

# Screening Task: Scilab Image Processing Toolbox Development

# On

# Scilab Image Processing Toolbox development

# Submitted by

# Anuska Rani

# Jan 26, 2025

Acknowledgement

I extend my heartfelt gratitude to FOSSEE and [contact@scilab.in](mailto:contact@scilab.in) for providing me with this opportunity, which has greatly enriched my learning experience.

Lastly, I thank my batchmates ,seniors ,family and academic advisors for their constant support and motivation, making this journey truly rewarding.

Thank you all for being part of this important phase of my growth.

# **Scilab Image Processing Toolbox Development**

## 

| **Table of contents** |
| --- |

## 

1. I**ntroduction** ............................................................................... **4**
2. **Selected Functions Development and Implementation**……………**6**

Function 1: bwarea ....................................................................6 Function 2: regionprops.............................................................. 8 Function 3: greythresh ........................................................... 10

1. **Testing and Results** ...................................................................... **13**
2. **Learnings** ..................................................................................... **14**
3. **Conclusion** ................................................................................... **14**
4. **References** .................................................................................... **14**

**1.Introduction**

Scilab is a powerful open-source software for numerical computation and high-level programming. It serves as an alternative to proprietary tools like MATLAB and Octave, offering capabilities in signal processing, image processing, optimization, and more. Among these, the Image Processing Toolbox is crucial for tasks like feature extraction, image analysis, and transformation.

This project focuses on enhancing Scilab’s Image Processing Toolbox by implementing three important functions: BWAREA, REGIONPROPS, and GREYTHRESH. These functions are inspired by their counterparts in GNU Octave's ‘image’ package, which provides robust image processing functionalities.

The objective of this task is to:

- Understand the implementation of the chosen functions in Octave.

- Develop equivalent functions in Scilab, ensuring similar functionality and accuracy.

- Create documentation and test cases to verify the reliability of these implementations.

This report outlines the development process, challenges encountered, and solutions implemented for each function. It also highlights the learning outcomes from the project and its potential impact on Scilab's usability in image processing applications.

Image processing is a crucial aspect of various scientific and industrial fields, including medical imaging, robotics, and remote sensing. The Scilab Image Processing Toolbox Development project focuses on replicating essential image processing functionalities available in Octave's image package.

The primary objective of this project is to implement and test three key image processing functions in Scilab:

| **Function** | **Purpose** | **File Name** | **Links** |
| --- | --- | --- | --- |
| bwarea | Calculates area of connected components in a binary image. | bwarea.sci | [bwarea (1).sci](https://drive.google.com/file/d/1bfTT4DGza3mfvwZcnp2KYy62clPG4SAC/view?usp=drive_link)  [README.txt for bwarea.sci](https://docs.google.com/document/u/0/d/1GuVmlj13IGw-nhrQIZb4hSJz3ExyAg93-J5tS90xuKg/edit) |
| greythresh | Computes a global threshold using Otsu's method. | greythresh.sci | [Graythresh.sci](https://drive.google.com/file/d/1jf8mTXip7zX-jyAjK8fa-2KXfhOQ4fS2/view?usp=drive_link)  [README.txt for graythresh.sci](https://docs.google.com/document/u/0/d/1Hpab42X_ta0oKMoQQdSI1rpXC-jpwUC9U6j6FkjNvyo/edit) |
| regionprops | Measures properties of image regions | regionprops.sci | [Regionprops.sci](https://drive.google.com/file/d/1UJEvo2FBTa2D-OazgpRMhY-DbS2hafRU/view?usp=drive_link)  [README.txt for regionprops.sci](https://docs.google.com/document/u/0/d/1r410B_o_V9CQDGmgPQqHPCYegXnyXqTmtFbKMEBwSAU/edit) |

Folder\_Link:[SIVP\_0.5.3.2](https://drive.google.com/drive/folders/1DWMRm3MEBsG3UKLcM5rwB7-EQ6R9bZku?usp=drive_link)

**2. Selected Functions Development and Implementation**

**BWAREA**

Overview

The bwarea.sci function calculates the area of white pixels (logical 1s) in a binary image. It applies a weighted method to account for pixels on the image boundaries. This is particularly useful in scenarios where precision is needed for measuring objects in binary images.

**Implementation**

The Scilab implementation of follows these step**s:**

1. Accept a binary matrix as input.

2, Use Scilab’s built-in operations to count white pixels.

3.Apply weights to account for boundary adjustments, as done in Octave:

* Full pixels: Weight = 1
* Pixels on edges: Weight = 0.5
* Corner pixels: Weight = 0.25

**Challenges:**

* Octave handles pixel connectivity (4 or 8) using built-in logic, which had to be implemented manually in Scilab.
* Used Scilab’s logical indexing and matrix operations for boundary checks.

