Image Processing with Scilab and Image Processing Design Toolbox

Contents

1 Introduction

There are various applications of image processing, e. g. detection of surface defects in industrial quality control, detection of anatomical landmarks in surgery, counting cells in biotechnology and classification of regions in remote sensing. Images are generated by optical cameras, ultrasound, x-ray machines and other imaging devices.

When processing an image with a computer, it must be digitized or created in a digital format. There are some basic methods to distinguish between objects and background and to describe regions in digital images. This document explains how digital images can be represented mathematically and basic methods for finding objects and describing regions.

Scientists, engineers and medical doctors who intend to analyze images usually know how their images were generated and how the objects of interest look like. However, these objects must be described mathematically when they are to be detected automatically with a computer. The goal of this document is introducing these mathematical descriptions in a way easily to understand.

The remaining part of the document is organized as follows: In chapter 2, a mathematical representation of digital images and characteristic properties of images are introduced. In chapter 3, basic image processing operations such as segmentation into connected regions and filtering are introduced. In chapter 4 methods for object detection are presented. References are given in the last chapter.

Examples are implemented with Scilab 5.3.1 and Image Processing Design Toolbox (IPD) 8.0. The photograph used in this document is taken from IPD. Scilab is available for download at www.scilab.org. IPD can be downloaded from atoms.scilab.org/toolboxes/IPD.

Figure 1.1: This image shows a Japanese tea cup and a can for tea powder. The author took this photograph at home. This image is used in the further chapters.

2 Characteristic Properties of Images

A 2D digital image consists of a finite number of points aligned as rows and columns. A point of a digital image is called “pixel”. “Pixel” means “picture element”. At each pixel there can be a scalar gray value or a vector of color components or gray values. In Scilab [1] there are the following types of images:

- Gray value images, also called “gray scale” images or “intensity images”: A gray value image is a matrix of gray values. In Scilab a gray value image is a 2D array. A gray value is usually an integer scalar between 0 and 255 or a real number between zero and one. An example is displayed in Fig. 2.2 b.
- Pseudo color images, also called “indexed images”: Each pixel of a pseudo color image corresponds to an item in a list of colors. A list of colors is called “color map”. A color map is a matrix. The rows of this matrix correspond to color vectors and the columns correspond to color channels. An example is shown in Fig. 2.2 c.
- Color images: At each pixel of a color image there is a vector of color components, e. g. red – green – blue. Mathematically a color image can be described as a triple of matrices. Each matrix corresponds to a color channel. In Scilab, color images are represented as 3D arrays. The first dimension corresponds to the rows, the second dimension corresponds to the columns and the third dimension corresponds to the color channel. A color component is usually an integer scalar between 0 and 255 or a real number between zero and one. A color image can be transformed to a gray value image by calculating the scalar product or each color vector and a constant vector. An example can be seen in Fig. 1.1 and 2.2 a.
- Logical images, also called “binary images”: A logical image is a matrix of boolean values. When a logical image is visualized, *false* is displayed in black and *true* is displayed in white. Logical images can be generated comparing each pixel of a gray value image to a threshold. Pixels with gray values at least as high as the threshold are mapped to *true* whereas pixels with gray values lower than the threshold get mapped to *false*.
- Object images, also called “label images”: An object image consists of objects and background. At each object image there is a number greater than zero. Pixels with the same number belong to the same object. Pixels with different numbers belong to different objects. Background pixels have the value zero. An object image can be created searching connected areas in logical images. Object images can be visualized as pseudo color images.

Fig. 2.1 shows how a color image, a gray value image and a pseudo color image can be generated and displayed in Scilab. The color image is read from disk and converted to a gray level image. The color image is displayed. The gray level images is displayed as both gray level image and color image. Fig. 2.2 shows the resulting images.

An important property of a gray level image is its gray level histogram. Fig. 2.3 shows how the histogram of a gray level image is calculated and displayed in Scilab. Fig. 2.4 shows the histogram of the gray level image displayed in Fig. 2.2 b. The whole image is very bright so there are a lot of pixels with gray levels >= 200. There are few pixels with a gray value lower than 100. The two peaks in the range between 0 and 255 correspond to the white tea cup segments and the background. The low peaks between 100 and 150 correspond to the comparatively dark blue and red areas.

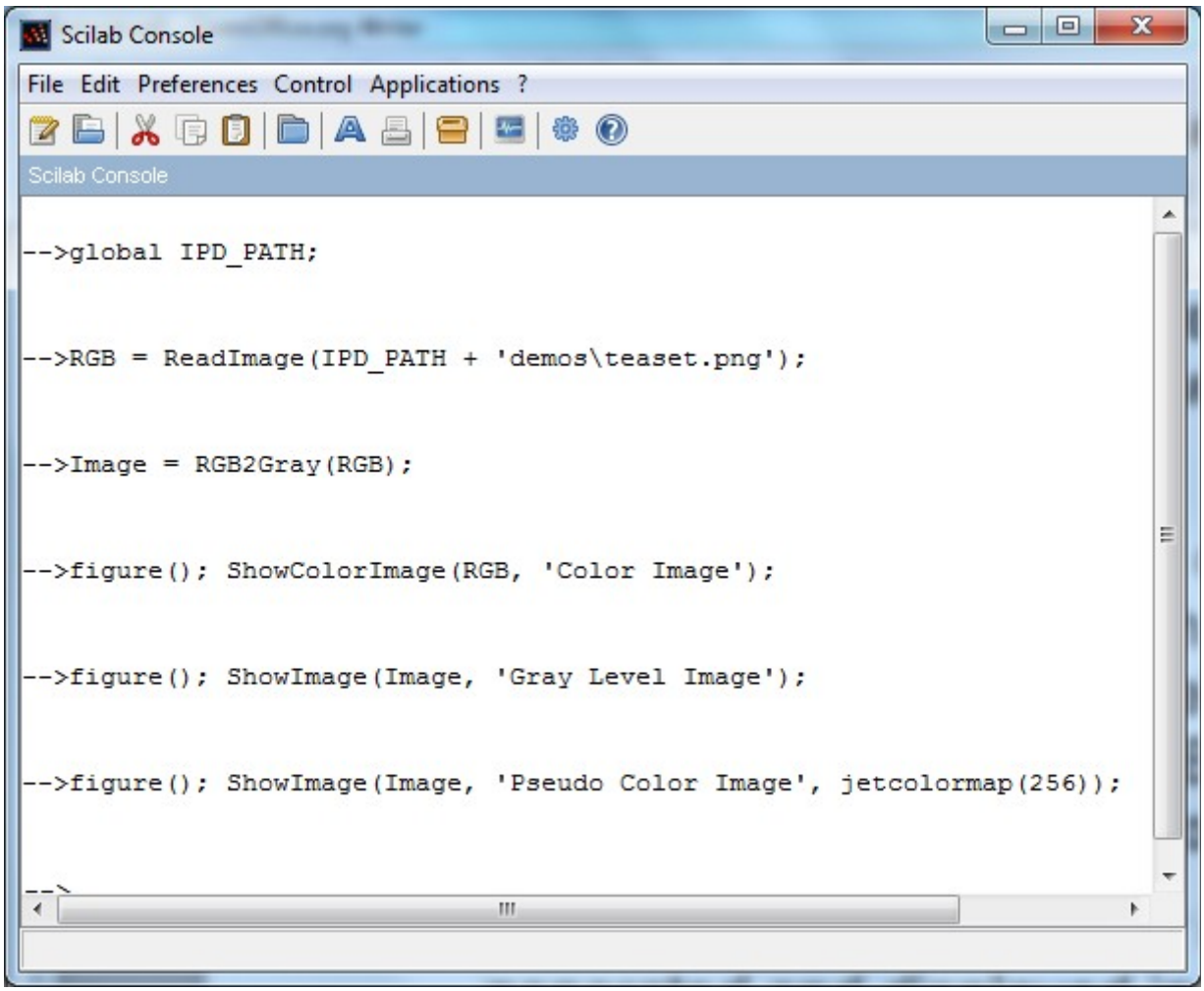


Figure 2.1: A color image is read from disk and converted to a gray level image. The color image is displayed. The gray level images is displayed as both gray level image and color image.

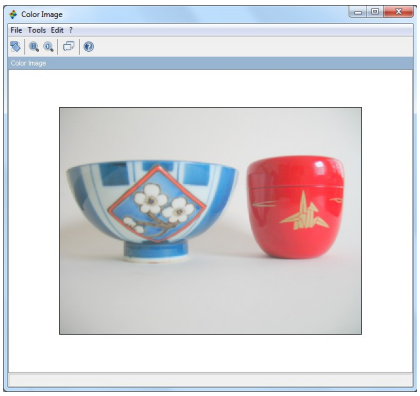


Fig. 2.2 a: A color image saved on hard disk.

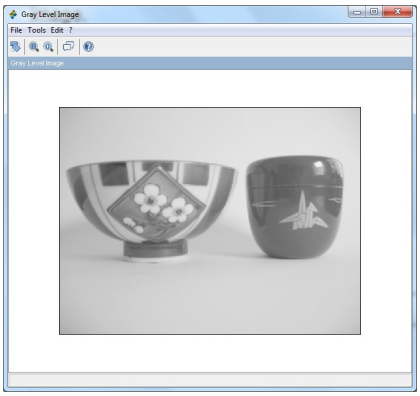


Fig. 2.2 b: Result of transforming color image to gray level image

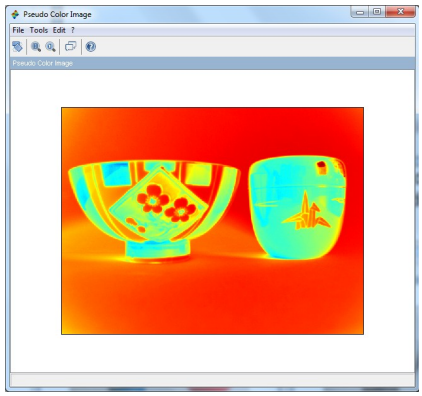


Fig. 2.2 c: A gray level image displayed as a pseudo color image.

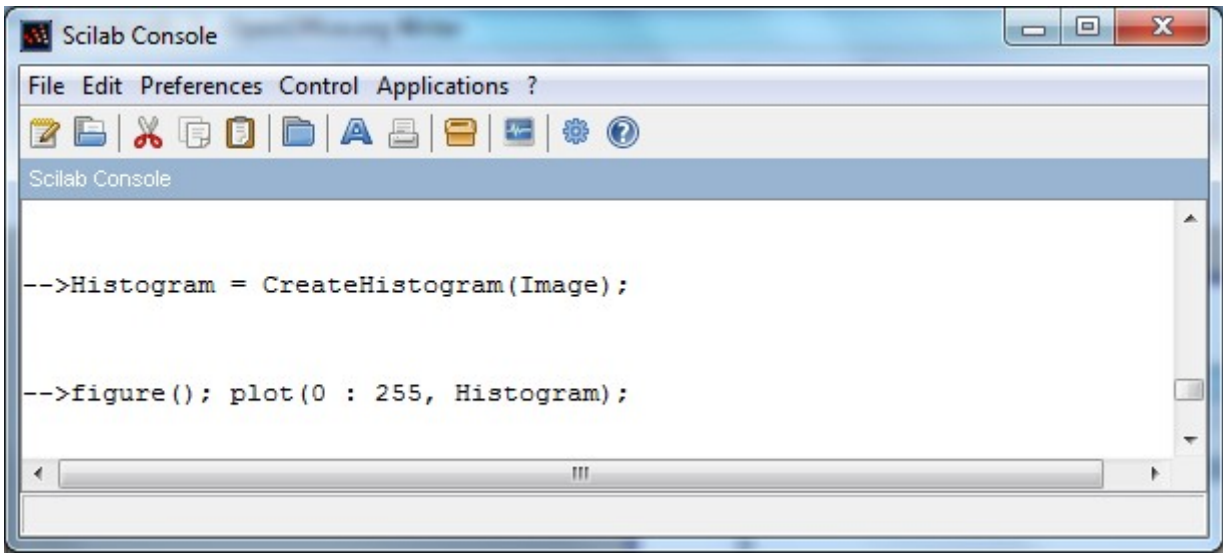


Fig. 2.3: How to calculate and display a histogram in Scilab

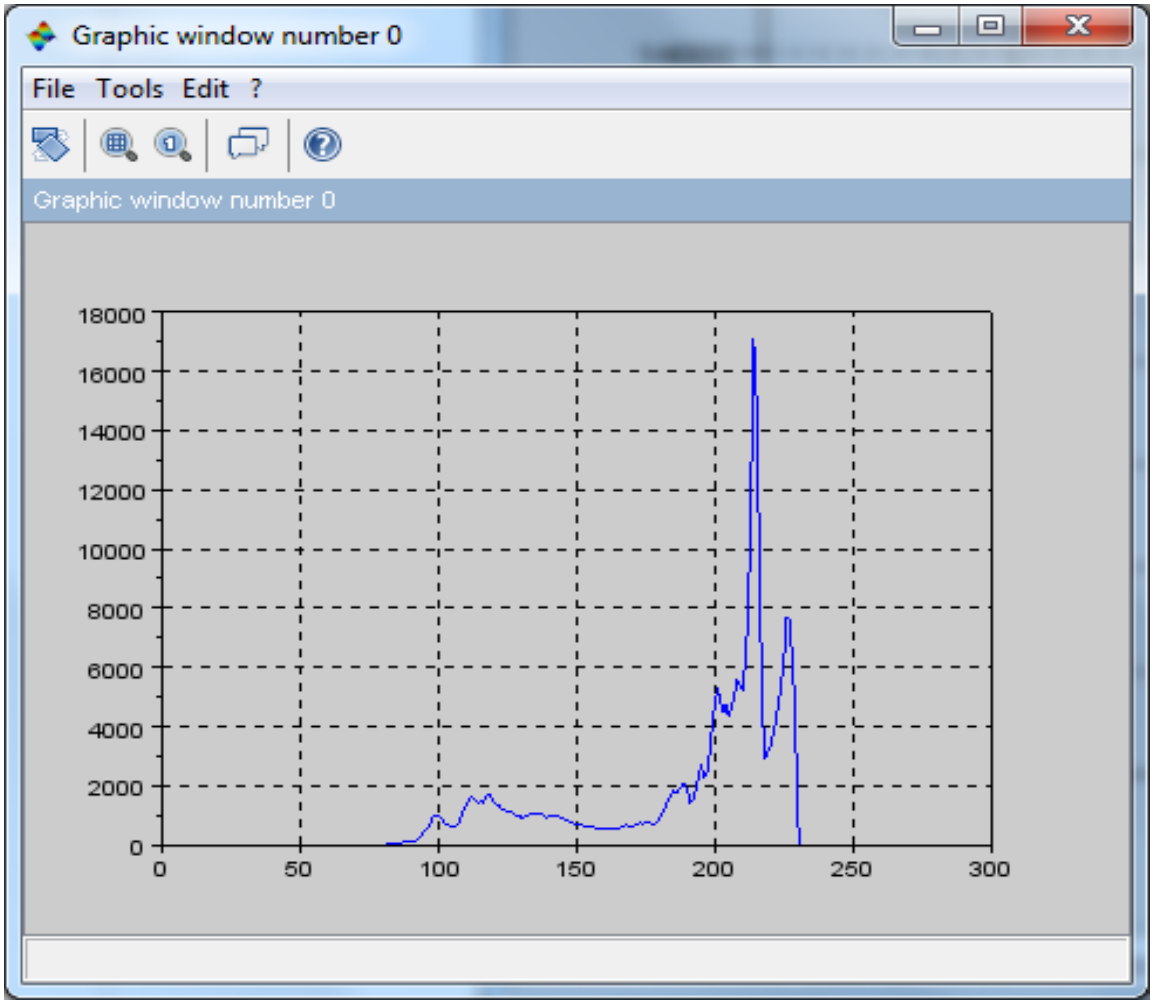


Fig. 2.4: The histogram of the image shown in Fig. 2.2 b. The two high peaks between 200 and 255 correspond to the white tea cup segments and the background. The three low peaks between 100 and 150 correspond to the comparatively dark blue and red regions.

In Scilab images can be analyzed interactively. This is shown in Fig. 2.5.

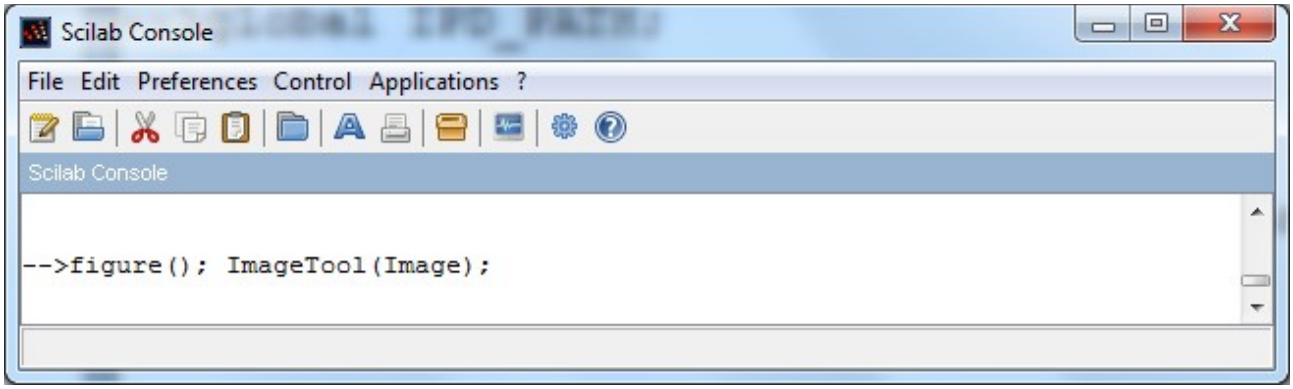


Fig. 2.5 a: The function ImageTool for interactive image analysis is called.

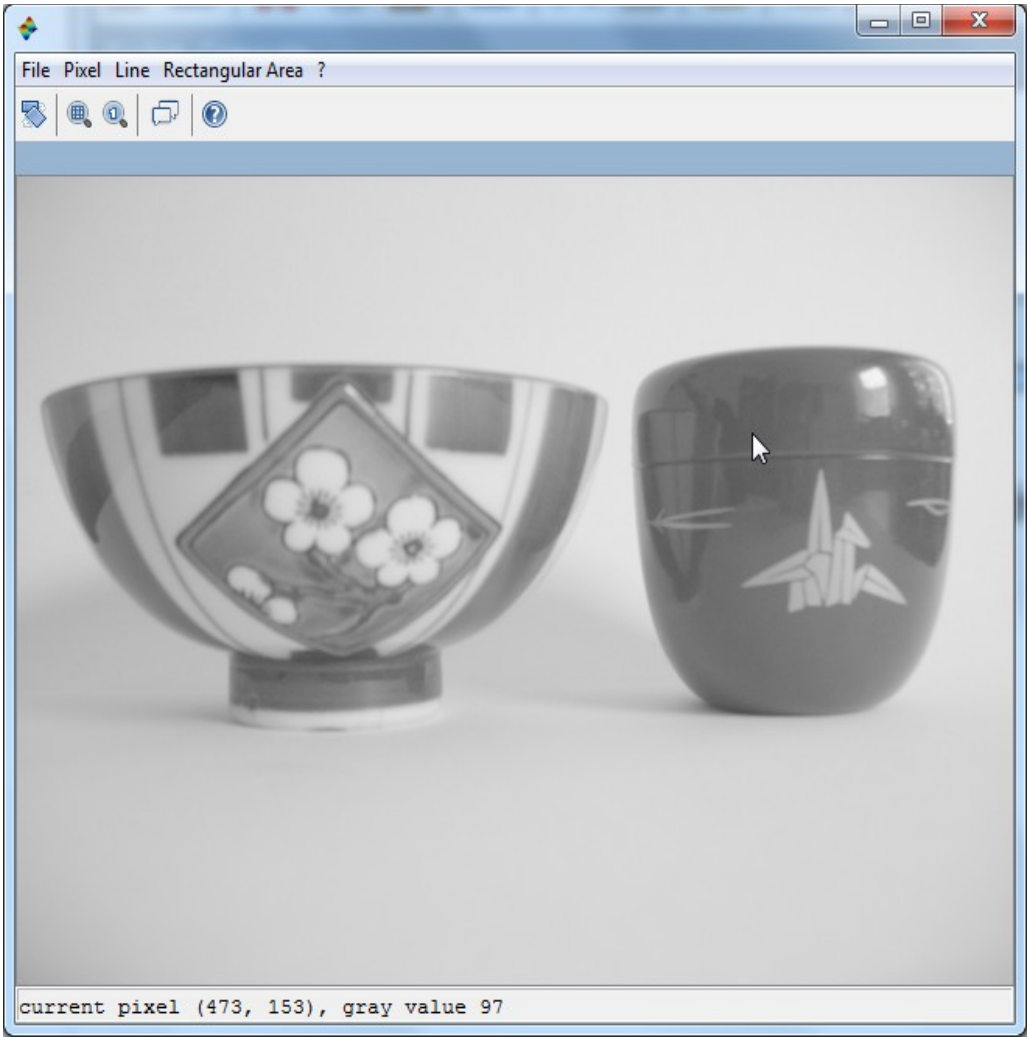


Fig. 2.5 b: The GUI for interactive image analysis. When the mouse pointer is inside the image, the position and gray value of the pixel the mouse pointer resides at is displayed. The user can select a single pixel, a line between two points or a rectangular area.

