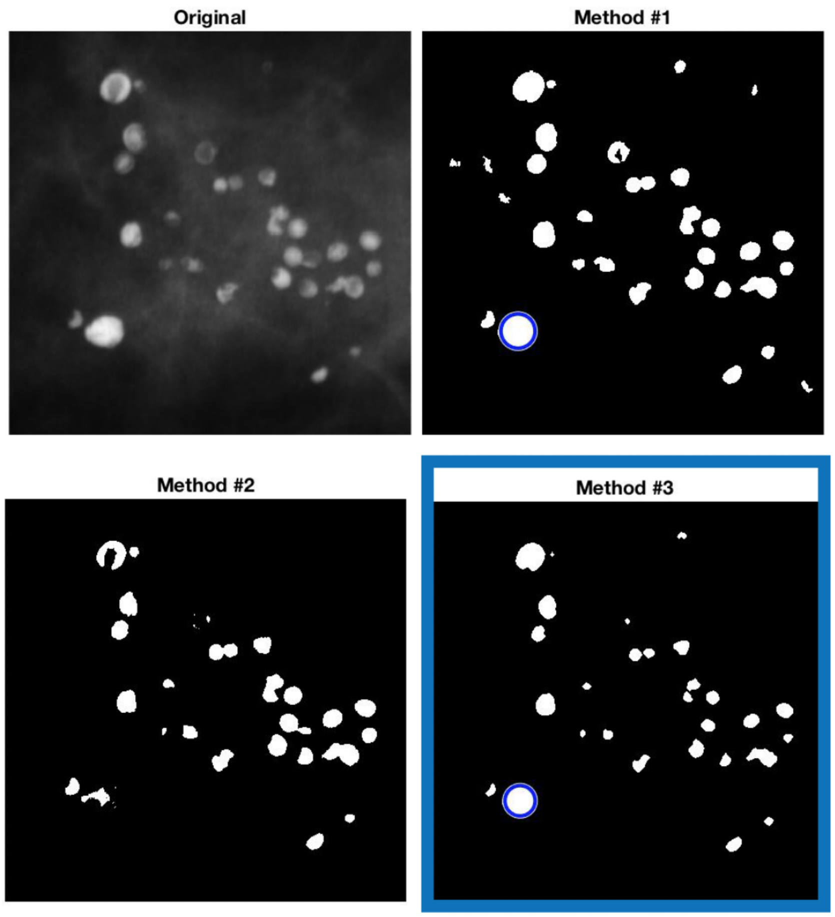
EXERCISE 1

The image *bencal1.tif (*Figure 6a) is a mammography of a woman with benign microcalcifications. In this lab session we were ask to calculate the number and the average size of the calcifications founded in a breast cancer mammography. To achieve this aim we follow the next list of steps:

1. Binarize the image
2. Segmentation of each microcalcification
3. Get the properties of each element of the image
4. Found the correspondent properties of the biggest element
5. Highlight the biggest element

To binarize and segment the image we tried three different methods (presented at Figure 6).

For the first method (Figure 6b) of binarization we applied Otsu threshold and after *bwareaopen()* function to erode the elements that are not relevant for the analysis, in this case — smaller than 50 pixels.



For the second method (Figure 6c) firstly a top-hat filter was applied to homogenize the illumination and function *imadjust()* was used to enhance the contrast of the image. By function *graythresh()* we obtained threshold by Otsu method of enchances pictute for following binarization.

Third method (Figure 6d) consist of binarization with adaptive threshold (explained earlier), dividing close, even merging calcifications by *watershed()* (also explained in Lab 1, part 1) and couple of morfological tramsformations. Structuring element for such manipulation has a disk shape that is reasonable for roundish microcalcifications.

Figure 6. Original and segmented images of calcifications.

Form visual assesment of this image, we can conclude that the method #1 is better to get the calcification without losing important information about their shape and size. However, by this approach a lot of noise and unexpected white spots appear and can not be deleted by morphological operations without elimination of actual objects. Moreover, some calcifications are merged. Therefore method #3 give more precise characterization of patology and it has more sence to use this approach in practical applications.