**Path(my\_directry\_used) : C:\Users\91799\use\_it\bwarea.sci**

#### **Code**

function area = bwarea(binaryImage)

[labeledImage, numRegions] = bwlabel(binaryImage);

area = zeros(1, numRegions);

for i = 1:numRegions

area(i) = sum(labeledImage == i);

end

endfunction

#### **Explanation**

1. The binary image is labeled using bwlabel, assigning unique labels to each connected component.
2. For each labeled region, the total number of pixels is calculated and stored in an array area.

**Test Case:**

BW = [1 0 1; 0 1 0; 1 1 0];

area = bwarea(BW);

// Expected Output: area = [1, 3]

**Test Case Table :BWAREA**

| **Input (Binary Image)** | **Output (Area)** | **Notes** |
| --- | --- | --- |
| [1 0 0; 1 1 0; 0 1 1] | 5 | Diagonal line of 1s, simple pattern. |
| [0 0 0; 0 0 0; 0 0 0] | 0 | Edge case: all black pixels. |
| [1 1 1; 1 1 1; 1 1 1] | 9 | Full image of 1s, uniform region. |
| [1 0 1; 0 1 0; 1 0 1] | 5 | Symmetrical pattern of 1s, non-adjacent. |
| [1] | 1 | Minimal input: single white pixel. |

**REGIONPROPS**

Overview

The regionprops function calculates properties of labeled regions in an image, such as area, centroid, and bounding box. This is essential for feature extraction in image segmentation tasks.

**Implementation**

1. Use bwlabel to label connected regions in a binary image.
2. For each labeled region:
   * Area: Count pixels in the region.
   * Centroid: Compute the mean of row and column indices.
   * Bounding Box: Identify the minimum and maximum row and column indices.
3. Return a structured array containing the calculated properties.

**Challenges:**

* Reproducing Octave’s property-based indexing using Scilab’s structs.
* Handling edge cases with overlapping or small regions.

#### 

#### **Code**

function props = regionprops(binaryImage)

if typeof(binaryImage) ~= "boolean" then

error("Input must be a binary image (boolean matrix).");

end

[labeledImage, numComponents] = bwlabel(binaryImage);

props = [];

for i = 1:numComponents

component = labeledImage == i;

[rows, cols] = find(component);

area = length(rows);

centroid = [mean(rows), mean(cols)];

boundingBox = [min(rows), min(cols), max(rows) - min(rows), max(cols) - min(cols)];

props($+1).Area = area;

props($).Centroid = centroid;

props($).BoundingBox = boundingBox;

end

endfunction

**Explanation**

1. The binary image is labeled using bwlabel to assign unique labels to connected components.

2.For each labeled region, the area, centroid, and bounding box are calculated and stored in a structured array props.

**TESTCASE:**

BW = [1 1 0; 0 1 1; 1 0 0];

props = regionprops(BW);

// Expected Output:

// props(1).Area = 3

// props(1).Centroid = [2, 2]

// props(1).BoundingBox = [1, 1, 3, 3]

**Test Case Table :**

| **Input (Binary Image)** | **Property** | **Output (Result)** | **Notes** |
| --- | --- | --- | --- |
| [1 0 0; 1 1 0; 0 1 1] | Area | **5** | Tests area of a simple connected region. |
| [1 0 0; 1 1 0; 0 1 1] | Centroid | [2.2, 1.4] | Centroid calculation for an irregular region. |
| [1 0 0; 1 1 0; 0 1 1] | BoundingBox | [1, 1, 3, 2] | Bounding box for the irregular region. |
| [0 0 0; 0 0 0; 0 0 0] | Area | 0 | Edge case: no regions (all 0s). |
| [1 1 1; 1 0 1; 1 1 1] | BoundingBox | [1, 1, 3, 3] | Full image region with hollow center. |

**GREYTHRESH**

Overview

The greythresh function calculates a global threshold for converting a grayscale image to binary using Otsu’s method. It minimizes intra-class variance and is widely used in image segmentation.

**Implementation**

1. Calculate the histogram of the input grayscale image using Scilab’s histplot.
2. Implement Otsu’s method:
   * Compute class probabilities and means.
   * Calculate between-class variance for each threshold.
   * Select the threshold with the maximum between-class variance.
3. Return the optimal threshold.