Figure 2.6 shows one rectangular area in the background, one in a blue region and one in the red region and the histogram of each region.

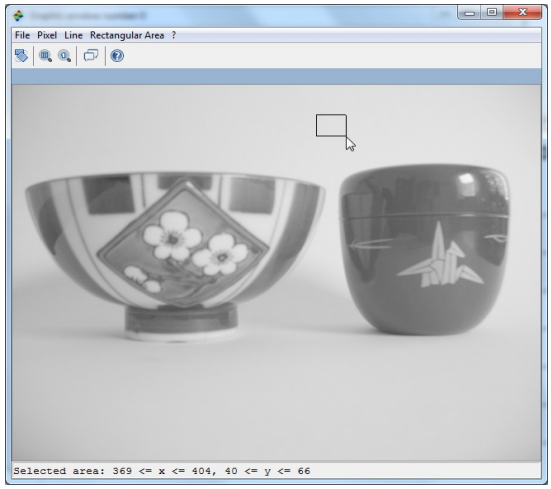


Fig. 2.6 a: Area in the white background.

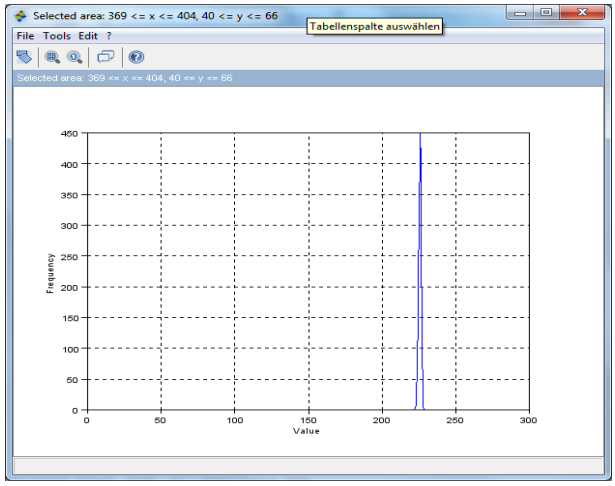


Fig. 2.6 b: Histogram of the white area.

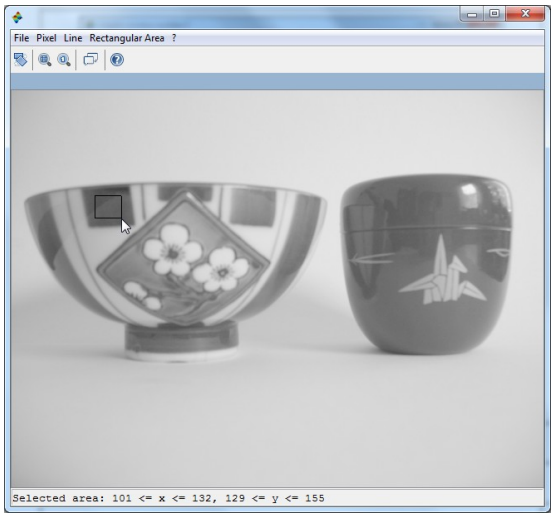


Fig. 2.6 c: Area in a blue region.

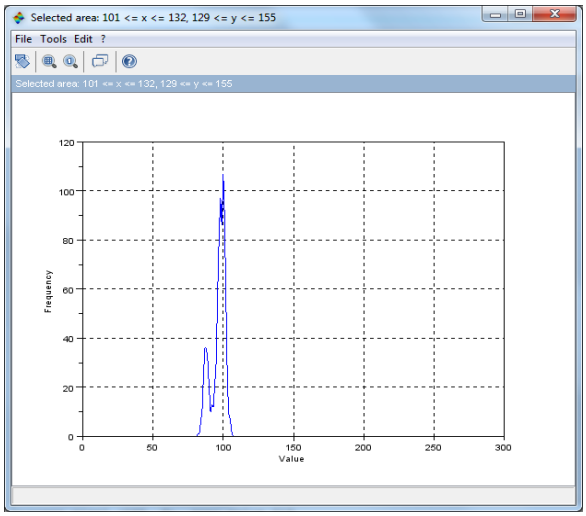


Fig. 2.6 d: Histogram of the blue area.

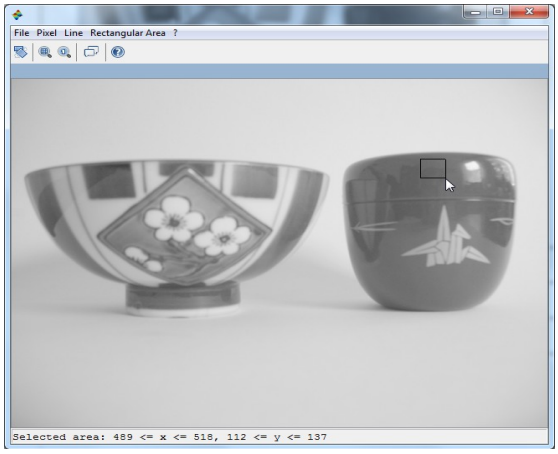


Fig. 2.6 e: Area in the red region.

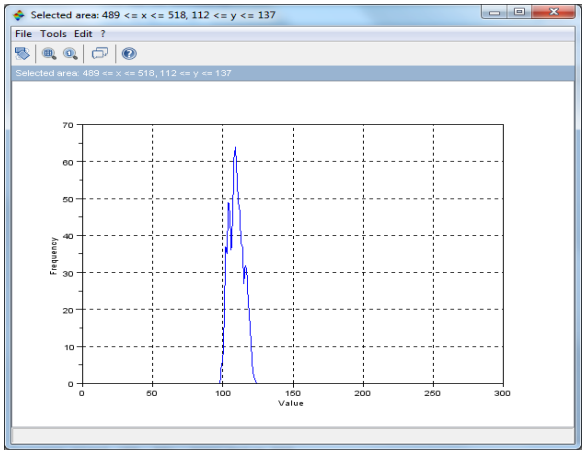


Fig. 2.6 f: Histogram of the red area.

As can be seen in Fig. 2.6, the gray level ranges of the red and blue regions partially overlap. The range of the selected white region does not overlap at all with the ranges of the red and blue regions. In this image, it is possible to distinguish between objects and background based on gray value ranges.

In the color image in 2.2 a there are blue, red, yellow and white regions. These colors are easily to distinguish when looking at the image. One might expect that the red and blue regions are clearly visible in the red and blue channel, respectively. However, this is not the case as can be seen in Fig. 2.7. In the red channel, the middle and right parts of the red are lighter than the background whereas the left part is darker. In the blue channel, the blue areas are darker than white. Moreover, the yellow region is brighter than the red one, even though yellow is certainly not more similar to blue than red.

The red – green – blue color space, also called “RGB”, is used for storing color image files and for displaying color images on screens. However, this color space does not seem to be appropriate for calculations on images. There are other color spaces more suitable for this purpose.

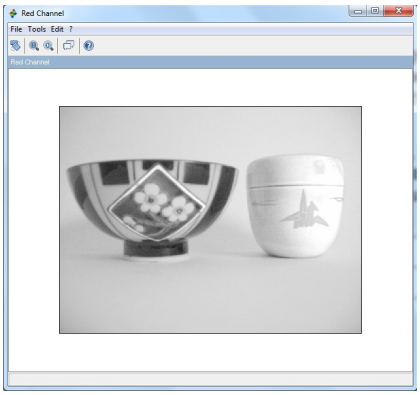


Fig. 2.7 a: Red channel.

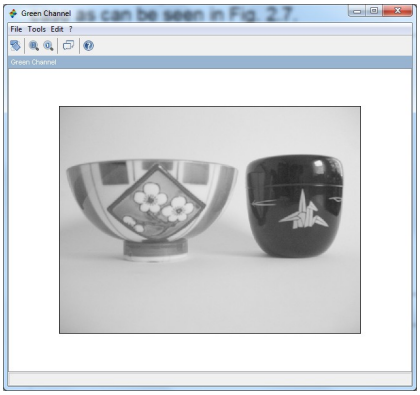


Fig. 2.7 b: Green channel.

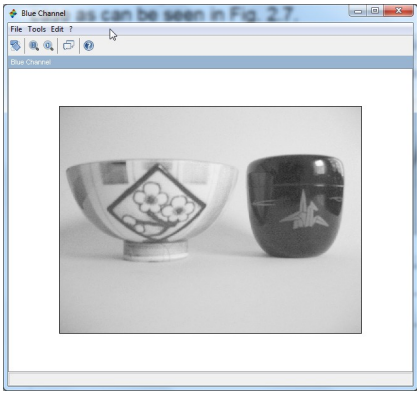


Fig. 2.7 c: Blue channel.

Fig. 2.8 shows how the channels of a color image can be displayed in Scilab.

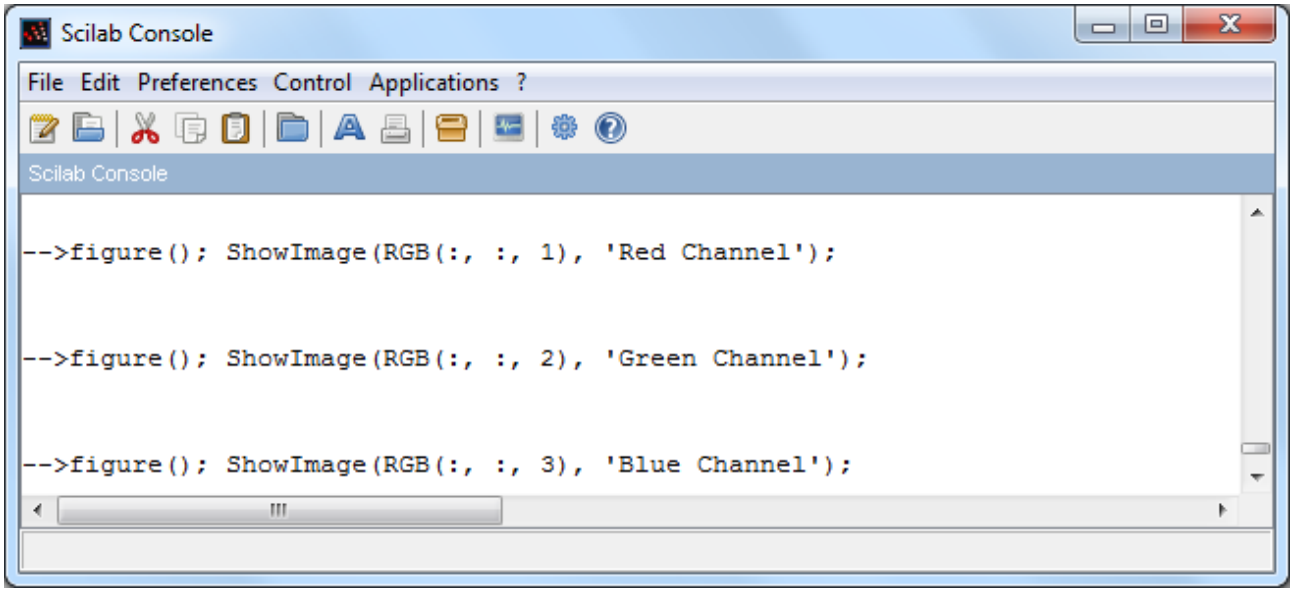


Fig. 2.8: The Scilab commands for displaying color channels are shown.

One of these color spaces is L*a*b*, also called “CIE_LAB” [2]. The L*a*b* is implemented in Scilab. Fig. 2.9 shows how an RGB image is converted to L*a*b* and how the components and their histograms can be visualized. The color channels are normalized so the minimum value is zero and the maximum value is one. This is done for the purpose of display and histogram calculation.

The results are shown in Fig. 2.10. The L* channel and its histogram look very similar to the gray level image in Fig. 2.2 b and its histogram shown in Fig. 2.5. The blue segments appear dark in both a* and b* channels whereas the red area appears bright in both channels. Both blue and red areas can be distinguished clearly from background. The yellow area appears dark in a* and bright in b*. The histograms of a* and b* have high peaks that correspond to the background and low broad maxima corresponding to objects.

In the L*a*b* space the different colors of this image can be distinguished from each other and from background better than in RGB space. It is important to choose an appropriate color space when working with color images.

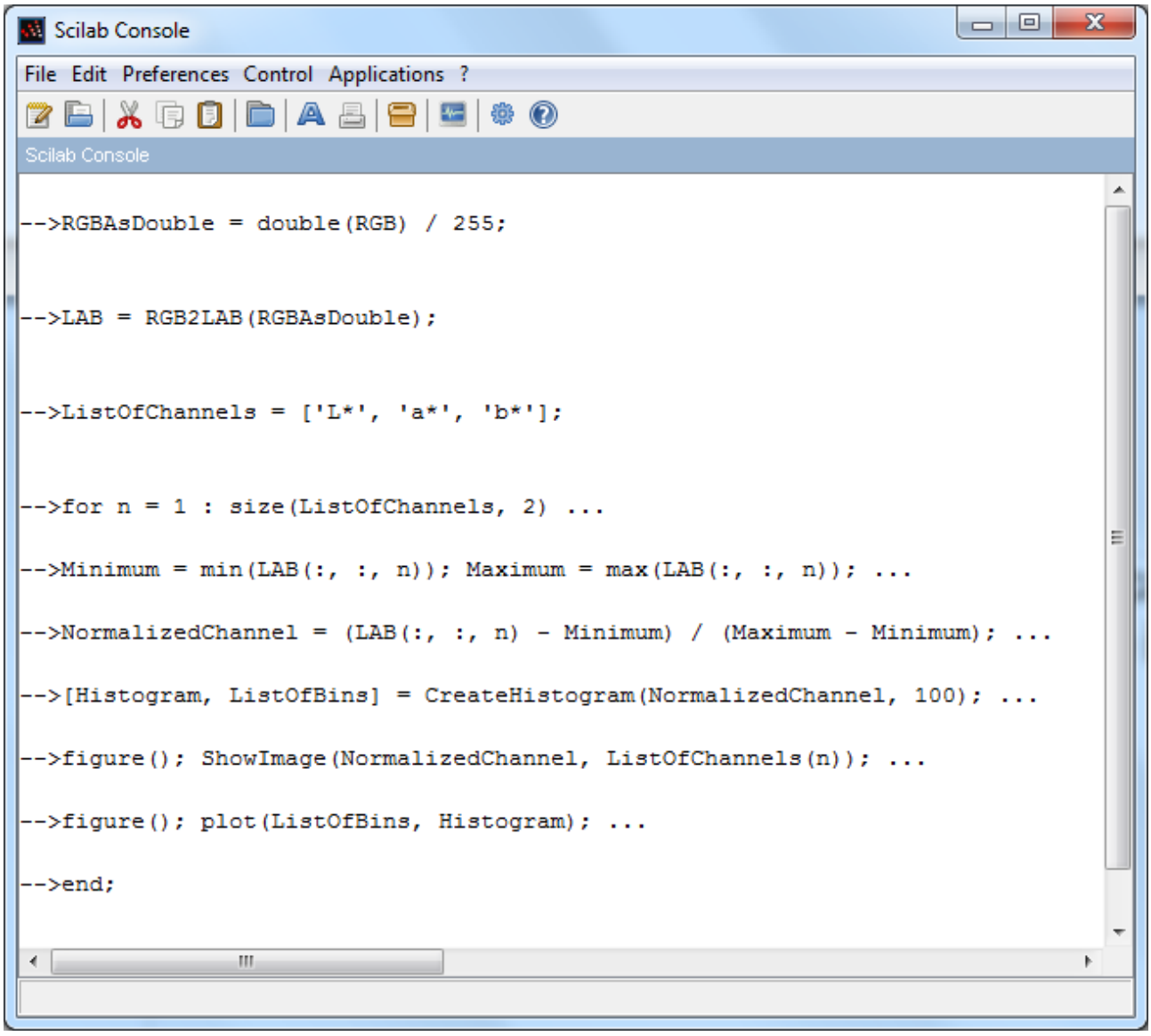


Fig. 2.9: Conversion of an RGB image to L*a*b* and visualization of the color channels and their histograms.

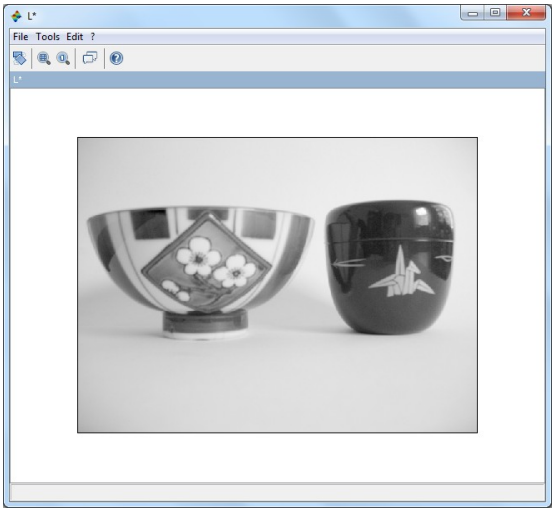


Fig. 2.10 a: Normalized L*.

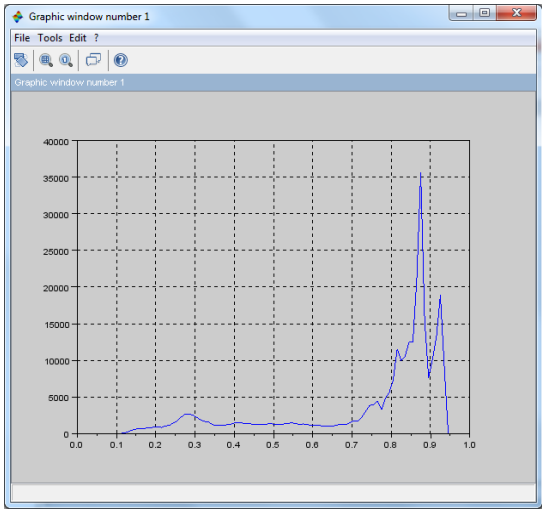


Fig. 2.10 b: Histogram of normalized L*.

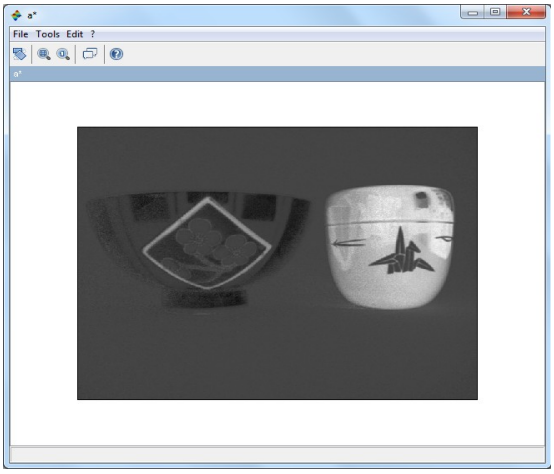


Fig. 2.10 c: Normalized a*.

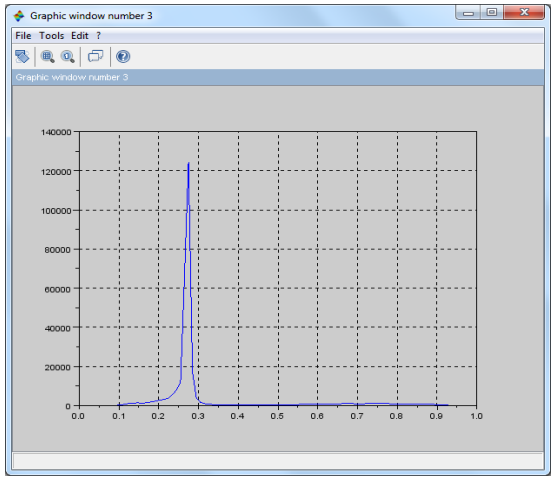


Fig. 2.10 d: Histogram of normalized a*.

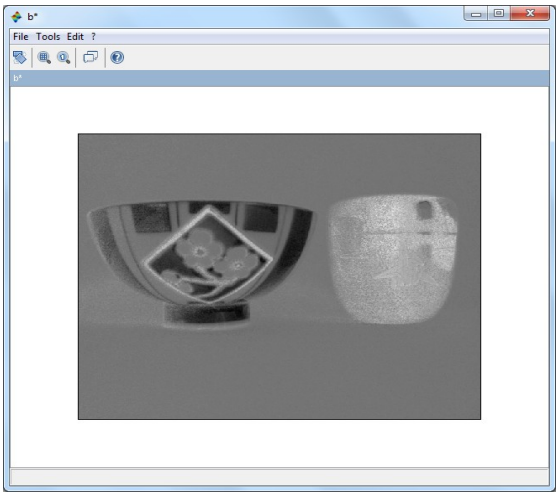


Fig. 2.10 e: Normalized b*.

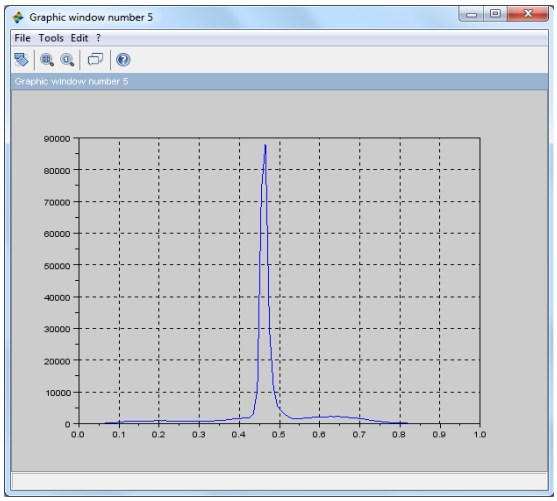


Fig. 2.10 f: Histogram of normalized b*.

