Machine Vision

Image Segmentation and Thresholding in MATLAB

Lab Activity Sheet 4

Author: Dr. Mostapha Kalami Heris

Introduction

Why this lab matters

Segmentation converts continuous pixel intensities into separate regions that represent different objects or materials in an image. This step is essential because almost every stage after it in a vision pipeline—such as feature extraction, measurement, and classification—depends on how well the image is segmented. In this lab, you will learn when and how to use different threshold-based methods and how image properties such as illumination and histogram shape affect your choice of method. You will also see situations where simple methods are not enough and how advanced approaches such as multilevel thresholds or watershed segmentation can help.

What you will learn today

By the end of this lab you should be able to:

1. Apply and compare **global thresholds** (Otsu) using `graythresh`, `otsuthresh`, and `imbinarize`.
2. Handle non-uniform illumination using **adaptive thresholding** with `adaptthresh`.
3. Segment images with multiple intensity levels using **multilevel thresholding** with `multithresh` and `imquantize`.
4. Recognize common segmentation issues such as uneven lighting, noise, or touching objects and suggest suitable improvements.
5. (Optional, advanced) Explore **watershed segmentation** for separating connected or overlapping regions.

Concepts and MATLAB functions used in this lab

Global thresholding (Otsu). Use `graythresh` or `otsuthresh` to calculate a threshold value automatically and convert the image to binary with `imbinarize`.

Adaptive or local thresholding. Use `adaptthresh` to compute a locally varying threshold based on neighborhood statistics and apply it with `imbinarize`.

Multilevel thresholding. Use `multithresh` to find several thresholds and `imquantize` to divide an image into multiple intensity levels. Visualize the segmented image with `label2rgb`.

Watershed (advanced). Use distance transforms and minima suppression to separate touching objects. The output is a label matrix that identifies each distinct region.

Dataset and setup

Use MATLAB's built-in images such as `coins.png`, `rice.png`, `peppers.png`, and `foggysf2.jpg`. You are also encouraged to bring one of your own images that has uneven lighting or overlapping objects to test your understanding. Reusing your helper functions from earlier labs for displaying results side-by-side will help with comparison.

Safety checklist before activities

- Convert images to `double` before mathematical operations and keep track of whether pixel values are in the range $[0, 1]$ or $[0, 255]$.
- When comparing methods, clearly note the reason for your parameter choices and how the image characteristics influence them.
- For multilevel segmentation, remember that the result is a *label map*, not a binary image. Use `label2rgb` only for visualization.
- When separating touching objects, start with thresholding and basic clean-up before using watershed to avoid oversegmentation.

Learning Outcomes and Activities Overview

Purpose of this lab

This lab will help you understand how segmentation converts image intensity information into regions that can be analyzed and measured. You will experiment with several thresholding techniques and evaluate how they behave under different lighting conditions and image types. The exercises are designed to strengthen both your theoretical understanding and your practical MATLAB skills.

Learning outcomes

After completing this lab, you should be able to:

1. Explain the concept of image segmentation and its role in the computer vision pipeline.
2. Apply and compare global, adaptive, and multilevel thresholding methods in MATLAB.
3. Interpret image histograms and decide when global or adaptive thresholding is appropriate.
4. Implement preprocessing or post-processing techniques to improve segmentation results.
5. Apply the watershed transform to separate touching or overlapping objects.
6. Evaluate and document segmentation quality using visual inspection and logical reasoning.

How to study and explore

- Begin with the provided sample images to understand each concept clearly.
- Once comfortable, test your methods on one of your own images.
- Change parameters such as sensitivity, number of thresholds, or neighborhood size, and observe how the results change.
- Record your findings and reflect on why each method performs differently on various images.
- Focus on understanding the reasoning behind parameter choices rather than memorizing specific values.

Activity 1: Global and Otsu Thresholding

Objective

The aim of this activity is to understand how global thresholding converts a grayscale image into a binary one by separating foreground and background based on intensity values. You will learn how to choose a suitable threshold manually and how to compute it automatically using Otsu's method.

Key idea

In global thresholding, a single threshold value divides the pixels of an image into two groups:

- Pixels with intensities *above* the threshold are considered foreground.
- Pixels with intensities *below* the threshold are considered background.

Otsu's method automatically determines the threshold that minimizes the intensity variance within each group, which usually corresponds to a well-defined histogram separation.

Step 1: Read and display the image

Start with a grayscale image. You can use the MATLAB built-in image `coins.png`.

```
I = imread('coins.png');
figure;
imshow(I);
title('Original Image');
```

Step 2: Examine the histogram

Plot the image histogram to observe how pixel intensities are distributed.

```
figure;
imhist(I);
title('Histogram of the Image');
xlabel('Intensity Values');
ylabel('Pixel Count');
```

Note: If the histogram shows two distinct peaks, the image is likely suitable for global thresholding.