Challenges:

* Scilab does not have a direct implementation of Otsu’s method, requiring manual computation.
* Ensuring numerical stability when calculating variances for large images

**Code**

function threshold = graythresh(image)

if size(image, 3) == 3 then

error("Input must be a grayscale image.");

end

[counts, bin] = hist(image(:), 256);

total = sum(counts);

sumB = 0; wB = 0; maxVariance = 0;

sumTotal = sum(bin .\* counts);

for t = 1:256

wB = wB + counts(t);

wF = total - wB;

if wB == 0 || wF == 0 then

continue;

end

sumB = sumB + bin(t) \* counts(t);

mB = sumB / wB;

mF = (sumTotal - sumB) / wF;

variance = wB \* wF \* (mB - mF)^2;

if variance > maxVariance then

maxVariance = variance;

threshold = t - 1;

end

end

threshold = threshold / 255;

endfunction

#### **Explanation**

1. The histogram of the image is computed with 256 bins.
2. Between-class variance is calculated for each possible threshold value.
3. The threshold that maximizes the variance is selected as the optimal value.

#### **Test Cases**

I = [50 50 200; 50 50 200; 200 200 200];

threshold = graythresh(I);

// Expected Output: threshold = 0.3922 (approximately)

| **Test Case Table:GREYTHRESH** |
| --- |

| **Input (Greyscale Image)** | **Output (Threshold)** | **Notes** |
| --- | --- | --- |
| [100 150 200; 50 75 125; 25 100 175] | 100 | Tests threshold with varied intensity. |
| [0 0 0; 0 0 0; 0 0 0] | 0 | Edge case: completely dark image. |
| [255 255 255; 255 255 255; 255 255 255] | 255 | Edge case: completely bright image. |
| [100 120 140; 110 130 150; 105 125 145] | 125 | Gradual intensity increase. |
| [0 255 0; 255 0 255; 0 255 0] | 128 (approx.) | Alternating bright and dark pattern. |

**Link:** [**for\_fossee\_screening\_task.m**](https://drive.google.com/file/d/1EEoYhqxqfSCJgXrIV2CFA52gEh5JY4YS/view?usp=sharing)

## **3. Testing and Results**

**1.Coding**

* The primary step involved translating the Octave implementations of bwarea, regionprops, and greythresh into Scilab while adhering to its syntax and features.
* Challenges during coding:
  + Adapting Octave’s functions like find and cumsum into Scilab equivalents.
  + Ensuring pixel connectivity and weighted adjustments for bwarea.
  + Handling edge cases for regionprops (e.g., single-pixel regions or overlapping areas).
  + Implementing Otsu’s method manually for greythresh as Scilab lacks a built-in equivalent.

**Documentation**

* For each function, a detailed README file was created, which includes:
  + A concise description of the function.
  + Input and output parameters with their types and roles.
  + A clear calling sequence.
  + Multiple test cases for verification.

**Testing**

* Testing was done using Scilab’s built-in assertion functions:
  + assert\_checkequal to validate exact outputs.
  + assert\_checkalmostequal to check numerical tolerances (useful for greythresh ).
* Edge cases were prioritized, such as empty or uniform images and irregular regions.
* The output of Scilab functions was compared against Octave results for consistency.

**Link:** [**Using Scilab for Testing**](https://docs.google.com/document/u/0/d/1udGUgl0vtxeeG2mHi1U9l8pXuG3WOWPWDgfAjmHLVSE/edit)

**Link:** [**first.png**](https://drive.google.com/file/d/1GV0_n_usV87LgTekf8IjEnkRuH0jhG7i/view?usp=drive_link)**,** [**second.png**](https://drive.google.com/file/d/1ycAMUC_c2N1V9y4AH4MxME-X4BFvMZYi/view?usp=drive_link)**,** [**third.png**](https://drive.google.com/file/d/1WPr7F7OSexMKaMz8RBXelZtA1OoUqGFz/view?usp=drive_link)**,**[**fourth.png**](https://drive.google.com/file/d/13y0qdPySg3Z1UegPqMkOdmsSvZNTPtNf/view?usp=drive_link)**,**[**fifth.png**](https://drive.google.com/file/d/15pwPTEHjwd7KQBA_TekFvTGx-35Hn0FB/view?usp=drive_link)

## **4. Learnings**

Through this project, I gained:

1. A deeper understanding of Scilab's syntax and capabilities.
2. Knowledge of image processing concepts like thresholding, region labeling, and property extraction.
3. Problem-solving skills by debugging and testing functions.

## **5. Conclusion**

The Scilab implementations of bwarea, greythresh, and regionprops successfully replicate the functionalities of their Octave counterparts. These functions are essential tools in the image processing workflow and have been tested to ensure reliability.

## **6. References**

1. Octave Image Package Documentation:<https://octave.sourceforge.io/image/>
2. Scilab Official Documentation:<https://help.scilab.org/>

**THANK YOU**