Assignment 3: Applications of Image Processing to Real-World Problems

Aleksandr Jan Smoliakov

2024–12–20

1 Introduction

As a final report in the trilogy, this document explores the practical applications of image processing techniques to real-world problems. It builds upon the theoretical foundation established in the previous assignments and demonstrates the application of these concepts to solve specific challenges in diverse domains.

The first section, **Theoretical Background**, provides an overview of the theoretical background, including concepts such as thresholding, morphological operations, and image segmentation.

The subsequent sections present three distinct applications of image processing techniques to real-world problems.

In **Application: FISH Signal Counts**, we analyze fluorescence in situ hybridization (FISH) images to detect and quantify genetic mutations in tumor cells. This involves identifying mutations using fluorescent probes and quantifying the signals to understand the mutation distribution within individual cells.

In **Application: Circuit Board Quality Assurance**, we analyze images of circuit boards to detect defects. This involves inspecting soldering regions, traces, and drilled holes using x-ray imaging and image processing techniques.

In **Application: Filled Bottles**, we analyze images of a production line to detect whether bottles are filled to the correct level. This involves detecting the liquid level in bottles using edge detection and intensity analysis.

Contents

2 Theoretical Background

Understanding the real-world applications described in the next sections of this report requires familiarity with several key concepts in digital image processing. This section provides a brief overview of these concepts.

2.1 Thresholding

Thresholding is a fundamental technique in digital image analysis used to separate objects of interest from the background. The process assigns a binary label to each pixel based on a predefined intensity level called the *threshold*. Mathematically, for a grayscale image $f(x, y)$ and a threshold T, the thresholded image $g(x, y)$ can be expressed as:

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) \geq T, \\ 0, & \text{otherwise.} \end{cases}$$

Here, $g(x, y)$ is referred to as a *binary* image, where 1 typically denotes foreground (e.g. object of interest) and 0 denotes background. Depending on the application, the foreground can represent different elements (e.g. cells in a microscopy image or elements in a circuit board).

This technique can be extended to *multiple thresholding* where more than two classes are defined based on different intensity ranges. Mathematically, for n thresholds $T_1, T_2, \ldots, T_n$, the thresholded image $g(x, y)$ can be expressed as:

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) \geq T_n, \\ \vdots & \\ n, & \text{if } T_{n-1} \leq f(x,y) < T_n, \\ 0, & \text{otherwise.} \end{cases}$$

However, in the subsequent sections, we focus on the basic binary thresholding technique for simplicity and clarity.

2.1.1 Basic Global Thresholding

The simplest form of thresholding assumes a single, constant threshold T that partitions the image into two classes (background and foreground). To determine this threshold, one can examine the intensity histogram of the image. A straightforward approach is to choose T as the *midpoint* between the average foreground and average background intensities, or use domain-specific heuristics.

An improved version of basic global thresholding involves iterative selection:

1. Select an initial estimate for the global threshold T.
2. Segment the image into background and foreground using T.
3. Compute the mean intensities μ_b and μ_f of the background and foreground classes, respectively.
4. Update $T \leftarrow \frac{\mu_b + \mu_f}{2}$.
5. Repeat until T converges (i.e. the change ΔT is below a predefined threshold).

Although this method is simple, it often performs adequately if the intensity histogram is bimodal.

2.1.2 Optimal Global Thresholding

The basic global thresholding methods are intuitive but may not be optimal in all scenarios. They can be sensitive to noise, uneven illumination, or complex object-background distributions.

Optimal global thresholding methods aim to find the threshold value T that best separates the classes in a statistical sense. Typically, this involves maximizing a criterion that quantifies class separability. For instance, one may minimize the overall misclassification error probability by modeling the intensities of the background and foreground with probability density functions (PDFs) and solving for the threshold that minimizes the error.