3 Basic Operations on Images

3.1 Introduction

In this chapter basic image processing operations are introduced. These operations are useful for detecting objects in single channel images, i. e. in gray level, logical and object images. The following operations are described:

- Thresholding: The gray level of each pixel is compared to a threshold. If the gray level is equal to or greater than the threshold, the pixel is mapped to *true*. If the gray level is below the threshold, the pixel is mapped to *false*. The result is a logical image [3, 4].
- Blob analysis, also called “connected component analysis”: In a logical image the connected regions of *true* pixels are searched. The pixels of each connected region are mapped to a number greater than zero. All *false* pixels are mapped to zero. The result is an object image. For each object characteristic properties such as the centroid or the area given in pixels can be calculated [4].
- Filtering: For each pixel a new gray value, logical value or object label is calculated from an area surrounding it. This can emphasize objects of interest or remove irrelevant objects or noise. Moreover, the gray level gradient can be approximated and used for edge detection. Filtering can be applied to gray level images, logical images and object images. The result is of the same type as the original image [3, 4].
- Watershed transform: Starting from some seed points, unlabeled pixels are assigned to regions of pixels that are already labeled and have a lower or the same gray value [3].
- Distance transform: For each pixel the distance to the next background pixel is calculated. Watershed segmentation can be applied to the result image [3].

3.2 Thresholding

A gray level image can be segmented by comparing the gray value with a threshold at each pixel. In Scilab, pixels that have a gray value greater than or equal to the threshold are mapped to *true*, whereas all other pixels are mapped to *false*.

Thresholding helps to find objects in an image if these objects are significantly brighter or darker than the background. In this case there is a gray level range typical for object pixels and another gray level range typical for background pixels. A gray level between these ranges can be a threshold. A good candidate for a threshold is a local minimum of the histogram.

Fig. 3.1 shows how a threshold can be determined manually. The objects are darker than the background in the original image. Therefore, the image is inverted. Then the histogram is calculated and displayed. The data tip mode is toggled and a local minimum of the histogram is searched.

The threshold is applied to the inverted image. The original image, the inverted image and the result are displayed in Fig. 3.2. The bright regions of the resulting image correspond very well to the dark regions of the original image.

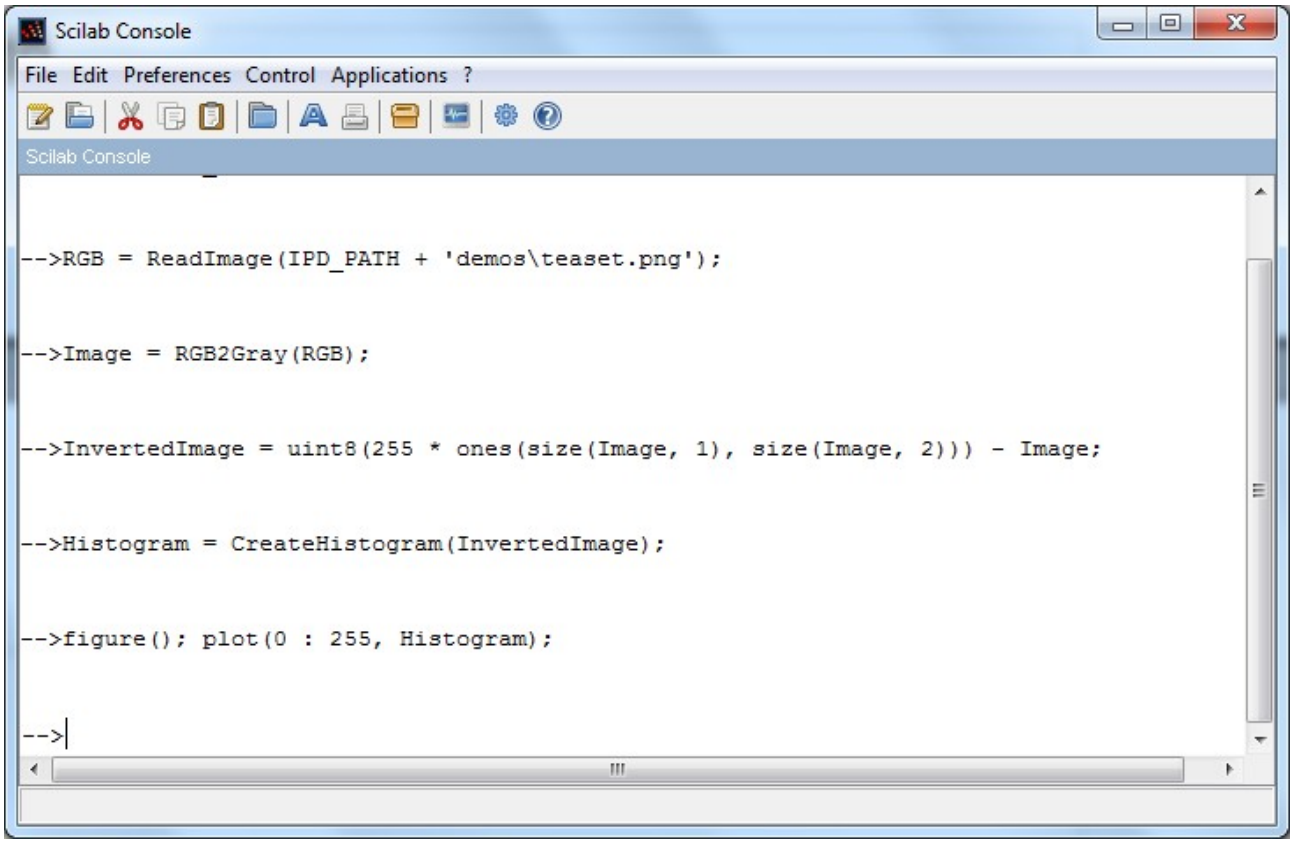


Fig. 3.1 a: The image is inverted, because the objects are dark and the background is bright. The histogram of the inverted image is displayed.

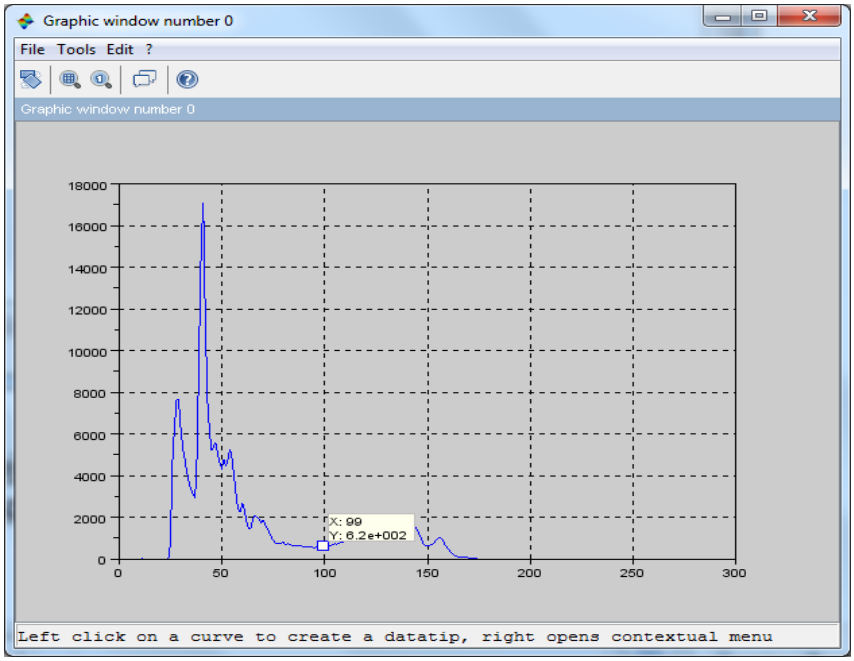


Fig. 3.1 b: The histogram of the inverted image is shown. A data tip is placed on a local minimum of the histogram.

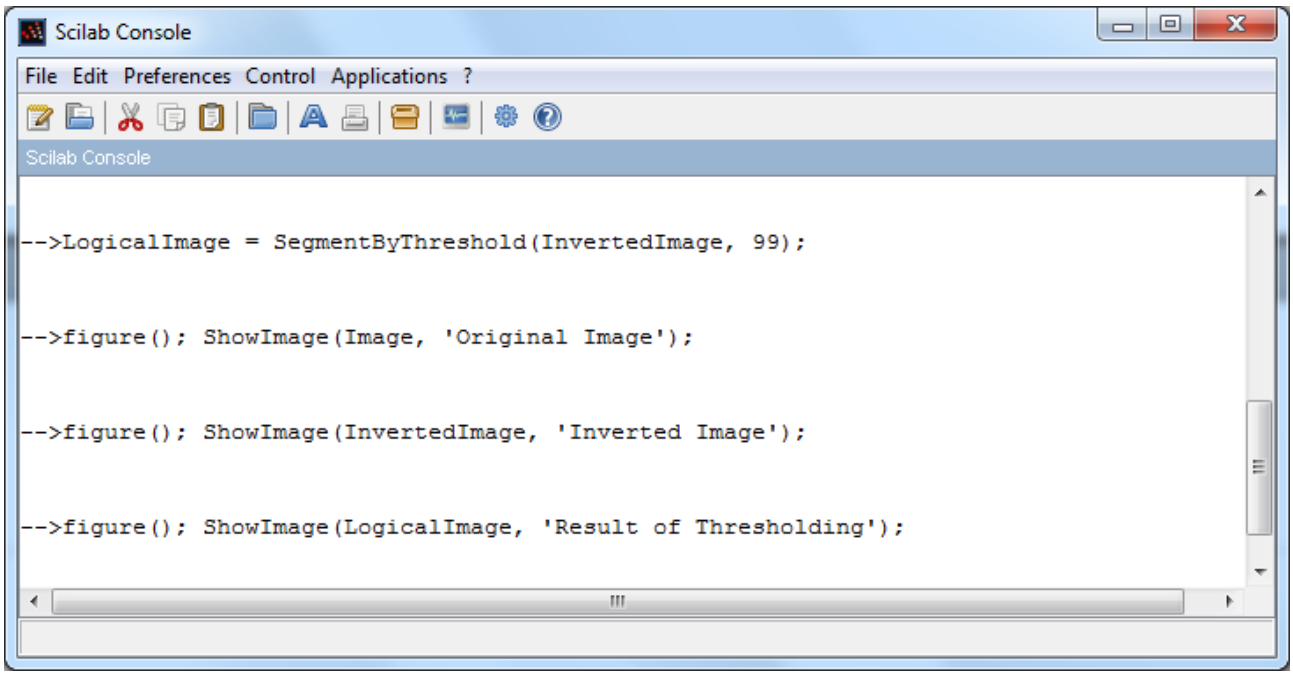


Fig. 3.2 a: The inverted image is segmented by the threshold.

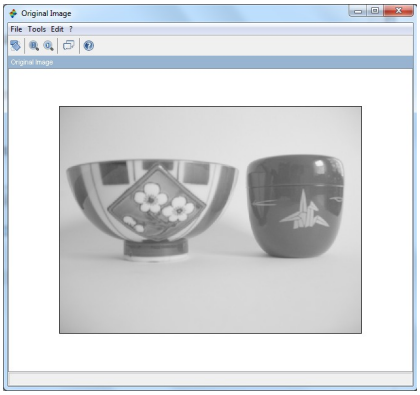


Fig. 3.2 b: Original image.

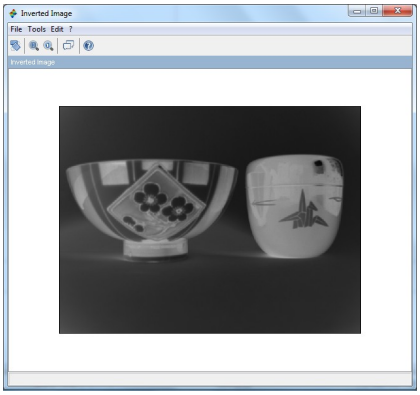


Fig. 3.2 c: Inverted image.

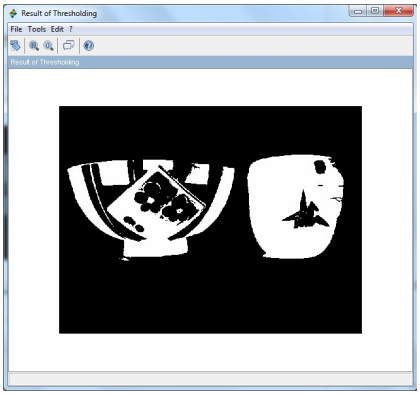


Fig. 3.3 d: Resulting image.

A threshold can be calculated automatically, too. A well known example for automated threshold calculation is the Otsu method: The value *k* that maximizes the between-class variance $\sigma_{between}$ is chosen as threshold. The between-class variance is defined by the following equations:

$$\sigma^2_{between} = \omega_{dark}\left(\mu_{dark} - \mu_{image}\right)^2 + \omega_{bright}\left(\mu_{bright} - \mu_{image}\right)^2 \quad \textbf{(3.1 a)}$$

$$\omega_{dark} = \sum_{q=0}^{k-1} p\left(r_q\right) \quad \textbf{(3.1 b)}$$

$$\omega_{bright} = \sum_{q=k}^{L-1} p\left(r_q\right) \quad \textbf{(3.1 c)}$$

$$\mu_{dark} = \frac{\sum_{q=0}^{k-1} r_q \, p(r_q)}{\omega_{dark}} \qquad \textbf{(3.1 d)}$$

$$\mu_{bright} = \frac{\sum_{q=k}^{L-1} r_q \, p(r_q)}{\omega_{bright}} \qquad \textbf{(3.1 e)}$$

$$\mu_{image} = \sum_{q=0}^{L-1} r_q \, p(r_q) \qquad \textbf{(3.1 f)}$$

$$p(r_q) = \frac{h(r_q)}{N} \qquad \textbf{(3.1 g)}$$

N is the number of pixels, L is the number of gray values, r_q is the *q*th gray value, $h(r_q)$ is the histogram value of r_q [3].

The Otsu method is implemented in Scilab. The calculation and the result are shown in Fig. 3.3.

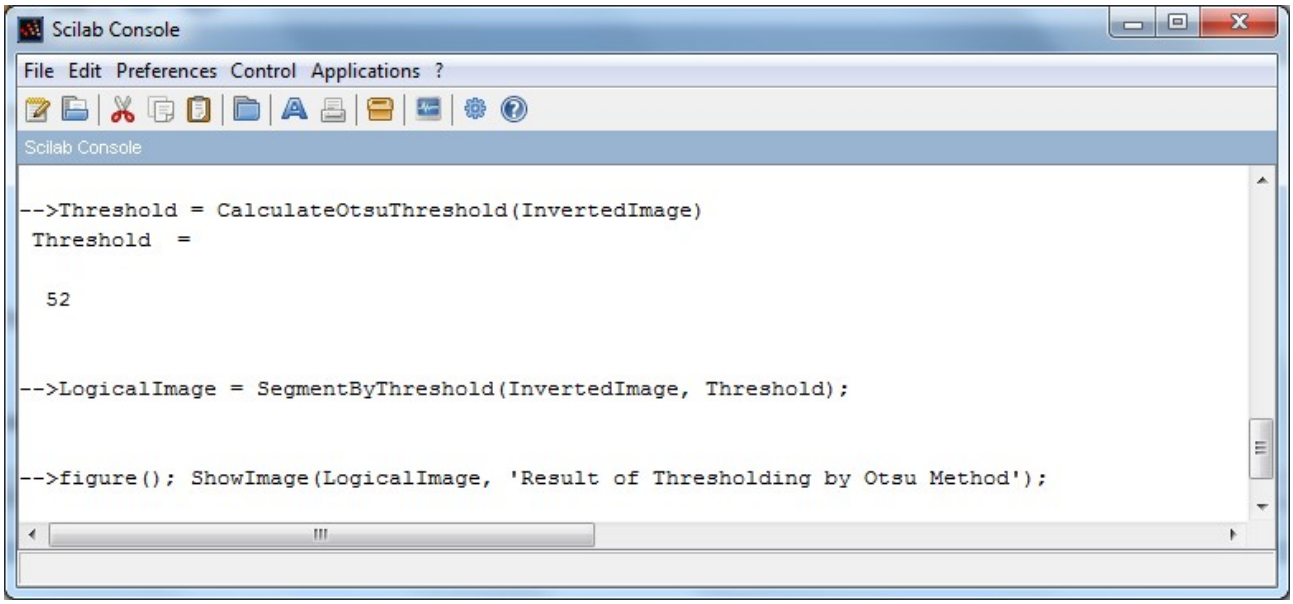


Fig. 3.3 a: Calculation of threshold using Otsu's method.

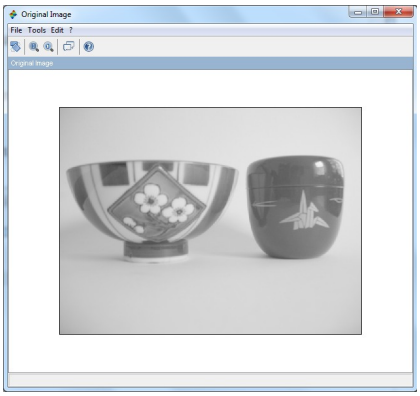


Fig. 3.3 b: Original image.

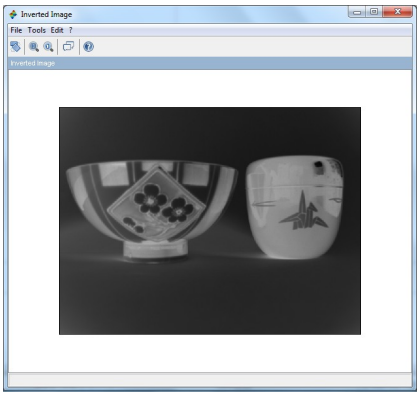


Fig. 3.3 c: Inverted image.

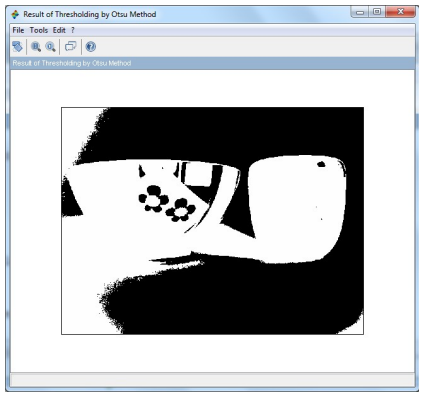


Fig. 3.3 d: Resulting image.

The result is not very accurate, because shadows and comparatively dark regions in the left part of the image have a gray value higher than the calculated threshold.

In Fig. 3.4 the background is analyzed by calling the function ImageTool(), selecting a pixel and displaying a row profile. All pixels on this row belong to the background. The gray value strongly varies on the row the selected pixel belongs to. The background brightness is inhomogeneous because the illumination was inhomogeneous when the original photograph was taken. Therefore, it is difficult to calculate a threshold automatically for the gray level image.

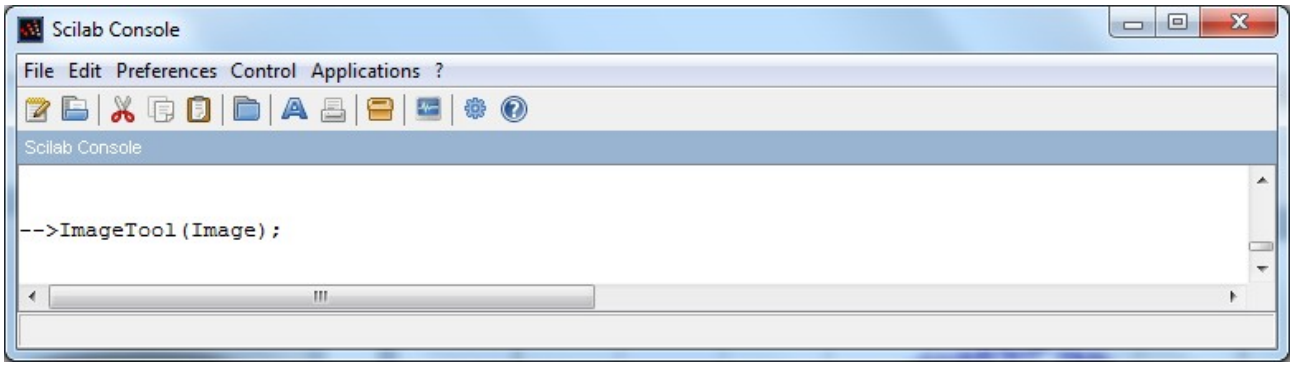


Fig. 3.4 a: The function ImageTool() is called for analyzing the image.

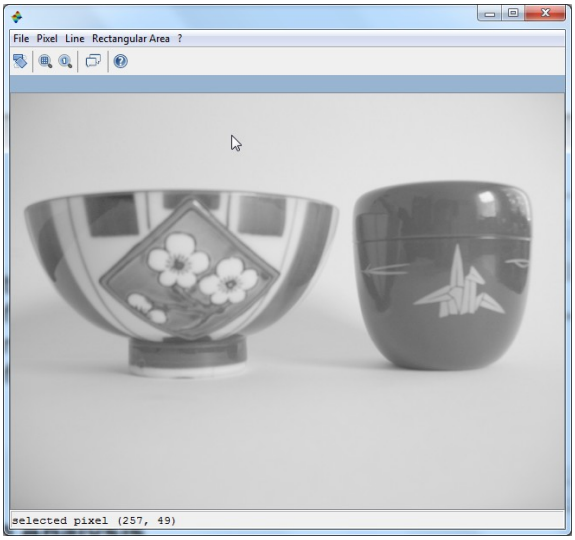


Fig. 3.4 b: A pixel is selected.