Step 3: Apply manual thresholds

Choose a few fixed threshold values and observe the results.

```
BW1 = imbinarize(I, 0.3);
BW2 = imbinarize(I, 0.5);
BW3 = imbinarize(I, 0.7);

figure;
montage({BW1, BW2, BW3}, 'Size', [1 3]);
title('Manual Thresholds: 0.3, 0.5, 0.7');
```

Discussion: Which threshold provides a clearer separation between coins and background? What happens when the threshold is too high or too low?

Step 4: Compute Otsu's threshold automatically

Use MATLAB's `graythresh` function to compute an optimal threshold automatically.

```
T = graythresh(I);
BW_otsu = imbinarize(I, T);

figure;
imshow(BW_otsu);
title(['Otsu Thresholding (T = ', num2str(T), ')']);
```

You can also confirm this threshold using histogram counts and the `otsuthresh` function:

```
[counts, x] = imhist(I);
T2 = otsuthresh(counts);
BW_otsu2 = imbinarize(I, T2);
```

Step 5: Compare manual and automatic results

Display the original and thresholded images side by side.

```
figure;
imshowpair(I, BW_otsu, 'montage');
title('Original Image (Left) vs. Otsu Thresholding (Right)');
```

Step 6: Open-ended exploration

- Try another image with uneven illumination (for example, `rice.png`).
- Compare how Otsu's method performs on that image.
- Record your observation about when global thresholding fails.
- Suggest one preprocessing method that might improve the result.

Reflection questions

1. What does Otsu's method try to minimize when calculating the threshold?
2. Under what lighting conditions does global thresholding fail?
3. How could you preprocess an image to make global thresholding more reliable?
4. Why might two different images with similar histograms still require different thresholds?

Summary

In this activity, you learned to apply both manual and automatic global thresholding. You have seen that while Otsu's method is effective for well-lit images with clear intensity separation, it can fail when illumination varies across the scene. This motivates the need for adaptive thresholding, which you will explore in the next activity.

Activity 2: Adaptive Thresholding

Objective

The goal of this activity is to understand how adaptive (or local) thresholding overcomes the limitations of global thresholding. You will learn to apply the `adaptthresh` function in MATLAB and observe how local statistics are used to create a spatially varying threshold map.

Key idea

Global thresholding applies a single intensity value across the whole image. In images with uneven illumination, shadows, or highlights, a single threshold often fails to separate objects correctly. Adaptive thresholding divides the image into smaller regions and determines a threshold for each local neighborhood. This makes it more effective for scenes where lighting or contrast changes across the image.

Step 1: Read and display the image

Use an image where lighting is not uniform, such as `rice.png`.

```
I = imread('rice.png');
figure;
imshow(I);
title('Original Image with Uneven Illumination');
```

Step 2: Compare with global thresholding

To appreciate the difference, first apply Otsu's global threshold from the previous activity.

```
T_global = graythresh(I);
BW_global = imbinarize(I, T_global);

figure;
imshow(BW_global);
title('Global Otsu Thresholding Result');
```

Step 3: Apply adaptive thresholding

Now compute the adaptive threshold and binarize the image.

```
T_local = adaptthresh(I, 0.5);     % Sensitivity = 0.5
BW_adapt = imbinarize(I, T_local);

figure;
imshow(BW_adapt);
title('Adaptive Thresholding (Sensitivity = 0.5)');
```

Explanation: The `adaptthresh` function calculates a locally varying threshold based on neighborhood mean intensity. The `Sensitivity` parameter controls how much the algorithm adapts to local contrast. Higher sensitivity values produce a more aggressive adaptation.

Step 4: Experiment with different sensitivities

Observe how changing sensitivity values alters the segmentation result.

```
BW_low = imbinarize(I, adaptthresh(I, 0.3));
BW_mid = imbinarize(I, adaptthresh(I, 0.5));
BW_high = imbinarize(I, adaptthresh(I, 0.7));

figure;
montage({BW_low, BW_mid, BW_high}, 'Size', [1 3]);
title('Adaptive Thresholding with Sensitivities: 0.3, 0.5, 0.7');
```

Observation: When sensitivity is low, the method behaves more like global thresholding. When it is high, the threshold map follows local brightness more closely.

Step 5: Visualize local threshold map (optional)

```
T_show = adaptthresh(I, 0.5);
figure;
imshow(T_show);
title('Local Threshold Map (Brightness = Local Threshold Value)');
```

This visualization helps you understand how the threshold varies across the image.