2.1.3 Otsu's Method

Otsu's method is a histogram-based thresholding approach that seeks to maximize the *between-class variance* (or minimize the *within-class variance*) of the thresholded image. Consider an image with an intensity histogram $p(i)$, where i ranges over all possible intensity levels $0 \leq i < L$. For a potential threshold T, define:

$$\omega_0(T) = \sum_{i=0}^{T-1} p(i), \quad \omega_1(T) = \sum_{i=T}^{L-1} p(i),$$

$$\mu_0(T) = \sum_{i=0}^{T-1} i\, p(i)/\omega_0(T), \quad \mu_1(T) = \sum_{i=T}^{L-1} i\, p(i)/\omega_1(T).$$

The total mean intensity is:

$$\mu_T = \sum_{i=0}^{L-1} i\, p(i).$$

The between-class variance $\sigma_B^2(T)$ is:

$$\sigma_B^2(T) = \omega_0(T)\left(\mu_0(T) - \mu_T\right)^2 + \omega_1(T)\left(\mu_1(T) - \mu_T\right)^2.$$

Otsu's method exhaustively searches for the threshold T^* that maximizes $\sigma_B^2(T)$. This method also performs well when the histogram exhibits clear foreground-background separation.

2.1.4 Some Considerations for Thresholding

Thresholding is a popular image segmentation technique because of its simplicity and effectiveness when the foreground and background intensities are well separated. It serves as a preprocessing step for numerous applications, such as object counting in microscopy images, industrial quality control, and document analysis.

By focusing on separating background and foreground, thresholding simplifies more advanced tasks (e.g. morphological operations) and often leads to computational efficiency.

However, as a simple technique, thresholding is sensitive to noise, illumination variations, and complex object-background distributions. For instance, non-uniform lighting can cause regions of the background to appear brighter or darker, leading to incorrect segmentation using a single global threshold. In practice, controlling or normalizing the illumination is often needed. Some common strategies to address these challenges include adaptive thresholding, smoothing, and edge detection.

2.1.5 Improving Global Thresholding

One strategy to improve global thresholding is to *smooth* the image before applying the threshold. Common filters include:

- **Gaussian smoothing**: Convolves the image with a Gaussian kernel to reduce high-frequency noise.
- **Median filtering**: Replaces each pixel with the median of its neighborhood.

By reducing noise, the histogram becomes less distorted, which typically leads to a clearer separation between the foreground and background intensities.

Another approach is to combine thresholding with edge information. Sharp intensity transitions, identified by edge detection filters (e.g. Sobel), mark the boundaries of objects. Integrating these boundaries helps refine the thresholding result:

- **Post-processing**: After global thresholding, apply edge detection and correct misclassified regions near the edges.
- **Pre-processing**: Use an edge map to guide the selection of local or global thresholds (e.g. adjusting T in areas with strong edges).

2.1.6 Variable Thresholding

If the lighting or object intensity is not uniform across the entire image, a single global threshold can be inadequate. One solution is to *partition* the image into subregions and apply a (potentially different) global threshold for each subregion. The subregions can be chosen based on:

- **Regular grids**, i.e. dividing the image into blocks of fixed size.
- **Adaptive segmentation** based on image properties (e.g. intensity gradients).

In this strategy, regions with similar illumination conditions can be thresholded more accurately, improving overall segmentation performance.

In *local thresholding*, the threshold at each pixel depends on local image properties, such as the mean or median intensity of its neighborhood. A common approach is to use a weighted sum of the local mean $\mu_{x,y}$ and standard deviation $\sigma_{x,y}$, where for each pixel (x, y):

$$T_{x,y} = a \cdot \sigma_{x,y} + b \cdot \mu_{x,y}.$$

This allows the threshold to dynamically adapt to changes in illumination across the image. Such methods are especially effective in cases where the intensity varies gradually across the scene.

In summary, thresholding is a cornerstone of image segmentation tasks, and the choice between global and local (adaptive) thresholding depends heavily on the application's requirements and the nature of the illumination. Methods like Otsu's enable automatic selection of a global threshold, while local thresholding approaches offer robust solutions under challenging illumination conditions or non-uniform backgrounds.

2.2 Mathematical Morphology

Mathematical morphology is a branch of image processing that focuses on the analysis and manipulation of structures within an image using concepts from set theory. A binary image A is viewed as a set of pixel coordinates in 2D integer space $\mathbb{Z}^2$ for which the corresponding pixels are 1 (foreground) or 0 (background). Morphological operations probe this image with a simpler shape or pattern $B \subset \mathbb{Z}^2$ called a *structuring element*. By examining how A interacts with B, morphological operations can expand or shrink objects, remove noise, fill holes, and extract relevant features.