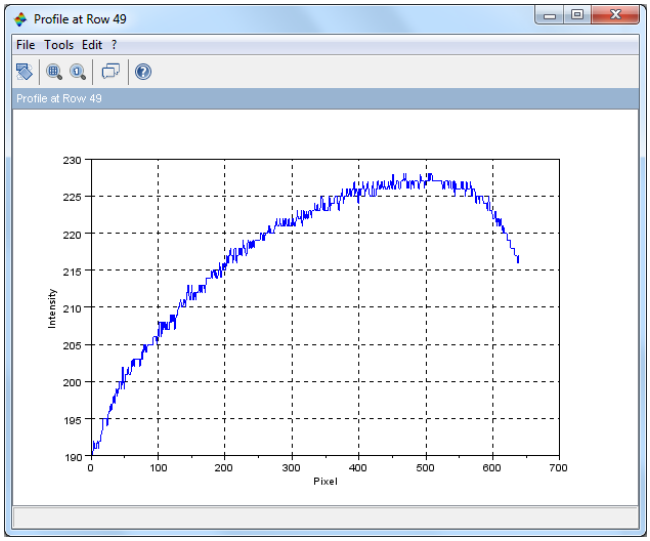


Fig 3.4 c: Gray value profile of the row the selcted pixel belongs to.

In the following example, the image is cropped and the threshold is calculated on the cropped inverted image. The threshold is close to the manually determined one and the result is quite accurate as we can see in Fig. 3.5. The histogram of the inverted image has a very high peak as can be seen in Fig. 3.1. This peak ends at the threshold calculated on the original gray level image. Contrasting this, the cropped inverted image does not have a peak that is much higher than all other peaks. The background brightness does not vary as strongly in the cropped image as in the original one.

The Otsu method is easy to implement and fast. However, it is important to consider back-

ground brightness variations when trying to segment an image using this method. Background variations can be caused by inhomogeneous illumination or by specular reflection.

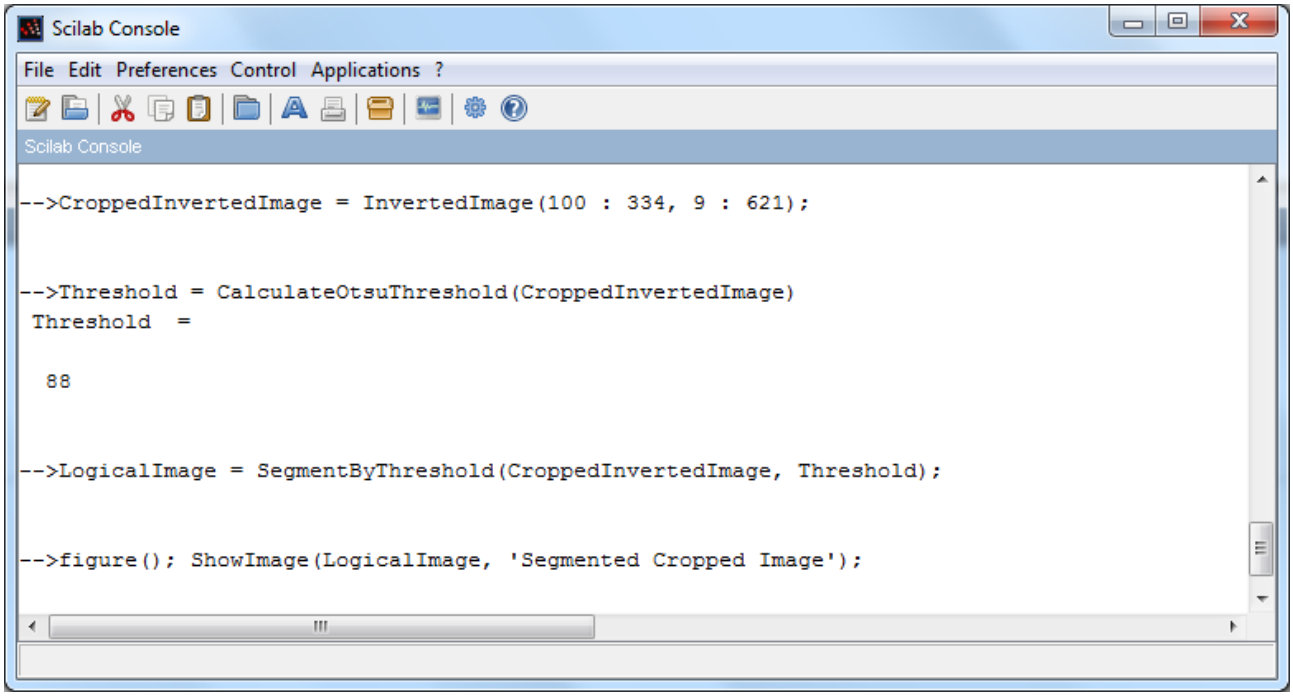


Fig. 3.5 a: The image is cropped and the threshold is calculated on the cropped inverted image.

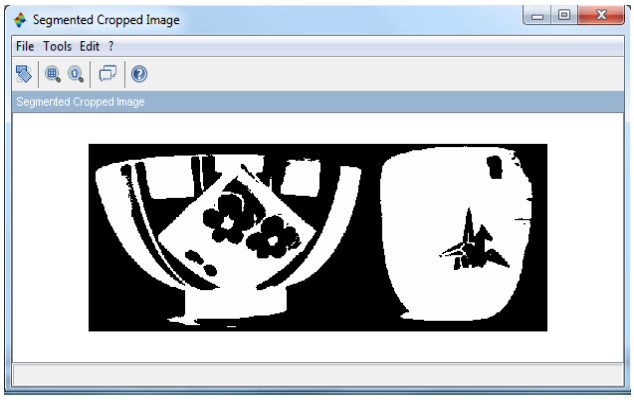


Fig. 3.5 b: The result of thresholding.

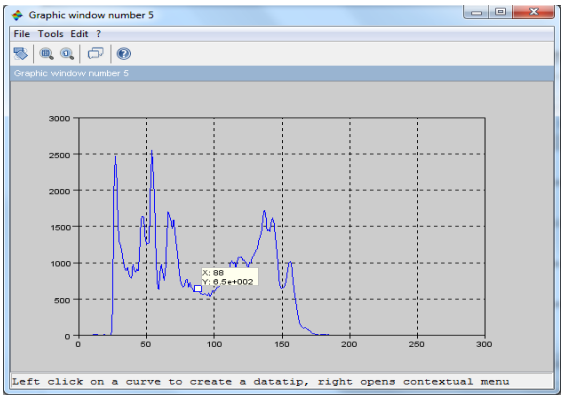


Fig. 3.5 c: The histogram of the cropped inverted image.

3.3 Blob Analysis

Objects can be found in a logical image by searching the connected areas of *true* pixels. The pixels of each connected area are mapped to an integer number greater than zero. All pixels of the same area have the same number whereas pixels belonging to different areas have different numbers. All false pixels are mapped to zero.

An algorithm for searching the connected components is described in [4] and this algorithm is implemented in Scilab. Fig. 3.6 shows how a blob analysis can be done in Scilab.

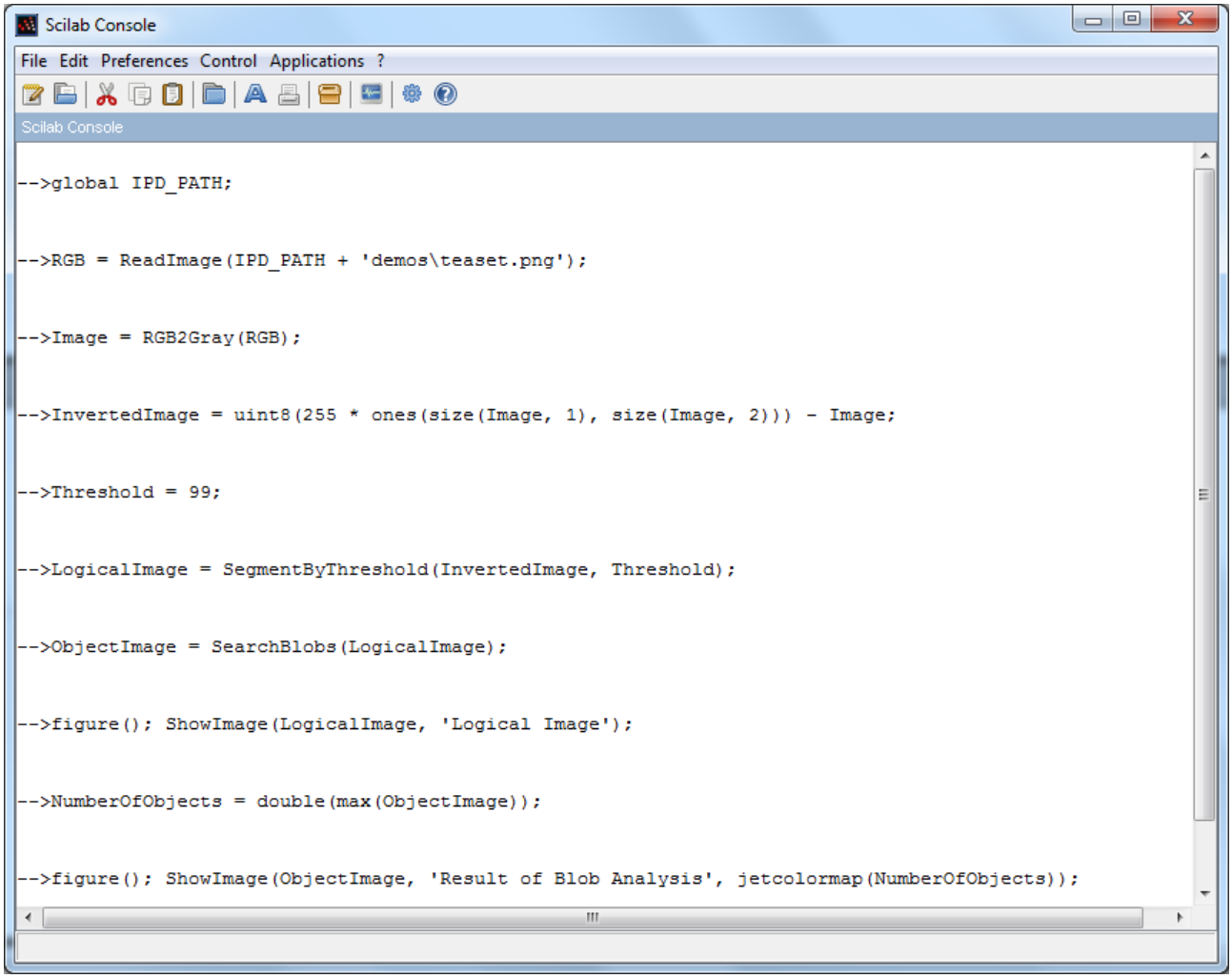


Fig. 3.6 a: A blob analysis is performed using the function SearchBlobs(). The result is shown by ShowImage(). The color map has as many entries as the resulting image has objects.

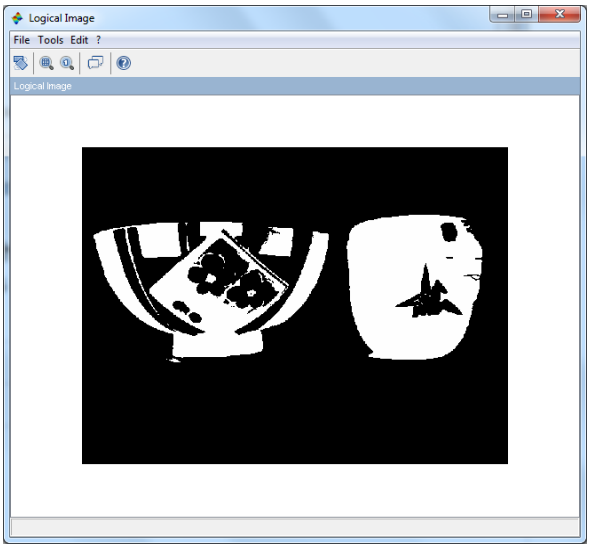


Fig. 3.6 b: The logical image.

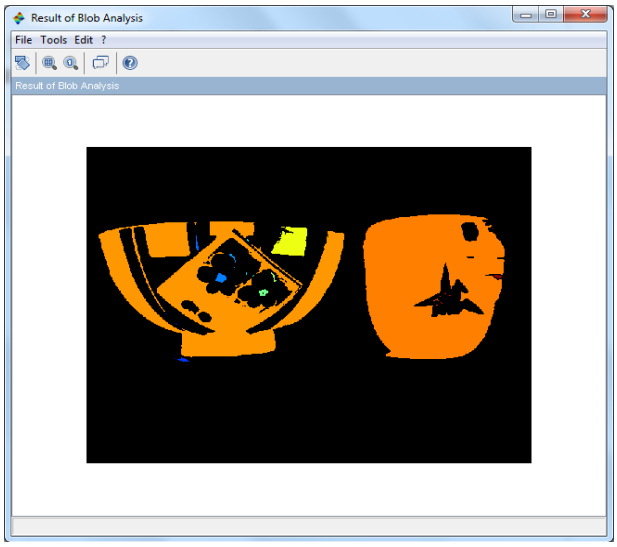


Fig. 3.6 c: The result of blob analysis.

3.4 Filtering

3.4.1 Introduction

Filtering means that a mask is placed on each pixel and a new gray value or logical value is calculated from the pixels below the mask. Objects of interest can be emphasized and irrelevant objects can be removed applying a filter.

In this document the coordinate system of a filter mask **f** is defined as shown in Fig. 3.7:

f(-1, -1)	*f*(-1, 0)	*f*(-1, 1)
f(0, -1)	*f*(0, 0)	*f*(0, 1)
f(1, -1)	*f*(1, 0)	*f*(1, 1)

Fig. 3.7: The origin of a filter mask coordinate system is on the central element.

The following types of filtering are described in this document:

- Linear filtering: The new gray value is a weighted sum of the gray values of the pixels below the mask.
- Median filtering: The median is calculated from the gray values or logical values of the pixels below the mask.
- Morphological filtering: Morphological filters can expand or shrink objects and connect or divide areas.

3.4.2 Linear Filtering

When an image **I** is filtered with a mask **f** that has *N* rows and *M* columns, the gray value $I_{filtered}(i,j)$ at the pixel (i,j) is calculated in the following way:

$$I_{filtered}(i,j) = \sum_{n=n_{min}}^{n_{max}} \sum_{m=m_{min}}^{m_{max}} f(n,m)\, I(i-n, j-m) \qquad \textbf{(3.2 a)}$$

$$\begin{aligned} -k \le n \le k,\ \ N = 2k+1 \\ -k+1 \le n \le k,\ \ N = 2k \end{aligned} \qquad \textbf{(3.2 b)}$$

$$\begin{aligned} -k \le m \le k,\ \ M = 2k+1 \\ -k+1 \le m \le k,\ \ M = 2k \end{aligned} \qquad \textbf{(3.2 c)}$$

An example is shown in Fig. 3.8: The original image consist of an upper region with gray value zero and a lower region with gray value one. The matrix

$$\boldsymbol{f} = \frac{1}{3}\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \qquad \textbf{(3.3)}$$

is used as filter mask. In the filtered image, the pixels in the lowest row of the upper half and the highest row of the lower half have the gray value one whereas all other pixels have the gray value zero. The mask used in this example emphasizes horizontal edges.

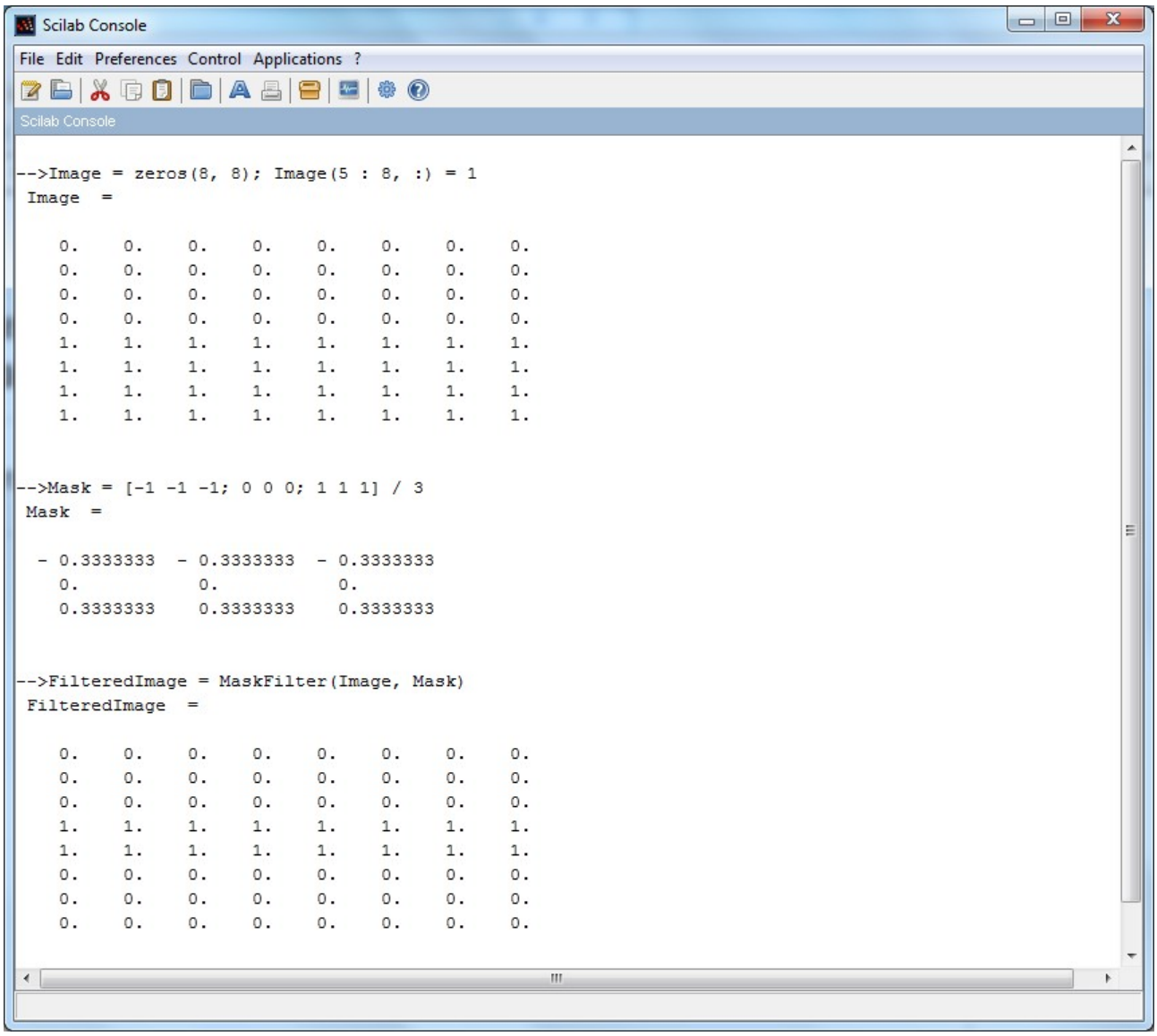


Fig. 3.8: An image is filtered with a mask that emphasizes horizontal edges.

The matrix of the previous example can be expressed at the product of a matrix with only one column and a matrix with only one row:

$$\boldsymbol{f}_{column} = \frac{1}{\sqrt{3}} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \qquad \textbf{(3.4 a)}$$

$$\boldsymbol{f}_{row} = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \qquad \textbf{(3.4 b)}$$

$$\boldsymbol{f} = \boldsymbol{f}_{column} \cdot \boldsymbol{f}_{row} \qquad \textbf{(3.4 c)}$$

In other words, this matrix is separable. Fig. 3.9 shows how an image can be filtered with a separable matrix. The column filter is applied to the columns and after this the row filter is applied to the rows or the other way around. This reduces the number of multiplications and additions when compared to using a matrix with several rows and columns and therefore speeds up the computation.