After applying the *regionprops()* function we found average characteristics of calcifications. They are shown below in Table 1.

Method	Amount of calcifications	Average size (pixels)
#1	27	352.6
#3	28	210.4

Table 1. Average characteristics of calcifications.

We find the biggest element by finding the maximum of the array corresponding to the Area property. Below Table 2 presents the characteristics of the biggest element.

Method	Centroid	Area (pixels)	Perimeter (pixels)	Radius (pixels)
#1	[107.7, 333.4]	1107	117.6	18.7
#3	[107.8, 333.3]	893	106.2	19

Table 2. Characteristics of the biggest calcification detected.

In Figure 6 the biggest element is highlighted using a blue circle drawn by build-in function *viscircles(center, radius)* where input parameters were obtained by *regionprops()* function (center is two coordinates of centroid and radius can be calculated using perimeter or area – we showed both methods).

In this case we can observe that the top-hat filter do not work as well as in the cases of Lab session 1 and method using it gave us quite messy result.

From tables and resulting images it is clear that main differences between different approaches are significant in small elements and almost disappear in segmentation of big objects.

EXERCISE 2

The file myocyte.tif is the image of a myocyte during a contractily test. Aim is to estimate the size of the myocyte: height and width.

In this exercise we measured the height and width of myocyte cell (known as muscle cell). To achieve this we applied a simple function *regionprops()* to calculate the *'MajorAxisLength'*, *'MinorAxisLength'* properties to binarized image.

To binarize the image we used 2 different methods, both represent consequence of different morphological operations. The first one consist of the built functions: imfill, bwareaopen, imopen (using a line as structure element). As threshold it get value be Otsu method. In the second method we use custom threshold *graythresh() - 0.0075* to binarize the image and SE of diamond shape.

In the following Figure 8 we can see the results obtained with these methods. We can observe that the results do not have a big difference in shape but we obtain different values for the height and width, these results are presented in the Table 3.

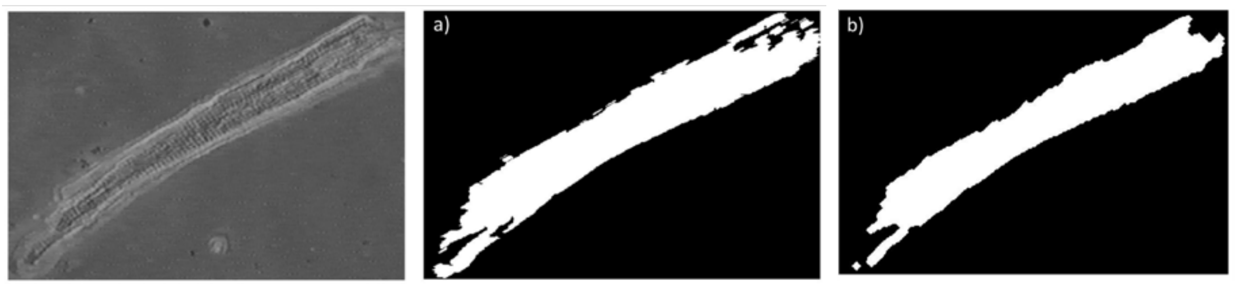


Figure 7. Myocyte binarization: original image; a) method #1; b) method #2

Method	Height	Width
#1	594.9099	84.37
#2	574.8450	76.1668

Table 3. Features of myocyte

Even though, the results are different we can say that 20 pixels of length difference and less than 10 in width is not so bad, so we can claim that both methods work really satisfactory.

Lab session 3

EXERCISE 1

Task in this part of the lab was to obtain the angle of the inclined text (Figure 9a) and rotate the image to have horizontal lines. All lines of text form a tilted rectangle, so our goal was actually to obtain orientation of this text rectangle.