2.2.1 Sets and Set Operations

Since images and structuring elements are treated as sets, we will rely on basic set operations:

- *Union* ($\cup$): The union of two sets A and B is the set of points in either A or B.
- *Intersection* ($\cap$): The intersection of two sets A and B is the set of points that are in both A and B.
- *Complement* (A^c): The complement of A contains all points not in A.
- *Difference* ($\setminus$): The difference of two sets $A \setminus C$ contains points in A but not in B.

2.2.2 Structuring Elements

A structuring element B is typically a small, simple set (e.g., a 3×3 square) that probes the image for the morphological operation. In two-dimensional images, we often represent B by its *origin* (the reference point, frequently in the center). For any point in 2D integer space $(x, y) \in \mathbb{Z}^2$, the translation of B by (x, y) is denoted

$$B_{(x,y)} = \{(b_x + x, b_y + y) : (b_x, b_y) \in B\}.$$

The choice of structuring element depends on the desired operation. Smaller structuring elements provide fine details, whereas larger ones capture coarse structures.

2.2.3 Morphological Erosion and Dilation

Erosion shrinks the foreground set by removing boundary pixels. Formally, the erosion of A by B is defined by:

$$A \ominus B = \{\, z : B_z \subseteq A\},$$

where B_z is the structuring element B translated so that its origin is at z. In essence, a point z remains in the eroded set only if the entire structuring element B translated to z fits inside A.

Erosion is useful for eliminating small artifacts and separating closely connected objects. Often, a round structuring element is used to preserve the circular shape of objects during erosion. In this solution however, a vectorized implementation of a square structuring element is used for computational efficiency.

In contrast, *dilation* grows the foreground set by adding pixels around object boundaries. It is defined by:

$$A \oplus B = \{ z : (B^R)_z \cap A \neq \emptyset \},$$

where B^R is the reflection of B about its origin:

$$B^R = \{(-b_x, -b_y) : (b_x, b_y) \in B\}.$$

Geometrically, a point z is included if, when we place the origin of B^R at z, there is at least one overlap with the set A.

Dilation is employed to close small holes and gaps within objects, ensuring more complete and connected structures. Similarly to erosion, a square structuring element is used for computational efficiency in this solution.

2.2.4 Morphological Opening and Closing

Two notable combinations of erosion and dilation are:

- **Opening** $(A \circ B) = (A \ominus B) \oplus B$. Opening smooths object boundaries and removes small objects or thin protrusions.
- **Closing** $(A \bullet B) = (A \oplus B) \ominus B$. Closing fills small gaps and connects nearby objects.

2.2.5 Boundary Extraction

The *boundary* (or contour) of a set A can be extracted by subtracting the eroded version of A from A itself:

$$\text{Boundary}(A) = A \setminus (A \ominus B),$$

where B is a simple structuring element (e.g., a 3×3 disk). This operation highlights the edges or boundaries of the objects in the image.

2.2.6 Hole Filling

Hole filling is used to fill in background regions that are surrounded by foreground in a binary image. One approach is to choose a seed point in each hole (often on the border of the hole) and iteratively perform dilations constrained to the complement of the original image until convergence. Alternatively, set-based morphological algorithms can fill holes by complement operations:

$$\text{Fill}(A) = \left(\bigcup_{n=0}^{\infty} (X_n \oplus B) \cap A^c \right)^c,$$

where X_0 is a seed point outside the hole region and we iteratively add all connected background pixels that can be reached via the structuring element.

2.2.7 Connected Component Extraction

Connected component extraction aims to isolate each group of connected foreground pixels (usually using 4- or 8-connectivity) from the rest of the image. A common approach uses iterative morphological operations:

1. Choose a single seed point $X_0 \subseteq A$ (e.g., any foreground pixel in the component of interest).

2. Define a connectivity structuring element B (for 4-connectivity, B might be a 3×3 cross; for 8-connectivity, a 3×3 square).