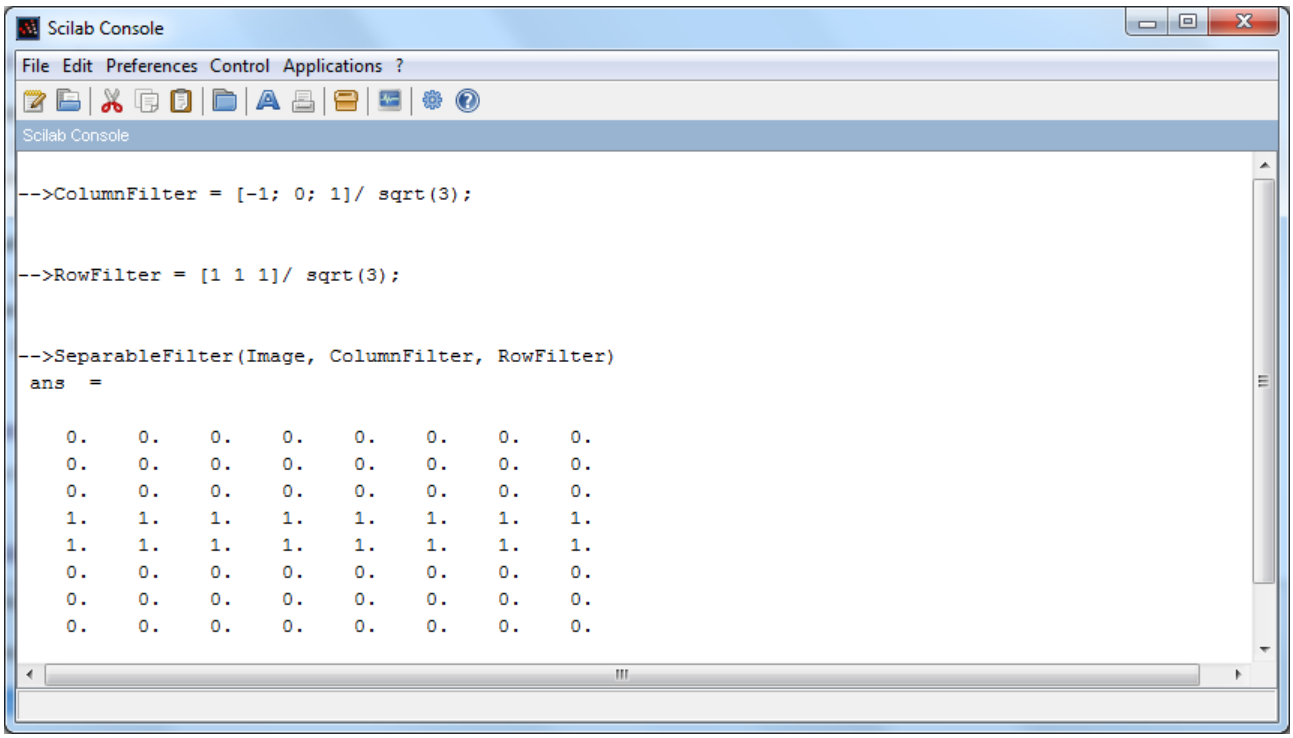


Fig. 3.9: The mask can be represented as a product of a row filter and a column filter. The function SeparableFilter() applies both filters to the image. The result is the same as in Fig. 3.8.

3.4.3 Median Filtering

When a median filter is applied to a gray level image, the median of the gray values of the pixels below the mask is calculated for each pixel. When a median filter with an odd number of elements is applied to a logical image, the pixel in the center is mapped to *true*, if there are more *true* than *false* pixels below the mask.

The list of the gray values at the pixels below the mask must be sorted and the median must be searched if the number of elements is odd or even be calculated if the number of elements is even. Therefore, median filtering is very time consuming when compared to other filtering methods.

Median filtering can be applied for removing small noise objects as can be seen in Fig. 3.10.

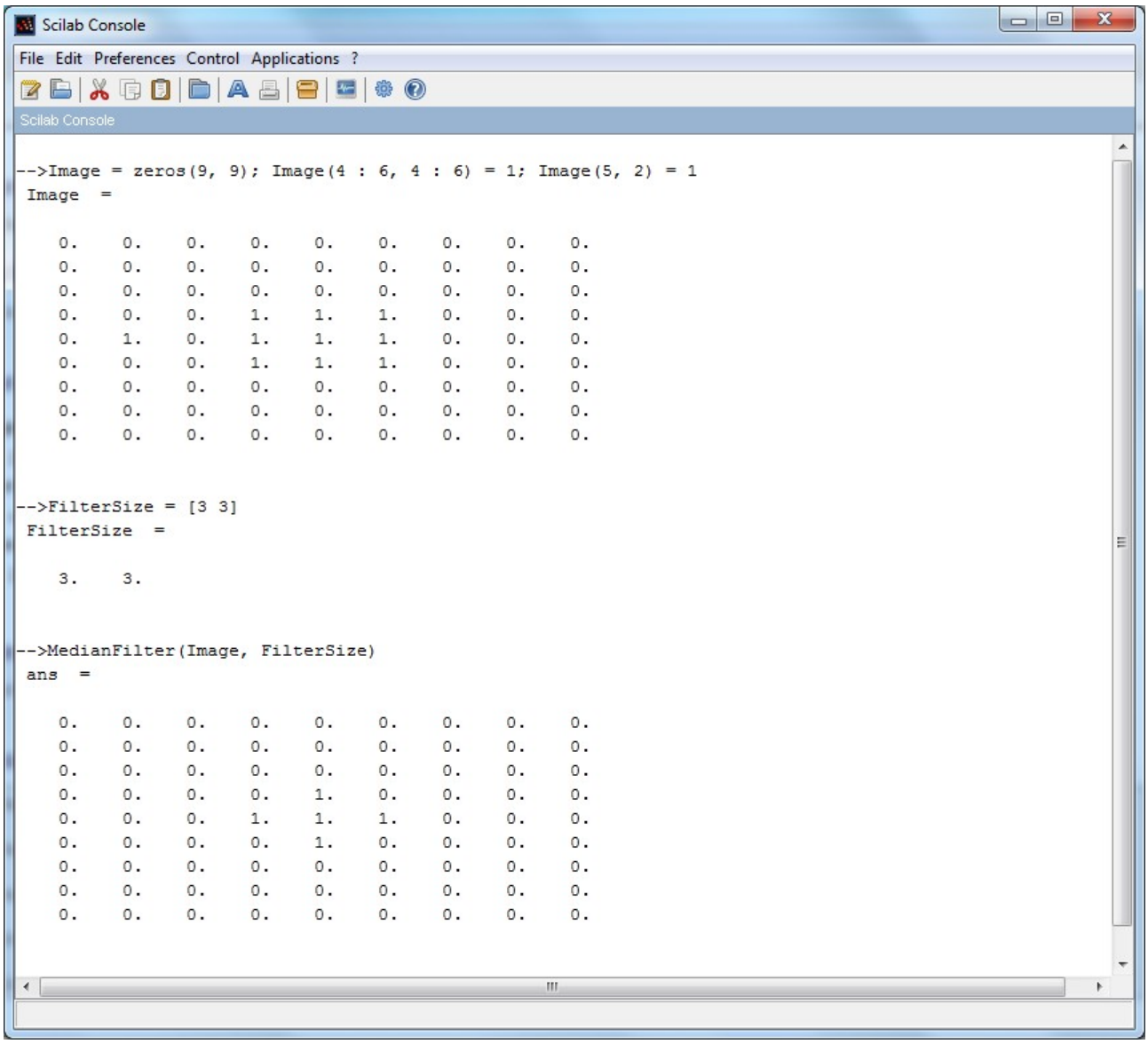


Fig. 3.10: In the original image there are a blob of nine object pixels and a single object pixel. The median filter removes the object consisting of only one pixel. The other object is still there even though some of its pixels become background pixels.

3.4.4 Morphological Filtering

Morphological filters expand or shrink objects and connect or disconnect areas in an image. The mask of a morphological filter is a matrix of logical values. If an element of this matrix is *true*, the pixel below it is included into the calculation of the new gray value, logical value or object index. If the element is *false*, the pixel below it is not included into the calculation. The set of *true* pixels of the filter mask is also called “structuring element”.

The basic morphological operations are dilation and erosion. Dilation maps the gray value to the maximum of the gray values blow the structuring element whereas erosion maps the gray value to the minimum. Mathematically, the dilation and erosion of an image **I** with a mask **f** that has N rows and M columns are defined as follows:

$$I_{dilated}(i,j) = \max_{f(n,m)=true} I(i+n, j+m) \tag{3.5 a}$$

$$I_{eroded}(i,j) = \min_{f(n,m)=true} I(i+n, j+m) \tag{3.5 b}$$

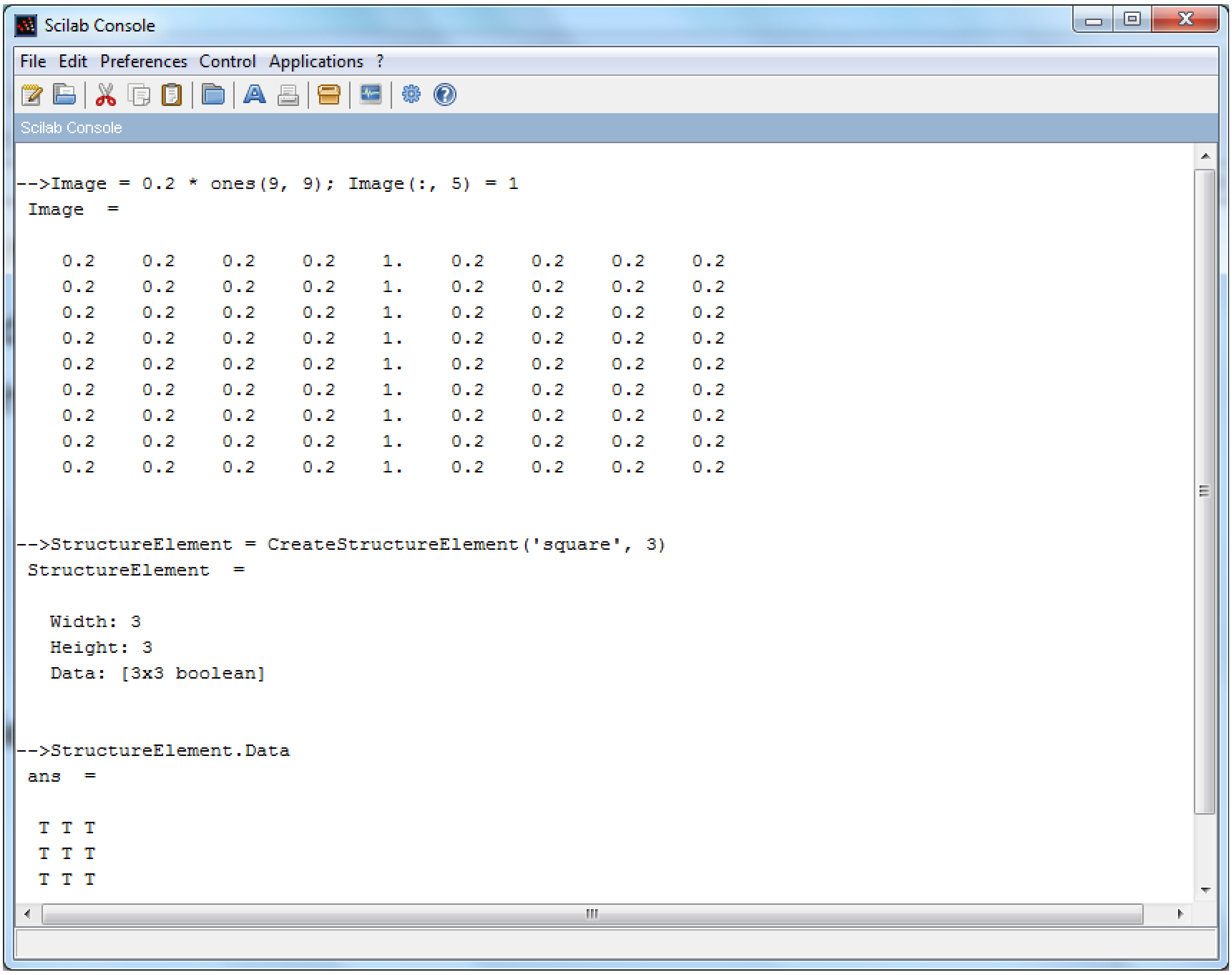
$$\begin{aligned} -k \le n \le k, \quad N = 2k+1 \\ -k+1 \le n \le k, \quad N = 2k \end{aligned} \tag{3.5 c}$$

$$\begin{aligned} -k \le m \le k, \quad M = 2k+1 \\ -k+1 \le m \le k, \quad M = 2k \end{aligned} \tag{3.5 d}$$

Further morphological operations are:

- Closing: dilation followed by erosion
- Opening: erosion followed by dilation
- Top Hat: emphasizes bright objects on a dark background, $I_{tophat} = I - I_{opened}$
- Bottom Hat: emphasizes dark objects on a bright background, $I_{bottomhat} = I_{closed} - I$

Fig. 3.11 shows the effects of dilation, erosion, closing, opening, top hat and bottom hat. Morphological operations are explained in detail in [3].

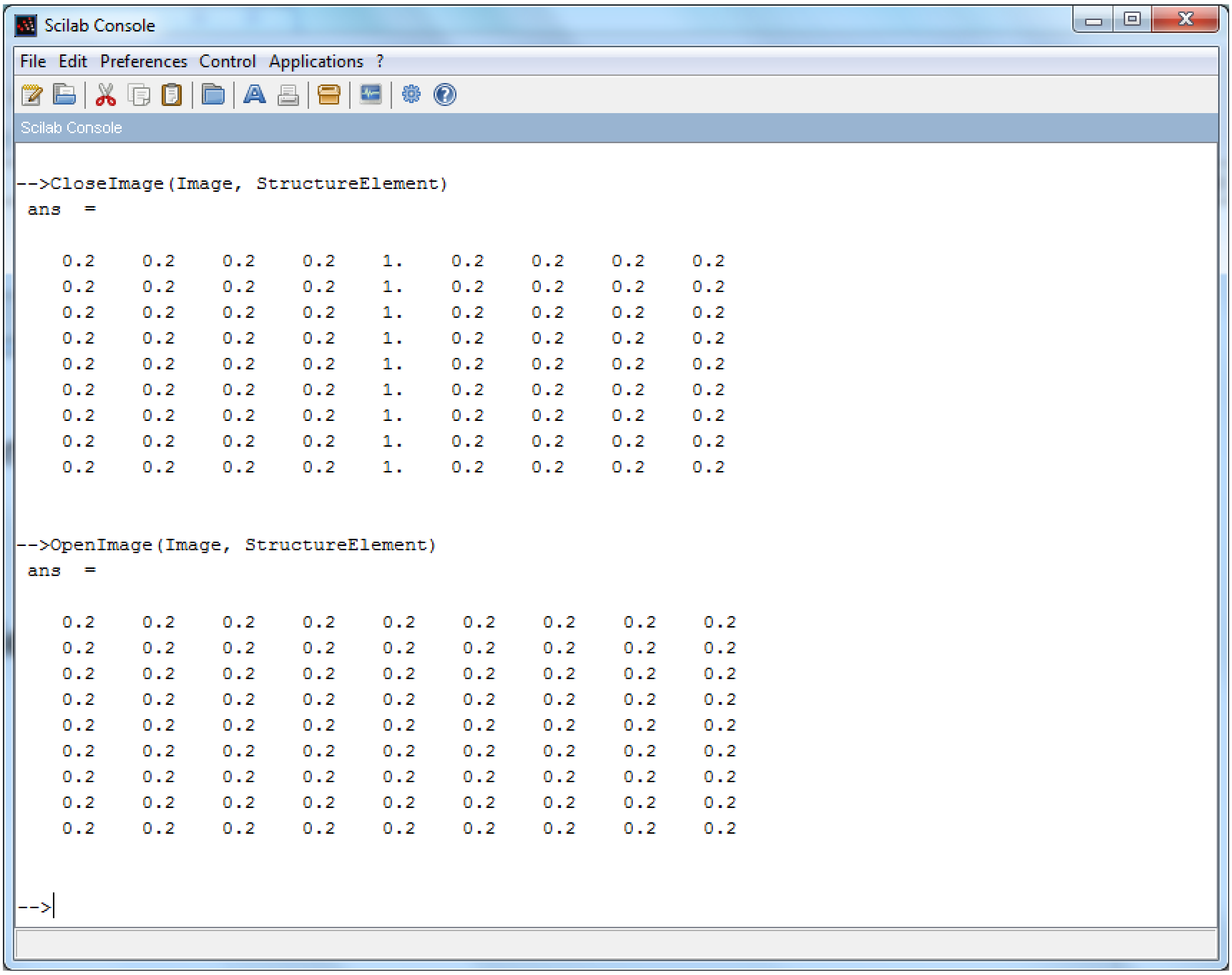


```
-->Image = 0.2 * ones(9, 9); Image(:, 5) = 1
 Image  =

    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2


-->StructureElement = CreateStructureElement('square', 3)
 StructureElement  =

   Width: 3
   Height: 3
   Data: [3x3 boolean]


-->StructureElement.Data
 ans  =

  T T T
  T T T
  T T T
```

Fig. 3.11 a: The image and the structuring element of the following examples are defined.

```
-->DilateImage(Image, StructureElement)
 ans  =

    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2
    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2
    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2
    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2
    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2
    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2
    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2
    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2
    0.2    0.2    0.2    1.    1.    1.    0.2    0.2    0.2


-->ErodeImage(Image, StructureElement)
 ans  =

    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2


-->
```

Fig. 3.11 b: Dilation makes the bright line broader whereas erosion removes the bright line.



```
-->CloseImage(Image, StructureElement)
 ans  =

    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    1.    0.2    0.2    0.2    0.2


-->OpenImage(Image, StructureElement)
 ans  =

    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2
    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2    0.2


-->
```

Fig. 3.11 c: Closing leaves the bright unchanged, because the effect of dilation is canceled out by the following erosion. Contrasting this, opening removes the bright line. Objects that are completely removed by erosion can not restored by dilation.

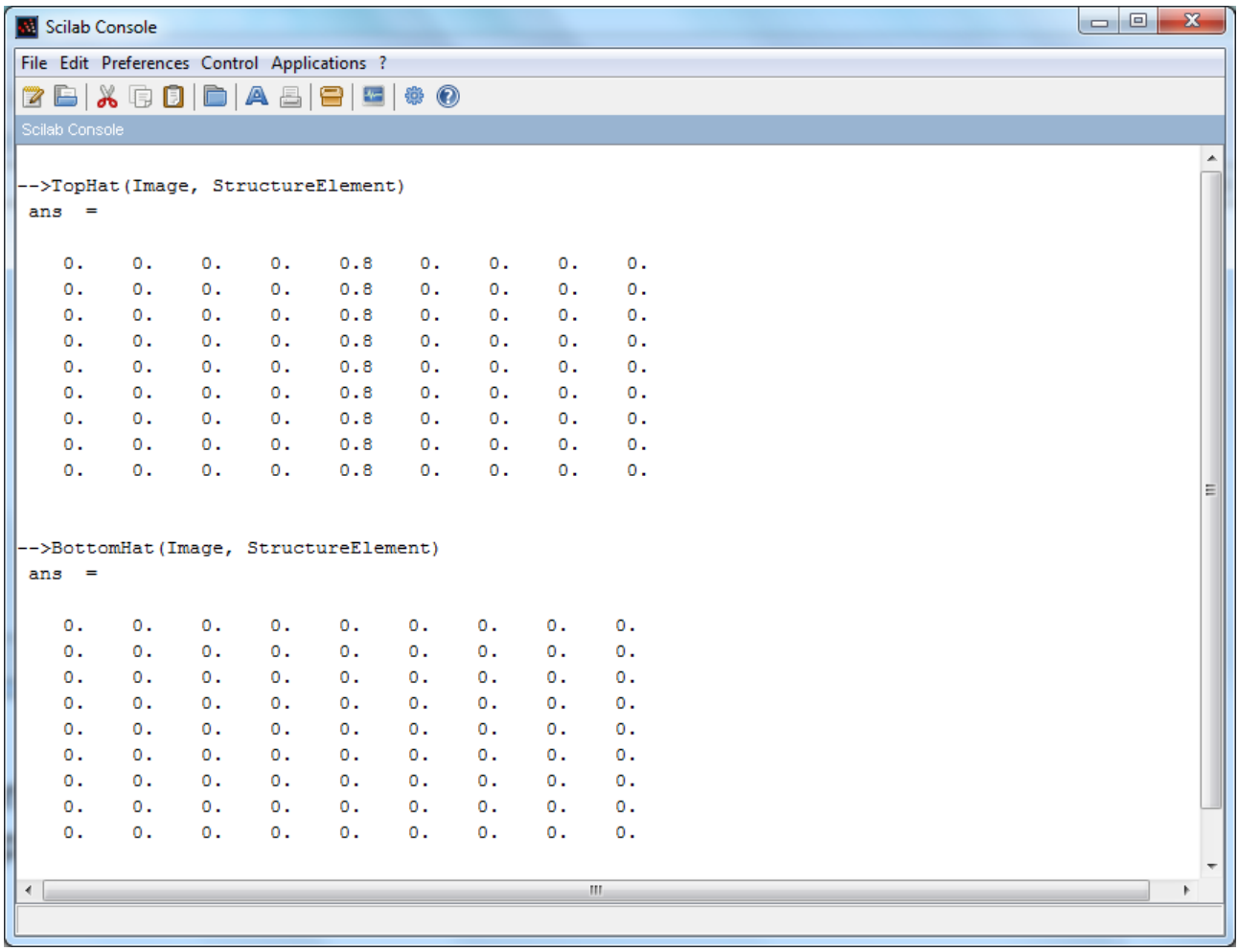


Fig. 3.11 d: The top hat filter leaves the bright line intact whereas the bottom hat filter completely removes it.

3.5 Watershed Transform and Distance Transform

Starting from some seed points, unlabeled pixels are assigned to regions of pixels that are already labeled and have a lower or the same gray value. Fig. 3.12 shows how the watershed transform works.

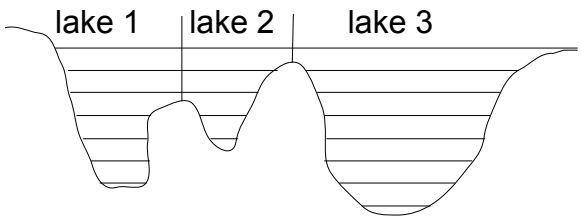


Fig. 3.12: Water fills several basins. There are watersheds where the water level exceeds the depth of basins. The between watersheds correspond to regions of an image segmented by watershed transform.