Step 6: Compare global and adaptive results

```
figure;
imshowpair(BW_global, BW_adapt, 'montage');
title('Global (Left) vs. Adaptive (Right) Thresholding');
```

Open-ended exploration

- Try adaptive thresholding on another image such as `text.png` or one of your own images.
- Change the sensitivity and window size (using the `'NeighborhoodSize'` option) to study their effects.
- Observe which parts of the image benefit most from adaptive processing.
- Record your findings and describe situations where adaptive methods are clearly superior.

Reflection questions

1. What problem does adaptive thresholding solve compared to global thresholding?
2. How does the sensitivity parameter affect the output?
3. Why might adaptive thresholding produce noisy results on some images?
4. In what situations would you still prefer a global threshold instead of an adaptive one?

Summary

In this activity, you have seen how adaptive thresholding addresses the limitations of global thresholding by computing thresholds that vary across the image. By adjusting parameters such as sensitivity and neighborhood size, you can control how responsive the algorithm is to local intensity changes. This prepares you to move to multilevel thresholding, where multiple thresholds divide the image into more than two regions.

Activity 3: Multilevel Thresholding

Objective

The aim of this activity is to learn how to divide an image into more than two regions using multiple thresholds. You will use MATLAB's `multithresh` function, which extends Otsu's method to find several threshold values that minimize the variance within each region.

Key idea

Multilevel thresholding is useful when the image contains more than two types of regions, such as background, midtones, and highlights. Instead of separating pixels into only foreground and background, the algorithm partitions the intensity range into several classes based on histogram analysis.

Step 1: Read and display an image

Start with an image that has gradual intensity changes, such as `foggysf2.jpg`.

```
I = imread('foggysf2.jpg');
I_gray = rgb2gray(I);

figure;
imshow(I_gray);
title('Original Grayscale Image');
```

Step 2: Compute multiple thresholds

Use `multithresh` to compute two thresholds that divide the intensity range into three levels.

```
thresh = multithresh(I_gray, 2);
disp(thresh);
```

Explanation: If you choose N thresholds, the image will be divided into $N+1$ distinct intensity classes. The returned thresholds represent boundary values that minimize within-class variance, similar to Otsu's single threshold method.

Step 3: Quantize the image into regions

Convert the image into a label map based on the computed thresholds.

```
seg_I = imquantize(I_gray, thresh);
figure;
imshow(seg_I, []);
title('Quantized Image (Label Map)');
```

Each label corresponds to a different region in the image.

Step 4: Visualize the segmented regions

Use color labeling to make the segmentation clearer.

```
RGB = label2rgb(seg_I);
figure;
imshow(RGB);
title('Segmented Image using Multilevel Thresholding');
```

Step 5: Try different numbers of thresholds

Experiment with three or more thresholds and observe how the image segmentation changes.

```
thresh3 = multithresh(I_gray, 3);
seg_I3 = imquantize(I_gray, thresh3);
RGB3 = label2rgb(seg_I3);
figure;
imshow(RGB3);
title('Segmentation with Three Thresholds');
```

Observation: Adding more thresholds divides the image into finer regions, but excessive segmentation can reduce clarity and interpretability.

Step 6: Apply multilevel thresholding to a color image

Try segmenting a color image channel by channel and compare it with full RGB thresholding.

```
I_color = imread('peppers.png');
threshRGB = multithresh(I_color, 5);
seg_RGB = imquantize(I_color, threshRGB);
figure;
imshow(seg_RGB, []);
title('Multilevel Segmentation on Color Image');
```

Note: You can also apply `multithresh` separately to each color plane (R, G, B) to see how results differ per channel.

Open-ended exploration

- Experiment with different numbers of thresholds and record how region details change.
- Compare segmentation results on grayscale versus color images.
- Test the method on one of your own images and note whether the computed thresholds make visual sense.
- Discuss when multilevel thresholding might produce better results than adaptive methods.

Reflection questions

1. How does multilevel thresholding extend Otsu's method?
2. What happens to segmentation quality when too many thresholds are used?
3. Why might different color channels require different thresholds?
4. In which practical applications might multilevel segmentation be more useful than binary thresholding?

Summary

In this activity, you have learned how to apply multilevel thresholding to segment images into multiple regions based on intensity levels. You have seen that `multithresh` and `imquantize` work together to classify pixels into distinct intensity bands. This approach is particularly helpful when an image contains several regions with different brightness or texture levels. In the next optional activity, you will explore the watershed transform, which can separate touching or overlapping objects.