3. Iteratively *dilate* the current set of connected pixels and *intersect* the result with the original set A. Formally:

$$X_{k+1} = \big(X_k \oplus B\big) \cap A, \quad k = 0, 1, 2, \ldots$$

4. Stop when no more changes occur, i.e., $X_{k+1} = X_k$. At this point, X_k corresponds exactly to the connected component in A that contains the seed point.

By repeating this procedure with new seeds on remaining unextracted parts of the foreground, one can isolate every connected component in the image. Note that this approach does not assign labels to the components, it merely identifies which pixels in A belong to the same connected structure.

2.2.8 Hit-or-Miss Transformation

The *hit-or-miss* transformation detects a specific pattern in the image by looking for the arrangement of foreground and background pixels. Given two structuring elements B_1 (defining foreground shape) and B_2 (defining background shape), the hit-or-miss transform of A is:

$$A \otimes (B_1, B_2) = (A \ominus B_1) \cap (A^c \ominus B_2).$$

Here, (B_1, B_2) is placed at a candidate point, and if B_1 fits inside A (hit) and simultaneously B_2 fits inside A^c (miss), that point is considered a valid detection.

Hit-or-miss transforms are useful for detecting specific patterns or structures in binary images, such as corners, T-junctions, or other geometric arrangements. Note that hit-or-miss transforms do pixel-perfect matching.

2.3 Image Segmentation

Image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels) to simplify its representation and make it more meaningful for analysis. It enables further operations such as object recognition, tracking, and scene understanding. This section provides an overview of segmentation techniques, focusing on point, line, and edge detection, edge linking, region growing, and thresholding.

2.3.1 Point Detection

Discontinuities in intensity can manifest as isolated points, lines, or edges. Detecting these primitive features is often the first step in segmentation tasks.

A point in an image can be identified if its intensity differs significantly from those of its surroundings. One simple approach is to use the *Laplacian filter* at each pixel:

2.3.2 Line Detection

Line detection typically uses convolution masks designed to respond maximally to lines in certain orientations (e.g., 0°, 45°, 90°). For example, a line-detection mask for 0° orientation could be:

$$M_0 = \begin{bmatrix} -1 & 2 & -1 \end{bmatrix},$$

which, when convolved with the image, provides a strong response along horizontal lines. Similarly, rotated versions of M_0 are used to detect lines at different angles.

2.3.3 Edge Detection

Edges generally correspond to significant shifts in intensity. A gradient-based approach uses first derivatives:

$$G_x(x, y) = \frac{\partial f}{\partial x}, \quad G_y(x, y) = \frac{\partial f}{\partial y},$$

using operators like Sobel. The gradient magnitude and direction are: Pixels with a gradient magnitude above a threshold T are flagged as potential edges.

2.3.4 Edge Linking

Local Processing

Global Processing — Hough Transform

2.3.5 Region Growing

2.3.6 Multivariable Thresholding

2.4 Connected Components Labeling

3 Application: FISH Signal Counts

Pathologists rely on identifying specific mutations within tumor tissues to determine appropriate cancer treatments. A widely used method to visualize these mutations is

Fluorescence In-Situ Hybridization (FISH), which combines molecular biology and imaging techniques. In this method, fluorescent probes are attached to specific RNA sequences in the tumor biopsy, enabling the detection of genetic mutations.

These fluorescent labels are visualized using fluorescence microscopy, where each label emits light at specific wavelengths corresponding to the attached probe.

The combined imaging technique allows the simultaneous detection of genetic mutations and the spatial identification of cells. This method is essential for quantifying the occurrence and co-localization of mutations in individual cells.

3.1 Methodology

The process for detecting cells and calculating signal counts and ratios involves a series of steps:

3.1.1 Input Acquisition

The pipeline accepts three input images corresponding to different fluorescence channels:

- **DAPI Channel**: Stains the cell nuclei, providing a reference for cell localization.
- **Acridine Channel**: Highlights specific cellular components stained with Acridine.
- **FITC Channel**: Represents signals from FITC-labeled molecules.

These images are loaded in TIFF format using the `load_tiff_image` function. The images are then combined into a single RGB image for visualization purposes: DAPI (red), Acridine (green), and FITC (blue).