Objects can be detected by applying a gradient filter and selecting seed points in regions where the gradient is low. Fig. 3.13 shows the results for manually selected seed points.

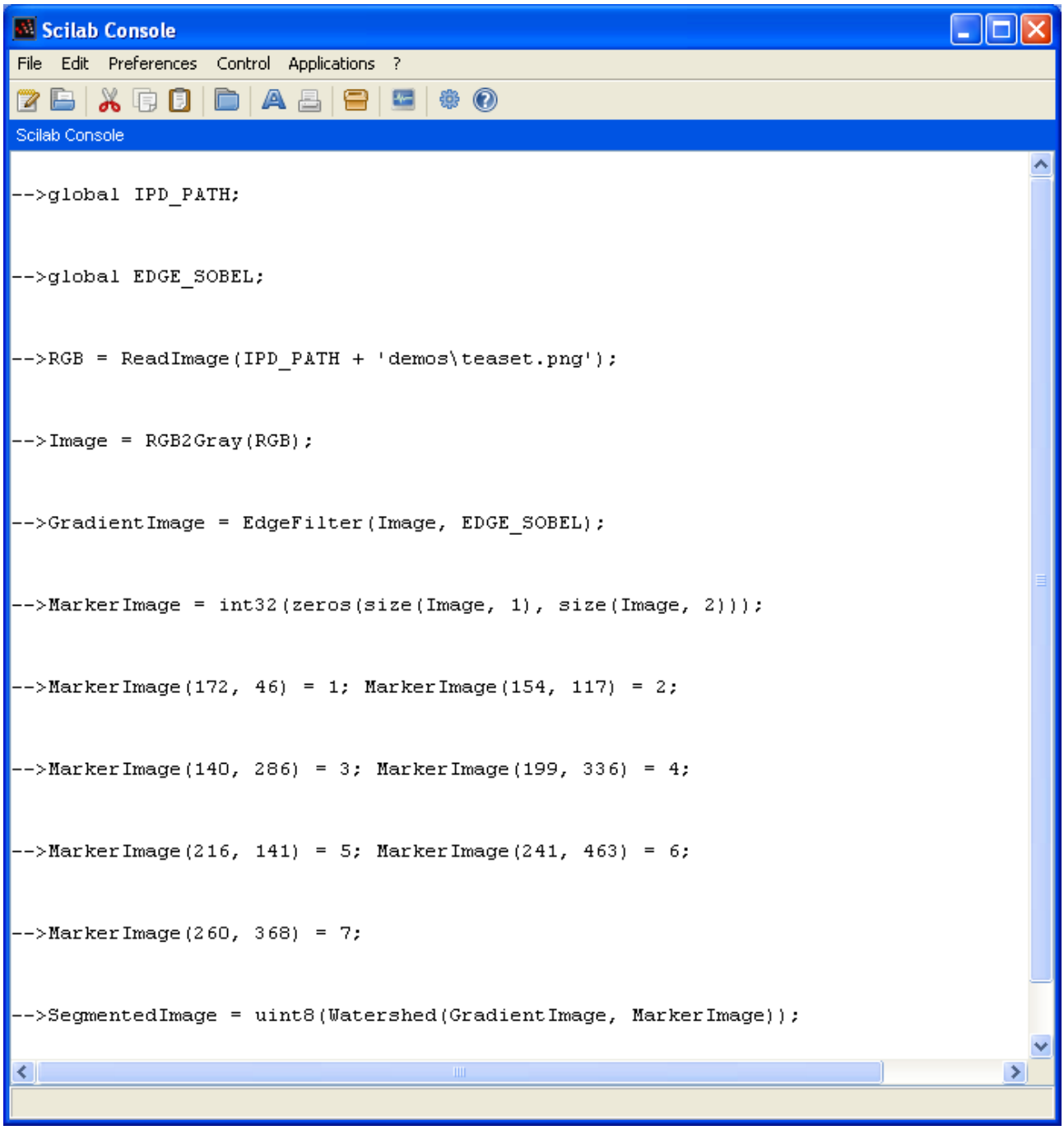
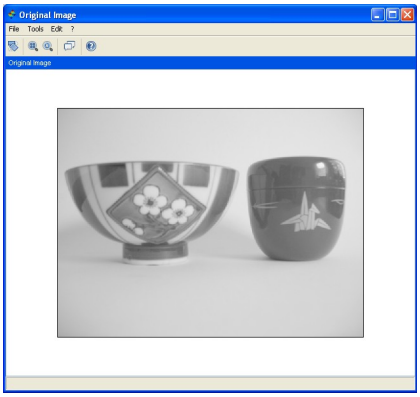
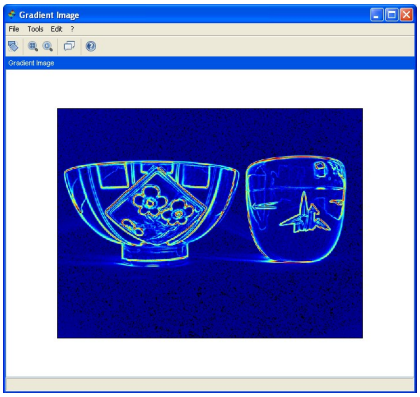


Fig. 3.13 a: The gradient is calculated and seed points are selected. Then the watershed transform is calculated.





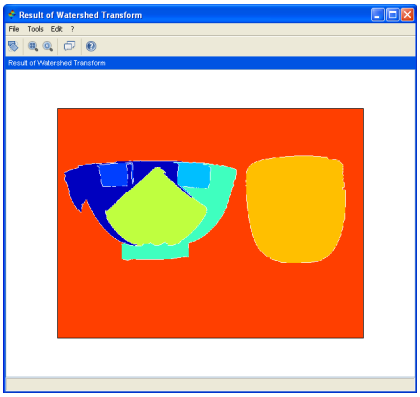


Fig. 3.13 b: Original image. Fig. 3.13 c: Gradient image. Fig. 3.13 d: Result image.

A possible work flow for automated seed point selection is shown in Fig. 3.14.

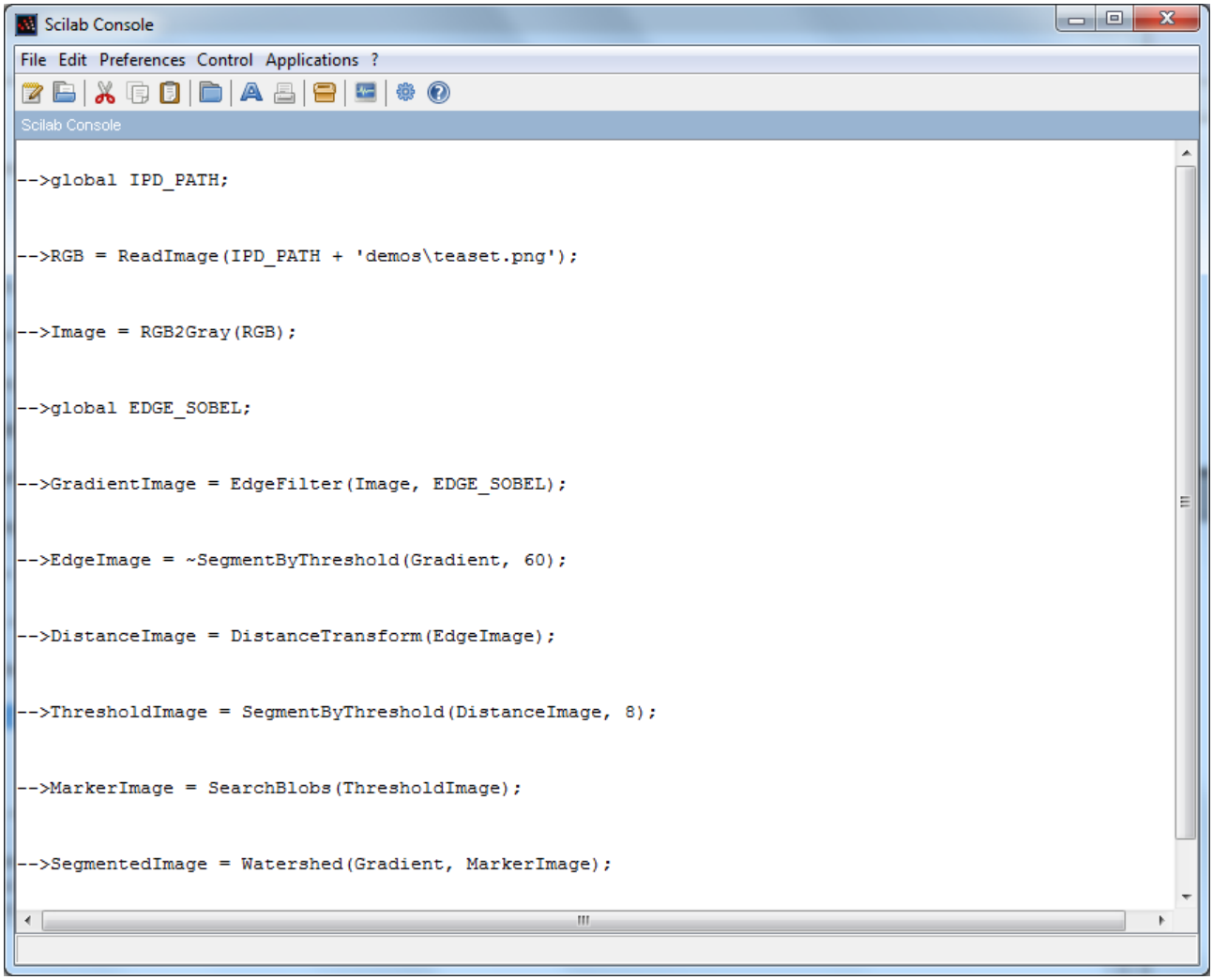


Fig. 3.14 a: Segmentation of an image using distance transform and watershed transform.

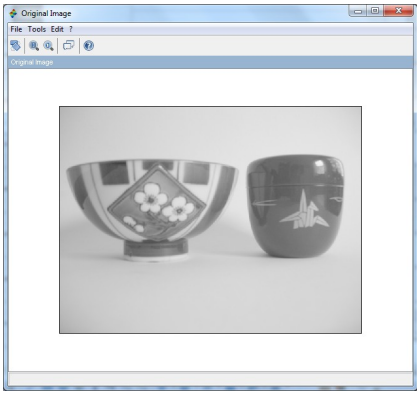


Fig. 3.14 b: Original image.

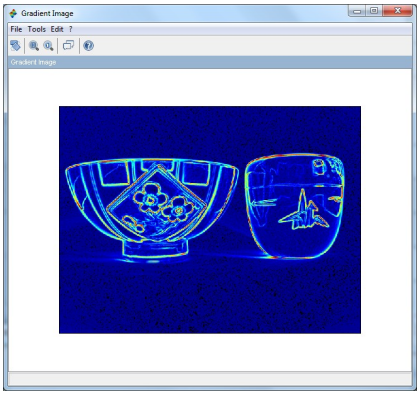


Fig. 3.14 c: Gradient image.

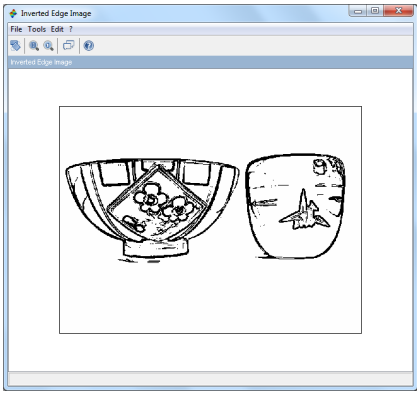


Fig. 3.14 d: Inverted edge image.

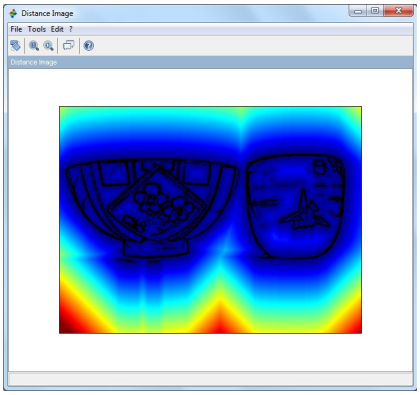


Fig. 3.14 e: Distance Image.

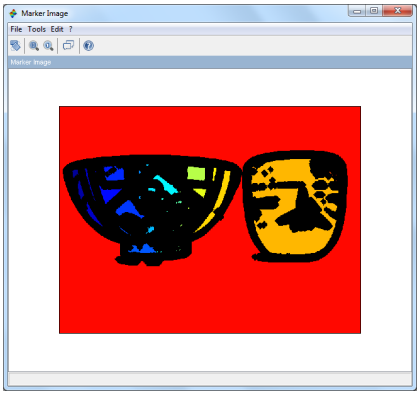


Fig. 3.14 f: Marker image.

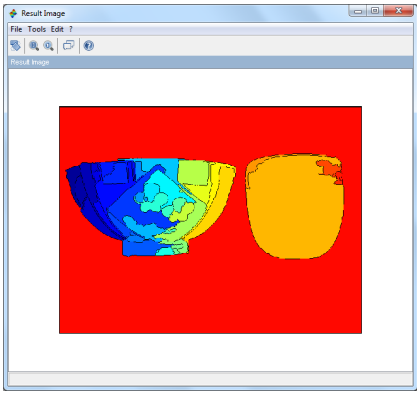


Fig. 3.14 g: Result image.

A gradient image is calculated from the gray level image. The gray level image is thresholded and inverted. In the inverted image, background pixels belong to edges and object pixels are non-edge pixels. The distance transform is calculated so pixels as far as possible from edge pixels are found. The distance transform result is thresholded and blobs in the thresholded distance transform image get markers for watershed transform.

A similar algorithm is described in [3].

4 How to Detect Objects in Images

4.1 Introduction

The goal of object detection is demarcating each object of interest against the background and to represent it by one area that covers it as well as possible. In this chapter two methods to achieve this goal are introduced:

- Thresholding and blob analysis: The image is segmented by a threshold to find areas with gray values typical for the objects. Then the objects are detected using blob analysis. Fragmented objects are connected by morphological filtering and small noise objects are filtered out.
- Edge detection and watershed transform: The edges are detected and seed points for watershed transform are determined. The biggest object after watershed transform is the background so the background is filtered out. Objects that are fragmented are connected by morphological filtering.

After the objects are detected, their bounding boxes are determined and drawn into the original image.

4.2 Thresholding and Blob Analysis

In the image used in the previous chapters, the tea cup on the left has areas darker than the background and the red tea powder can is entirely darker than the background. Therefore, these two objects can be detected by searching these comparatively dark regions.

This can be done the following way: The image is converted from RGB to a gray level image. The gray level image is inverted and segmented using a manually selected threshold. Then the connected areas are searched. This process and its result are shown in Fig. 4.1.

In the original image two objects are visible. However, more than two connected areas are found as can be seen in Fig. 4.1 b. Therefore, all objects except for the two biggest are filtered out. For this purpose, a cumulated histogram of area sizes is calculated. The cumulated histogram value of a specific size shows the percentage of connected areas that have a number of pixels less than or equal to this specific size. This is shown in Fig. 4.2.

The resulting image contains two objects that cover partially the objects in the original image. The bounding boxes of these objects are searched and drawn into the gray level image. This is shown in Fig. 4.3.

As this example shows, it is important to distinguish not only between objects and background, but also between the objects of interest and noise objects when working with thresholding and blob analysis.

It is possible to merge the fragments of an object applying a closing filter to the logical image or the blob image. The two objects to be detected are far from each other so they are still separate as can be seen in Fig. 4.4. However, if the objects to be detected are nearer to each other than the fragments belonging to the same objects, they are merged. Therefore, it is necessary to consi-der distances between objects when merging fragments.

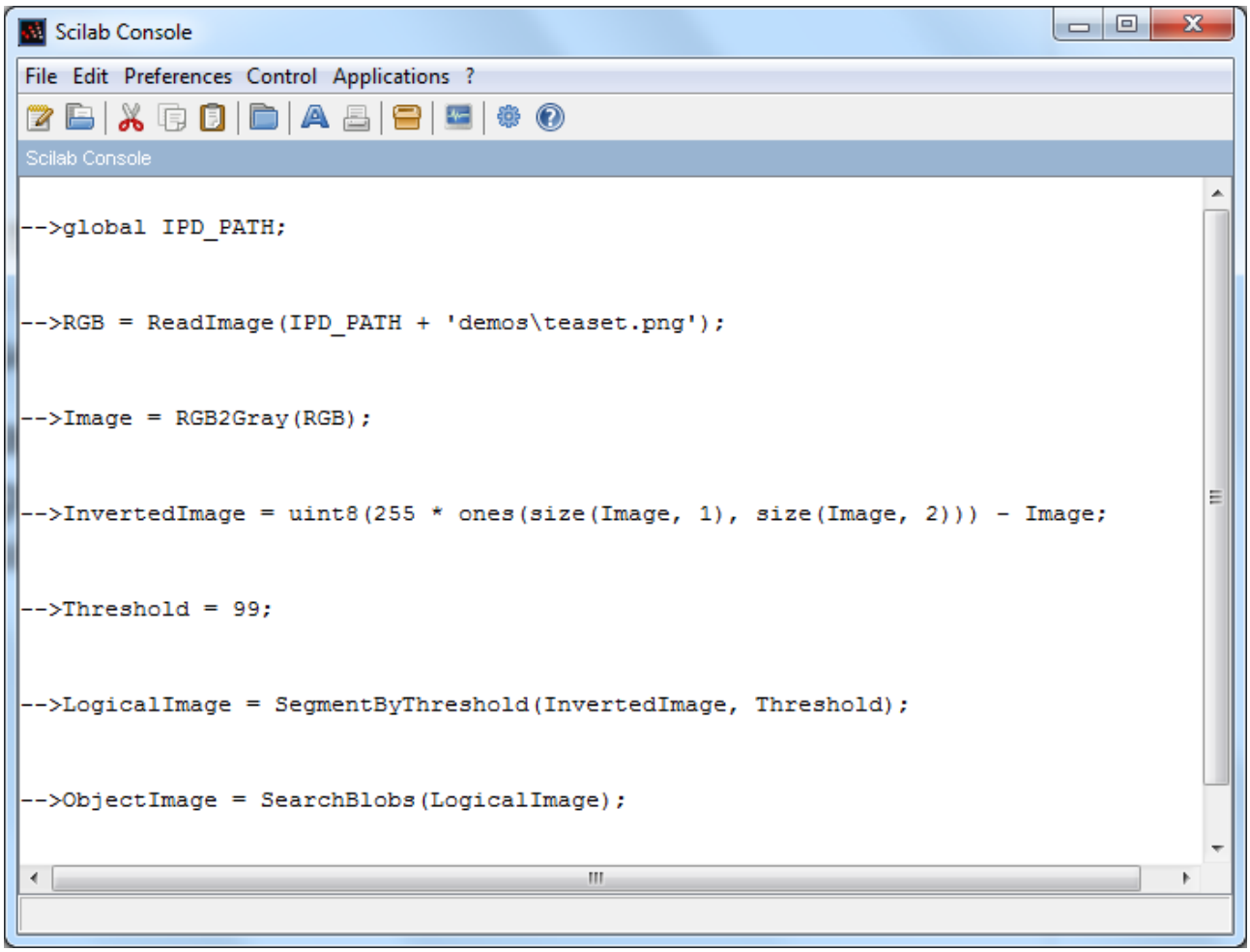


Fig. 4.1 a: The color image is transformed to a gray level image. The gray level image is inverted and segmented using a manually determined threshold. Then the connected areas are searched.

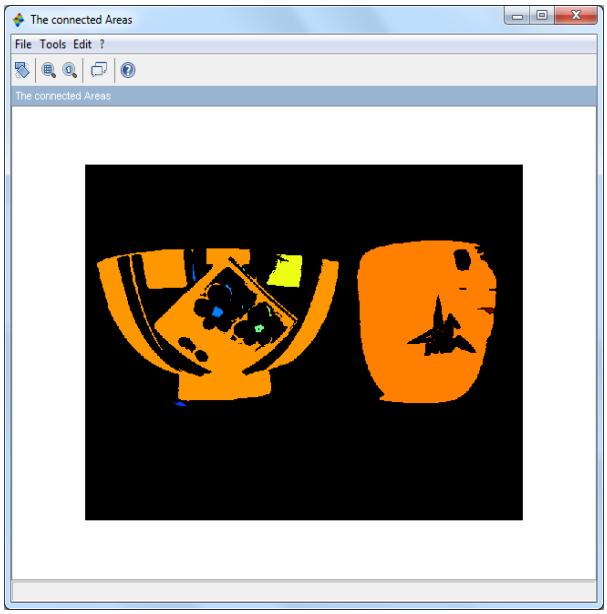


Fig. 4.1 b: The two biggest connected areas correspond to the objects to be detected.