For this binarized and inverted image of the digits was dilated by big structure element (diamond with size of 10 px). Result is illustrated at Figure 9b. Therefore all elements expended and merged in one figure. Using build-in matlab function *regionprops()* that measure properties of image regions we were able to get angle between x-axis and main axis of general figure by orientation property of this function. Then we only had to apply *imrotate()* transform to move initial picture in opposite direction by this angle. For the next exercise we need not only correct oriented original image, but also horizontal black & white image (Figure 9c).

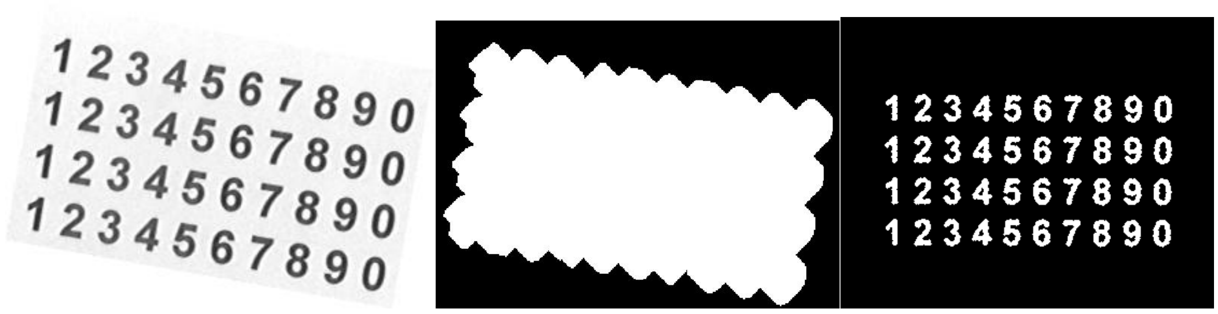
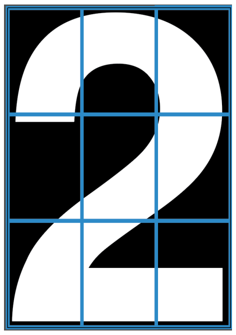


Figure 8. a) Original image; b) merged all digits; c) result of rotation

EXERCISE 2

Based on previously obtained image of digits our goal is to develop a classification model to determine a value of input character. Process could be divided in two main parts: feature extraction and classification itself.

Features extraction



The general idea of this process is to divide area of a character by grid 3x3 (9 cells in total) and calculate the white area in each cell. These nine values became feature vector of the element.

At the beginning areas of the digits were determined by property bounding box of *regionprops()* function. We got these features vectors for 40 characters in the pictures by processing data of each character in two 'for' loops iteratively sum amount of white pixels. In each stage of the loop we restrict area by one cell.

Figure 9. Example of dividing digit area

Classification

First three lines of digits in the provided image were training set for classificator. Using features vectors of the we calculated prototype matrix. How can be seen, in training set each digit repeats three three times. So we computed the average value for every cell for them and got only one value. As output we got matrix with 10 rows (which represent 10 classes or values of different digits) and 9 cols (amount of cells).

Special function *myClassify* was developed to predict to which class belongs character in the input. It calculates a column vector of euclidean distances between features vector of provided character and each features vectors of prototype matrix. Euclidean distances are easily computed by matlab function *dist()*. By finding minimum distance we can determine to each digit our character is more similar and closer. As output the function give number of class that is equal to digit value.

Testing

If testing set in the fourth line of the same image from where we took training set, the results are quite good and **accuracy is 90%**. Classification of almost all the characters give a correct result, except of mistake in character 5 prediction – we got 9 as result somehow. Probably these two symbols have the same amount of pixels in case of 9 cells due to their very similar appearance.

For another image inclined_paragraph_fotocop.tif (in the right, Figure 10a) accuracy is low. After implementation of median filtering to reduce noise, deleting small objects in the corners we got a perfect binarization (Figure 10b). Even in this case classification mostly works incorrect.