3.1.2 Thresholding

3.1.3 Saving Binary Masks

3.1.4 Morphological Operations

3.1.5 Connected Components Labeling

Post morphological processing, connected components labeling is applied:

- **Labeling Cells**: The cleaned DAPI mask is labeled using `label_connected_components`, identifying individual cells.
- **Labeling Signals**: Similarly, Acridine and FITC masks are labeled.

3.1.6 Border Touching and Cell Filtering

3.1.7 Signal Counting

The number of Acridine and FITC signals within each cell is counted using the `calculate_signal_count` function. This function iterates over each labeled cell and computes the number of signals present in the corresponding Acridine and FITC masks.

3.2 Results

Terminal command to run the analysis:

```
$ $python python/main.py fish-signal-counts \
    input_path_acridine \
    input_path_fitc \
    input_path_dapi
```

Here, the `input_path_acridine`, `input_path_fitc`, and `input_path_dapi` are the file paths to the Acridine, FITC, and DAPI images, respectively. In this analysis, we processed the provided sample images *data/input/BXY-ABCD_Region 002_FOV 00040_Acridine.tif*, *data/input/BXY-ABCD_Region 002_FOV 00040_FITC.tif*, and *data/input/BXY-ABCD_Region 002_FOV 00040_DAPI.tif*.

3.3 Binary Mask Visualization

Figure 1 presents the initial binary masks generated from the DAPI, Acridine, and FITC channels. The masks effectively isolate the nuclei and signal regions, providing a clear basis for subsequent analysis.

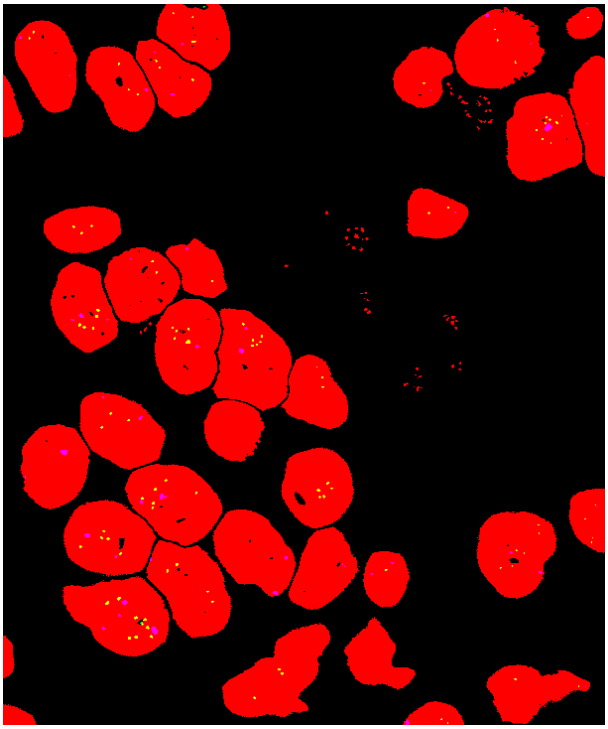

Figure 1: Initial Binary Masks for DAPI (red channel), Acridine (green channel), and FITC (blue channel).

Figure 2 shows the cleaned binary masks after morphological operations and filtering. The processed masks exhibit well-defined cell boundaries and reduced noise, facilitating accurate signal quantification. Cells that touch the image borders are marked with half-intensity.

Figure 2: Cleaned Binary Masks after Morphological Operations.

3.4 Signal Counts and Ratios

The pipeline successfully identified and labeled individual cells, enabling the calculation of Acridine and FITC signal counts within each cell. The program identifies 30 complete cells in the provided sample images and computes the signal counts and ratios for each cell.

Table 1 shows the signal counts and ratios for a sample of cells.

Table 1: Signal Counts and Ratios per Cell

Cell ID	**Centroid (X, Y)**	**Area (pixels)**	**Acridine**	**FITC**	**Acridine/FITC**
1	(586.82, 19.07)	841	1	0	-
2	(497.55, 45.82)	5015	3	2	1.50
3	(44.79, 60.05)	4161	2	2	1.00
...	...	...	...	...	...
30	(530.05, 691.11)	3332	2	1	2.00

4 Application: Circuit Board Quality Assurance

4.1 Methodology

4.2 Results

5 Application: Filled Bottles

5.1 Methodology

5.2 Results