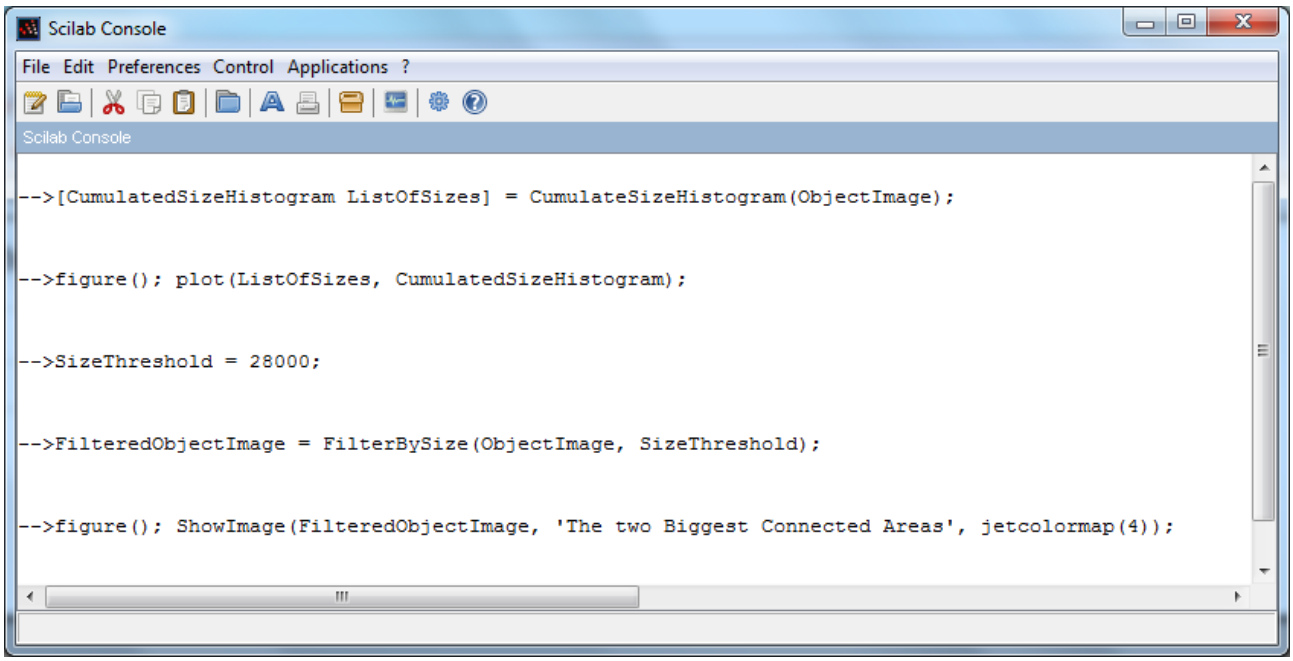


Fig. 4.2 a: The cumulated size histogram of the object image is calculated and a minimum size is determined from it. Connected areas smaller than this minimum size are filtered out.

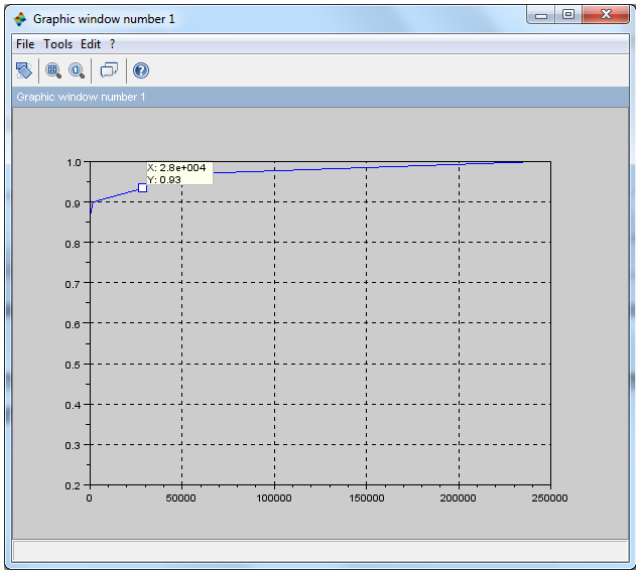


Fig. 4.2 b: The cumulated size histogram of the connected areas.

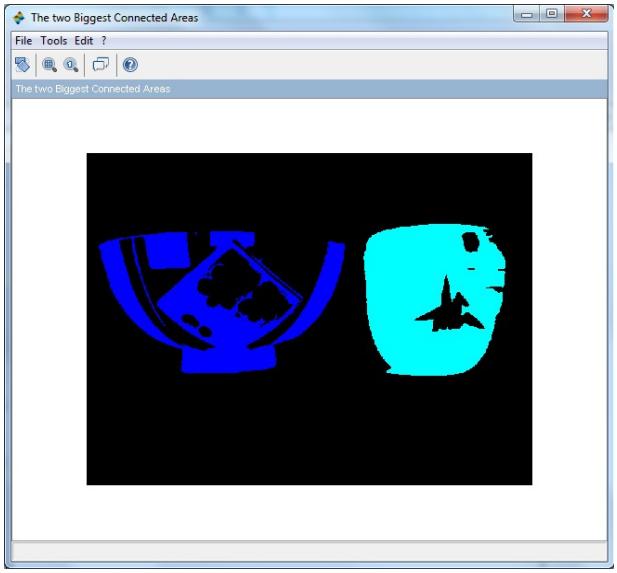


Fig. 4.2 c: All objects except for the two biggest are filtered out.

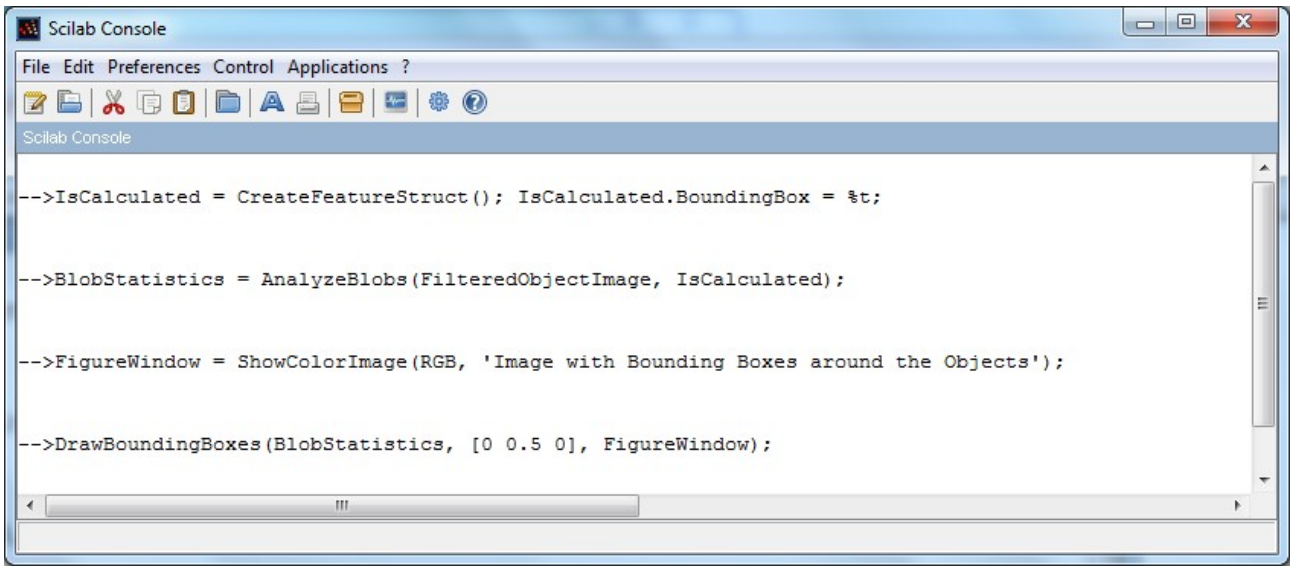


Fig. 4.3 a: The bounding boxes of the two detected objects are determined and drawn into the original image.

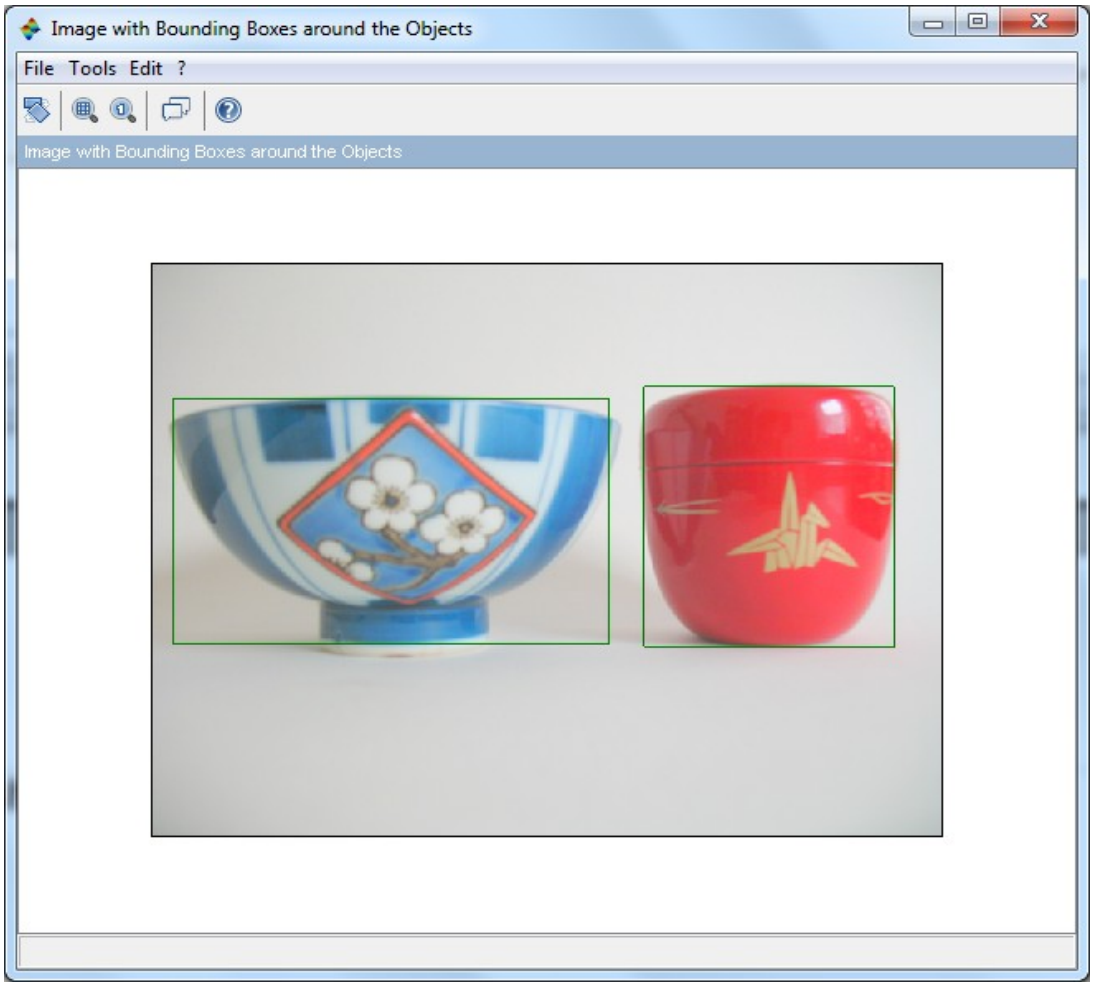


Fig. 4.3 b: The original image with the bounding boxes of the detected objects.

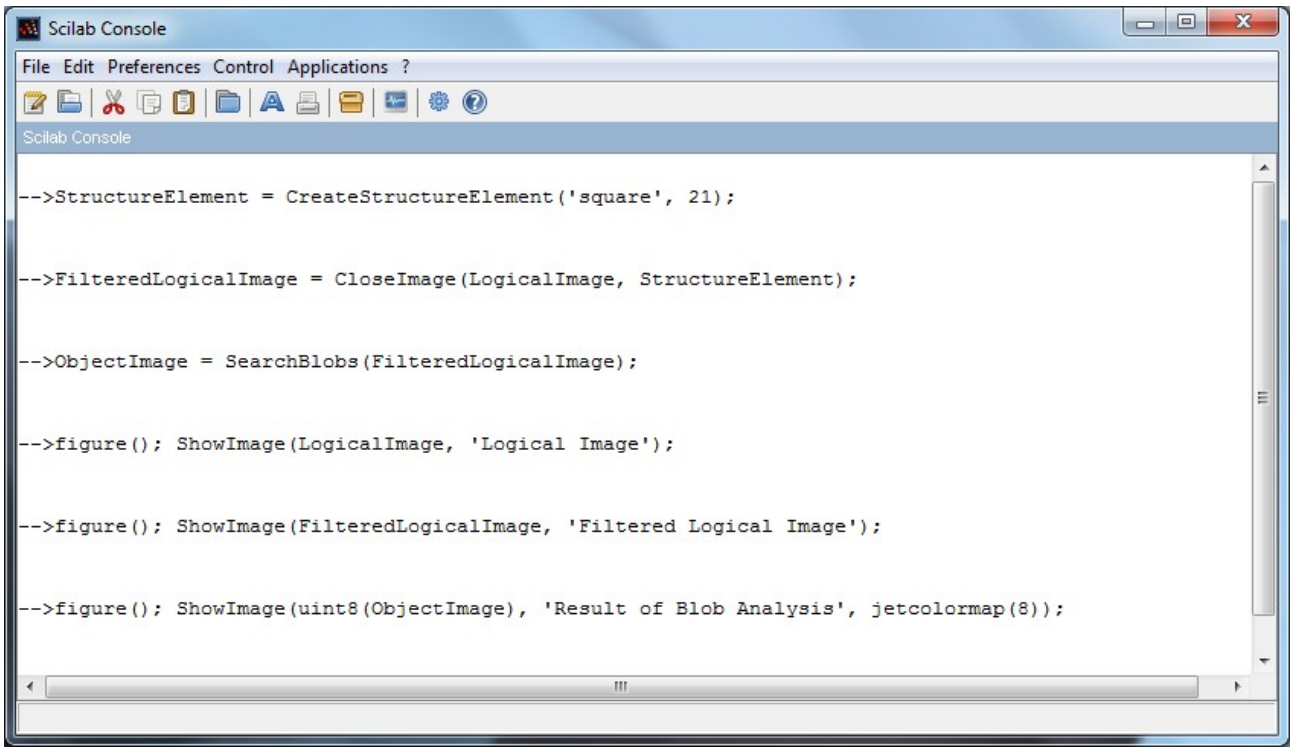


Fig. 4.4 a: The logical image is filtered by a closing filter with a structuring element of 21 * 21 pixels. Then the connected areas in the filtered logical image are searched.

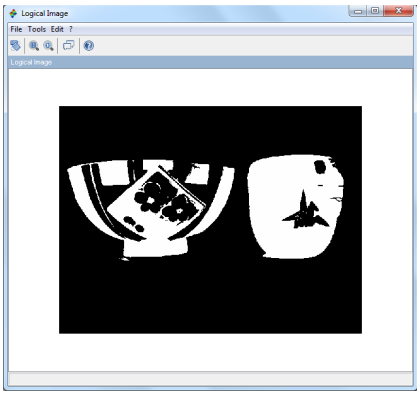


Fig. 4.4 b: Logical image.

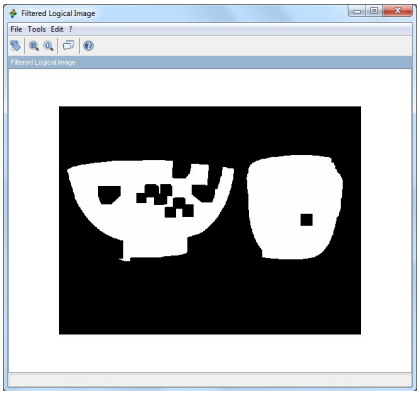


Fig. 4.4 c: Filtered logical image.

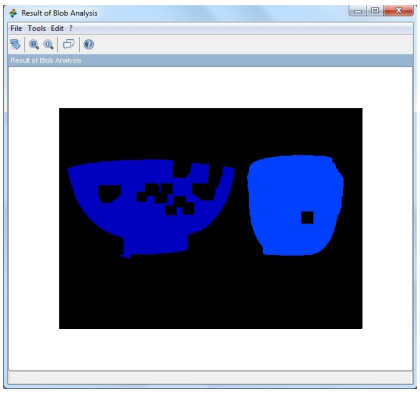


Fig. 4.4 c: Two connected areas are found.

4.3 Edge Detection and Watershed Transform

The objects in the example image can be demarcated against the background by their edges. It is possible to detect the edges, select points within the areas surrounded by edges and apply watershed transform to detect the areas between the edges.

However, there are also edges within the objects. Therefore, it is necessary to select seed points for watershed transform in several areas. The biggest area found then is the background. The background must be filtered out and then the remaining areas can be mer-

ged.

The seeds for watershed transform are selected in the following way: The edge image is inverted and a distance transform is carried out. The result of distance transform is thresholded and the connected areas in the resulting logical image are searched. These areas get the seeds of watershed transform.

The process from edge detection to distance transform is shown in Fig. 4.5.

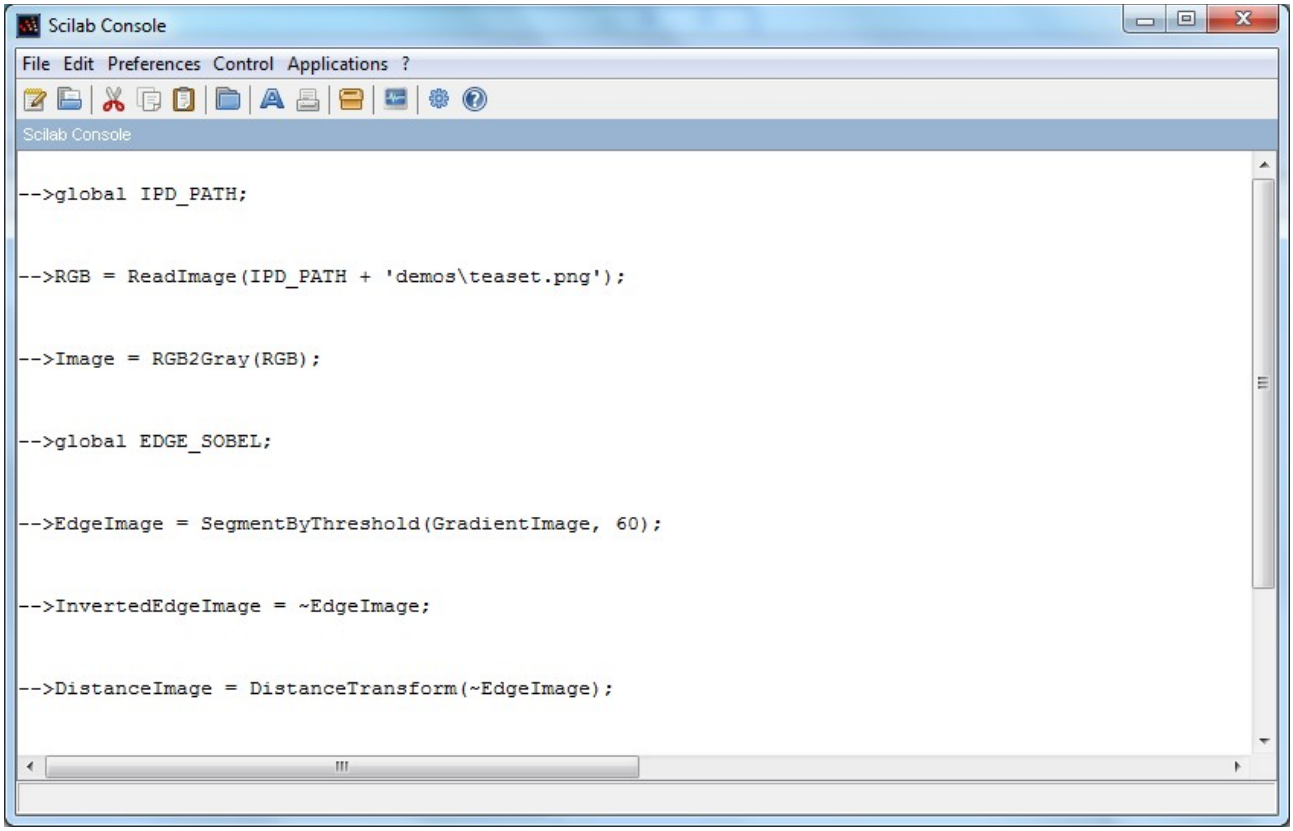


Fig. 4.5 a: The edges are detected, the edge image is inverted and a distance transform is applied to the inverted edge image.

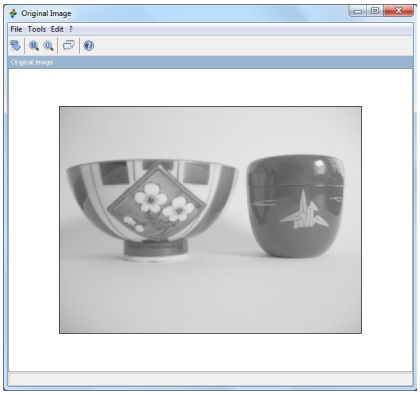


Fig. 4.5 b: Original image.

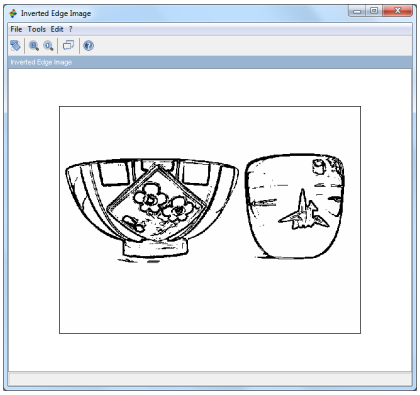


Fig. 4.5 c: Inverted edge image.