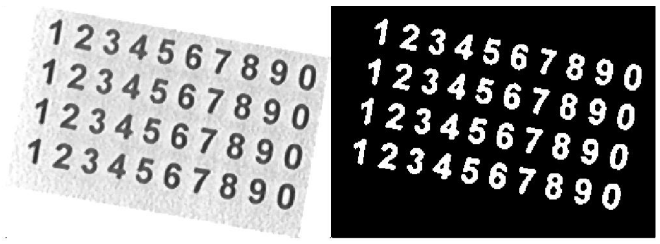


Figure 10: a) original noisy image; b) binarized version

It's due to the fact that after filtering shape changes slightly, but it is enough because size of the image (and so size of every character) is very small. Generally it is better to use high quality image for training.

EXERCISE 3

In this exercise is asked to determine speed limits which are indicated on road signs using classificator developed earlier.

Besides reading the text (appropriate speed limit value) we had to find speed limit sign area. This was done by *imfindcircles()* function applied to binary version of the image where only red colors are presented by white due to subtraction of channels and making red channel dominant. Results are shown below (Figure 11).

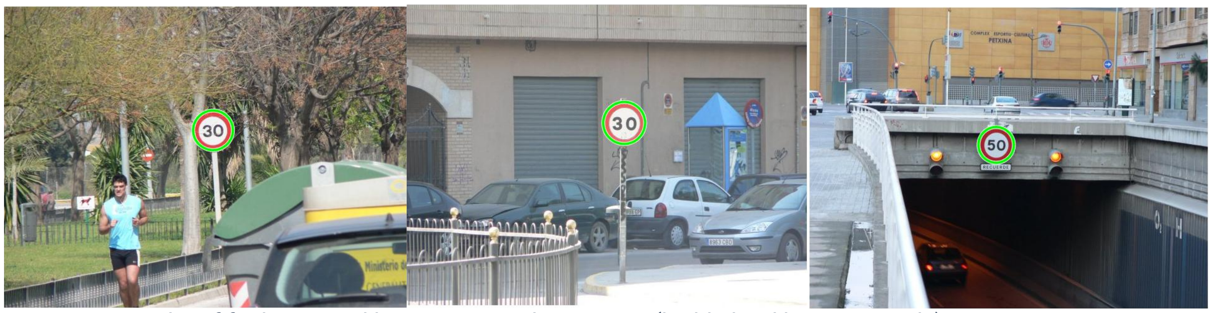


Figure 11. Results of finding speed limits signs in the pictures (highlighted by green circle).

After this step, we binarized the area of sign to determine exact position of two characters by *regionprops()* function the same way we did previously for exercise 2. Only one difference – now we had four objects at each of the corner because they are parts of the outline circle (as at Figure 12). Because elements in regionprops array always have an ascending order according to their x coordinate, this corner object always will be located at two first and two last rows. Therefore we are interested only in 3th and 4th elements.

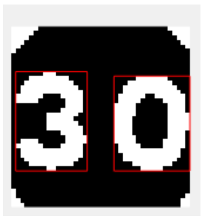


Figure 12. Unneeded objects at the corners.

Extraction of the features and classification from both digits are absolutely the same as described in exercise 2. But results are not so satisfactory:

image 1 - 11; image 2 - 30; image 3 – 11.

So only for second image classification works correct. Accuracy is only 33%. Probably it is due to the fact that for signs and training set different fonts were used. Also, how was said earlier, quality of image of training set is not good enough.

Conclusions of Lab session 3

For good robust classification system of digits or any other character we should improve several factors. Most important parameter is a quality of training set. It include bigger size of the initial image (so bigger size of the character) and increased number of elements. Set should provide a wide variety (characters could vary in appearance, font, etc.). Also it may be a good idea try to divide symbol in more than just 9 cells (to easier differentiate between similar characters). It could solve problems like in our case with “5” interpreted like “9”. But of course for this we need a good enough resolution of elements.