Activity 4: Watershed Segmentation (Optional)

Objective

This optional activity introduces the watershed segmentation algorithm, which is used to separate touching or overlapping objects. You will learn how to use the distance transform and the watershed function in MATLAB to identify individual regions based on object boundaries.

Key idea

The watershed algorithm treats the grayscale image as a topographic surface, where pixel values represent elevation. Bright regions correspond to peaks, and dark regions correspond to valleys. The algorithm "floods" the surface from the minima, and the lines where water from different catchment basins meet form the segmentation boundaries.

Step 1: Read and preprocess the image

Start with an image that contains overlapping objects, such as `coins.png`. Apply noise reduction and convert it to a binary form for processing.

```
I = imread('coins.png');
I_filt = imgaussfilt(I, 2);        % Smooth the image
BW = imbinarize(I_filt, 'adaptive');
BW = imcomplement(BW);             % Invert: objects as basins

figure;
imshow(BW);
title('Preprocessed Binary Image');
```

Step 2: Compute the distance transform

The distance transform assigns each pixel a value representing its distance from the nearest background pixel.

```
D = bwdist(~BW);
figure;
imshow(D, []);
title('Distance Transform');
```

Step 3: Prepare the distance map for watershed

Invert the distance map so that object centers become basins. Use `imhmin` to suppress shallow minima and reduce oversegmentation.

```
D_neg = -D;
D_neg = imhmin(D_neg, 2);   % Suppress shallow minima
```

Step 4: Apply the watershed transform

Apply the watershed function and visualize the boundaries.

```
L = watershed(D_neg);

% Mark boundaries
BW_watershed = BW;
BW_watershed(L == 0) = 0;

figure;
imshow(BW_watershed);
title('Watershed Segmentation Result');
```

Step 5: Color labeling for visualization

Label each segmented region with a unique color for clearer interpretation.

```
RGB_label = label2rgb(L, 'jet', 'w', 'shuffle');
figure;
imshow(RGB_label);
title('Colored Label Map (Watershed Segmentation)');
```

Open-ended exploration

- Experiment with different preprocessing steps (e.g., Gaussian filtering, morphological opening or closing) before applying watershed.
- Adjust the value used in `imhmin` and observe how it affects oversegmentation.
- Try watershed segmentation on a real-world or custom image with touching objects.
- Combine global or adaptive thresholding with watershed to create a hybrid segmentation pipeline.

Reflection questions

1. What is the purpose of the distance transform in watershed segmentation?
2. How does the parameter in `imhmin` influence the final segmentation result?
3. Why can watershed segmentation produce too many regions, and how can you reduce this effect?
4. In what scenarios is watershed segmentation more effective than simple thresholding?

Summary

In this activity, you explored the watershed transform, a powerful technique for separating touching or overlapping objects. You have seen how preprocessing and minima suppression play a crucial role in controlling oversegmentation. The watershed method combines geometric and intensity-based reasoning, making it an important bridge between classical image processing and more advanced segmentation approaches.

Conclusion and Reflection

Summary of key learning points

In this lab, you explored several threshold-based image segmentation techniques, each suitable for different types of images and illumination conditions.

- **Global thresholding** works well when the image has uniform lighting and clear separation between foreground and background.
- **Adaptive thresholding** improves results for images with non-uniform illumination by adjusting thresholds locally.
- **Multilevel thresholding** extends Otsu's idea to divide an image into multiple intensity regions, revealing finer details.
- **Watershed segmentation** helps to separate touching or overlapping objects based on geometric distance information.

Through these activities, you have learned not only how to apply the methods in MATLAB but also how to analyze their performance and limitations. You should now be able to select appropriate segmentation methods based on image characteristics and justify your choice with logical reasoning.

Critical reflection

1. Which segmentation technique gave the most reliable results for your chosen test images, and why?
2. What preprocessing steps were most effective in improving segmentation outcomes?
3. How did parameter tuning (such as sensitivity, number of thresholds, or minima suppression) influence your results?
4. If you were to design a fully automated segmentation pipeline, how would you combine these techniques to handle diverse image conditions?

Further exploration and extension

If you complete the main activities early or wish to continue learning beyond this lab, you may try:

- Combining thresholding and watershed segmentation into a single workflow to handle noisy or overlapping images.
- Comparing segmentation performance between grayscale and color images using different color spaces (e.g., HSV or Lab).
- Applying post-processing operations such as morphological filtering, region labeling, or boundary extraction.
- Investigating how advanced methods such as edge-based segmentation or machine learning can build on the principles explored in this lab.

Final note

Image segmentation is a fundamental task in computer vision, bridging low-level image processing and higher-level analysis. Understanding when and how to use each technique will help you develop robust vision systems and make informed decisions when designing your own image analysis workflows.