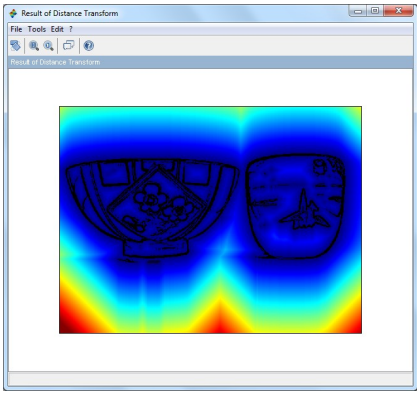


Fig. 4.5 d: Distance transform of inverted edge image

The result of distance transform is thresholded. The threshold is selected manually. Fig. 4.6 shows the result of thresholding the distance transform image and watershed transform.

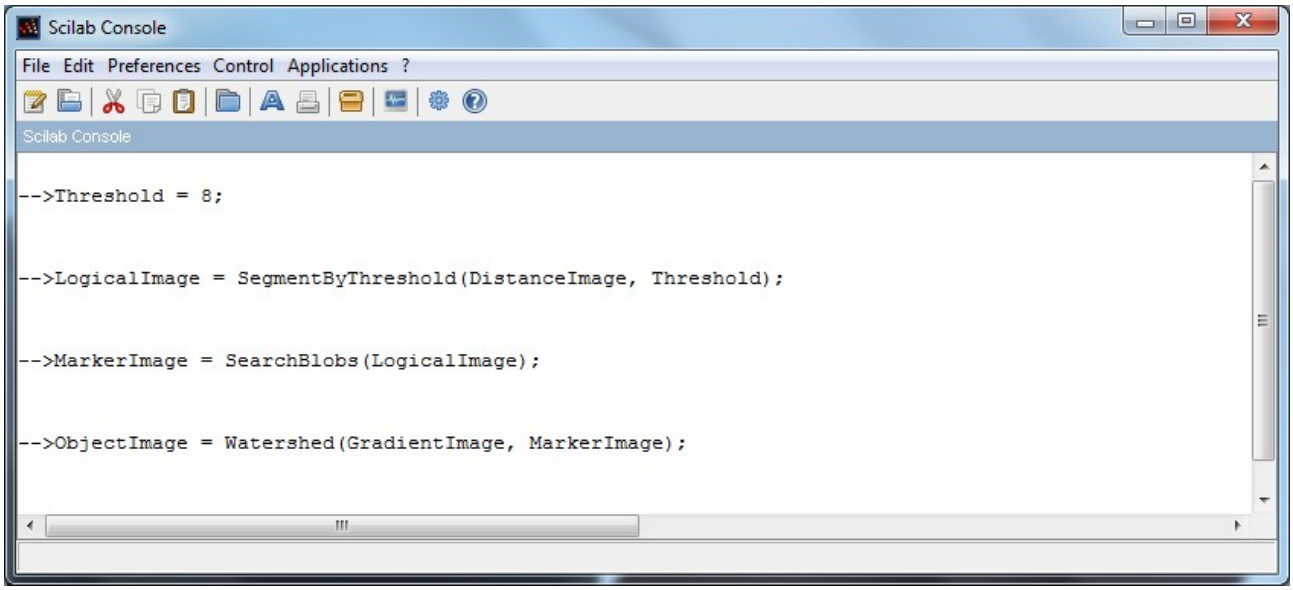


Fig. 4.6 a: The result of distance transform is thresholded and the connected areas are used as markers for watershed transform.

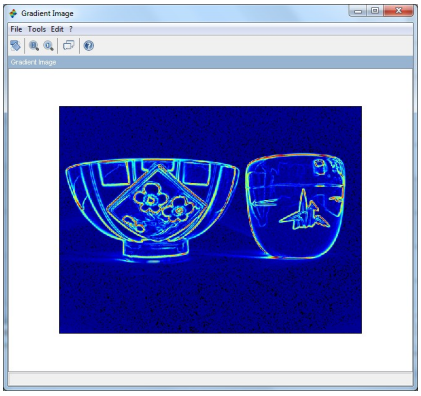


Fig. 4.6 b: Gradient image.

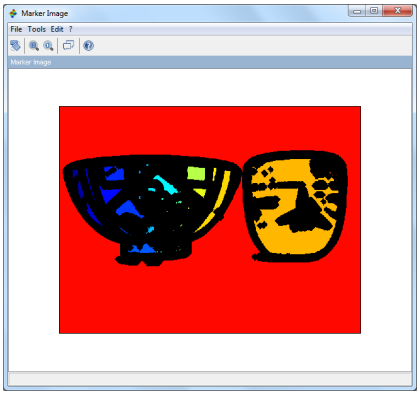


Fig. 4.6 c: Marker Image.

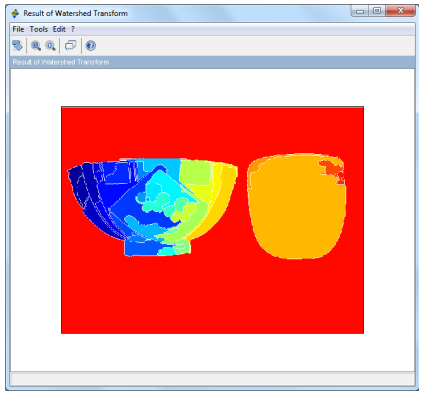


Fig. 4.6 d: Result of watershed transform.

In the image resulting from watershed transform the biggest blob is the background. There are edge pixels between the blobs. The background and edges are set to zero and the image is transformed to a logical image by thresholding with the value one. This logical image is filtered by a closing filter with a 3*3 square as structuring element. The blobs in the resulting image are searched. This process is shown in Fig. 4.7.

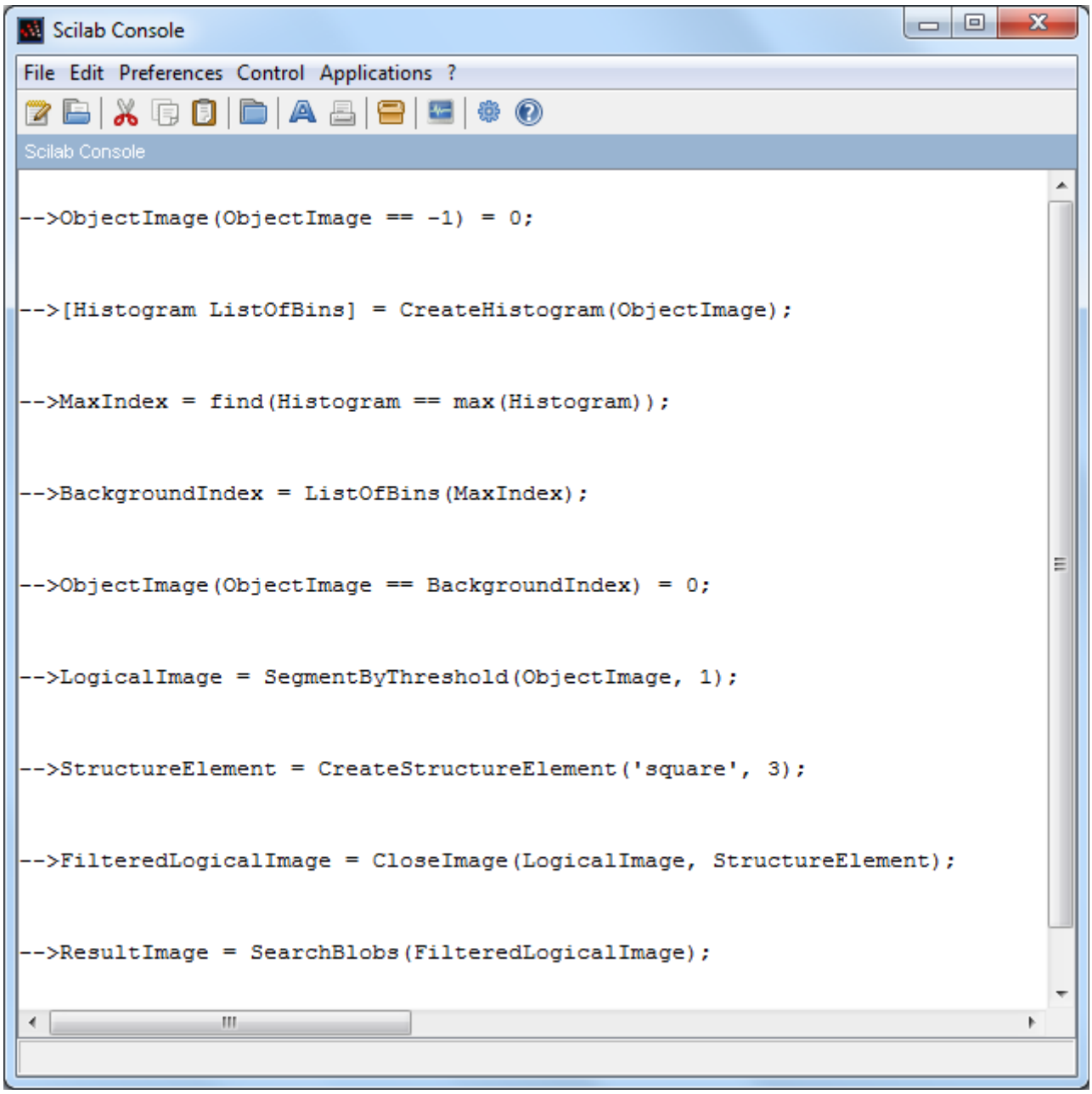


Fig. 4.7 a: The background pixels and edges in the object image are set to zero. Then the image is transformed to a logical image, filtered with a closing filter and transformed back to an object image again. There are exactly two regions corresponding to the objects to be detected.

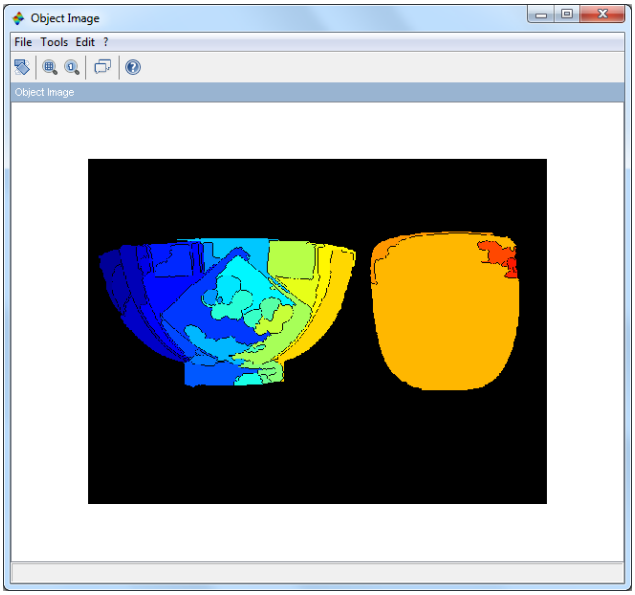


Fig. 4.7 b: Object image after setting edges and background to zero.

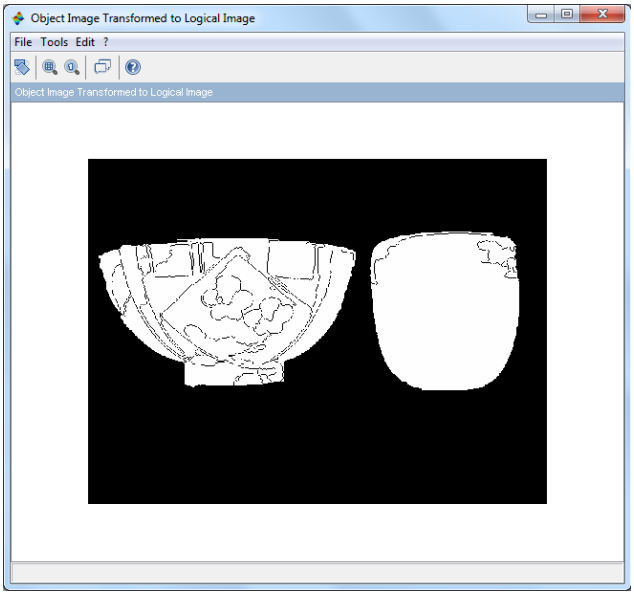


Fig. 4.7 c: Logical image.

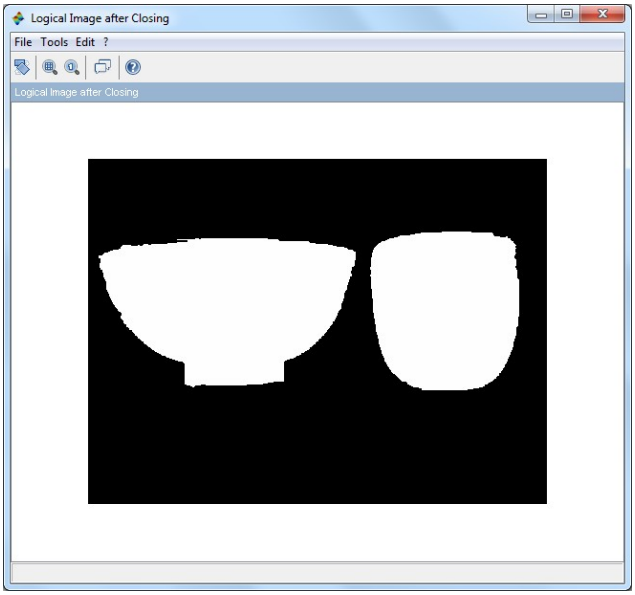


Fig. 4.7 d: Logical image after closing.

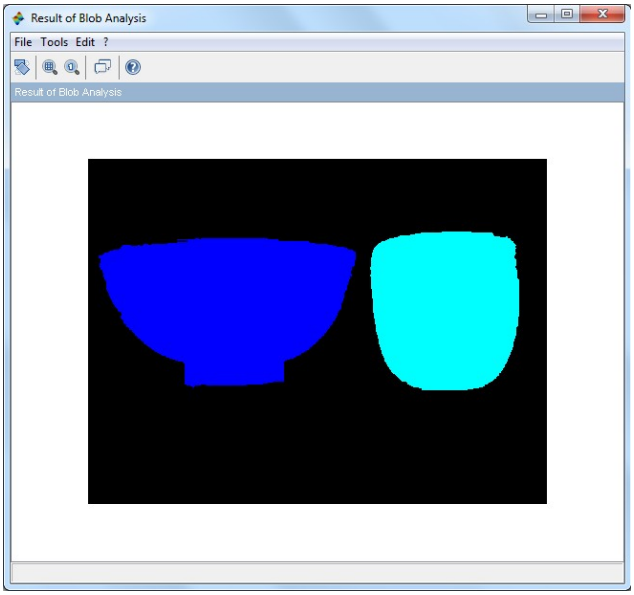


Fig. 4.7 e: Result of blob analysis.

The blobs of the result image cover the images to be detected. The bounding boxes are determined and drawn into the original image (s. Fig. 4.8).

When working with edge detection and watershed transform, it is important to select suitable seed points and to detect and filter out the background in the image resulting from watershed transform.

The distance transform can help to find seed points for watershed transform. However, it is necessary to find a threshold for segmenting the result of distance transform. This requires some knowledge concerning the smallest fragments where seed points have to reside.

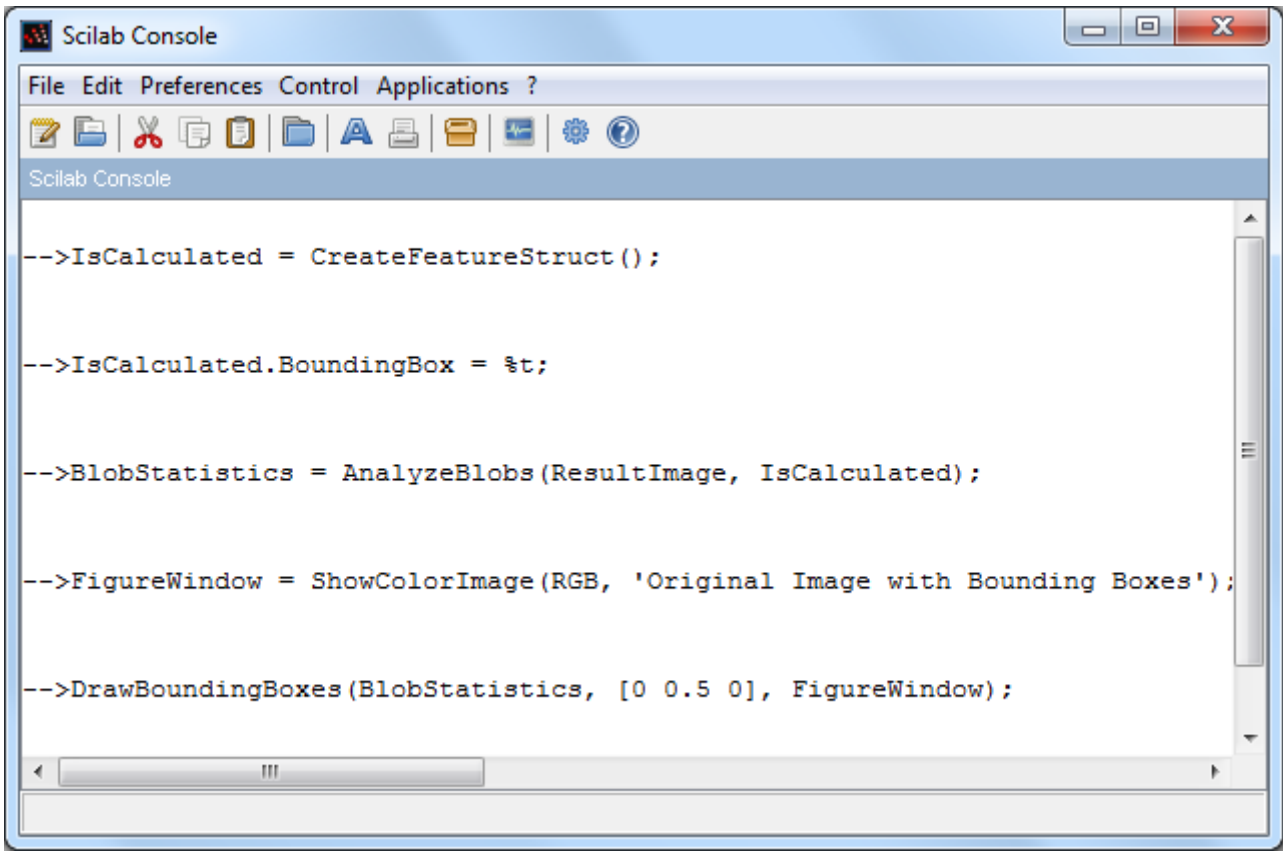


Fig. 4.8 a: Bounding boxes are determined and drawn into the original color image.

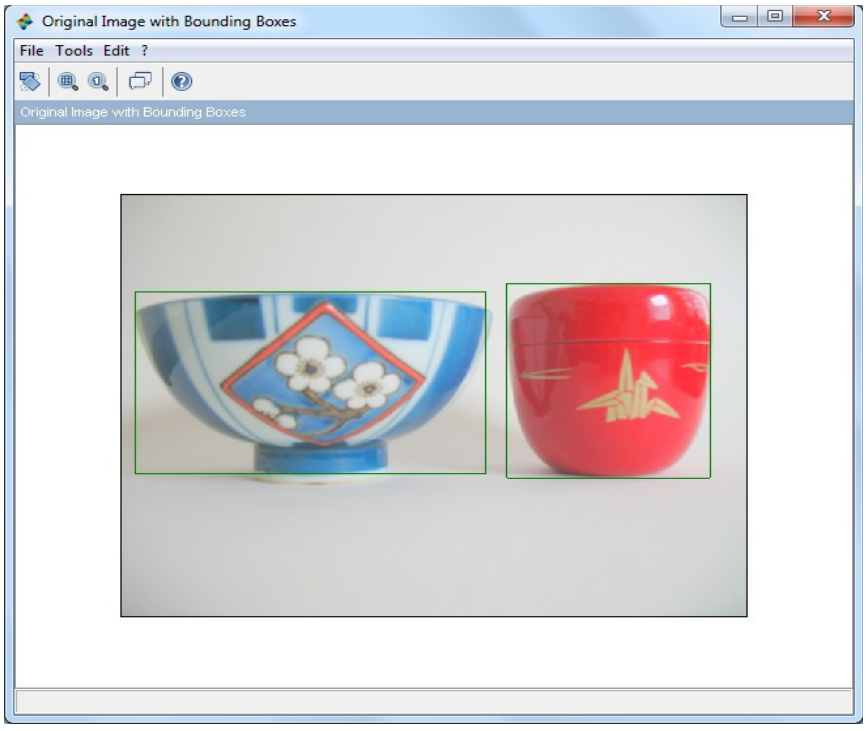


Fig. 4.8 b: The original color image with superimposed bounding boxes.

References

[1] Stephen L. Campbell, Jean-Philippe Chancelier, Ramine Nikoukhah, *Modeling and Simulation in Scilab/Scicos*, Springer, New York, 2006

[2] Harald Galda, *Development of a segmentation method for dermoscopic images based on color clustering*, Kobe University, August 2003

[3] Rafael C. Gonzales, Richard E. Woods, Steven L. Eddins, *Digital Image Processing Using MATLAB*, Pearson Prentice Hall, Upper Saddle River, New Jersey, 2004

[4] Linda G. Shapiro, George C. Stockman, *Computer Vision*, Prentice-Hall Inc., Upper Saddle River, New Jersey, 2001