Student: Aditya Sandhu

University: Colorado State University Global

Course: 25FC - CSC515 - 1 [Module 6 – Segmentation]

Critical Thinking Assignment [OpenCV Application of Adaptive Thresholding Scheme Accounting for Changes in Illumination]

Instructor: Dr Dong Nguyen

Date – 10/16/2025

GIT LINKS

Document Link –

Python File –

Image segmentation is the process of dividing an image into distinct and meaningful regions to simplify analysis and make object identification easier for computers. It assigns a label to every pixel so that pixels with similar characteristics, such as brightness, color, or texture that belong to the same region. This process helps in identifying and isolating objects within an image, such as separating a tumor from healthy tissue in a medical scan or distinguishing vehicles from the background in traffic footage. There are several segmentation methods, with two common ones being thresholding and region-based segmentation.

Thresholding separates an image based on pixel intensity values, making it simple and fast but sensitive to lighting variations and noise.

Region-based segmentation, on the other hand, groups neighboring pixels with similar properties, resulting in higher accuracy but at a higher computational cost. Together, these techniques form the foundation for many advanced applications in computer vision and image analysis.

**Figure 1 -** Python Script for Adaptive Thresholding Preprocessing Steps Using OpenCV.

A screenshot of a computer program

AI-generated content may be incorrect.

This code defines functions for loading, converting, filtering, and locally binarizing images to prepare them for adaptive thresholding segmentation, enabling noise reduction and illumination-adjusted object detection.

The import libraries are essential for image processing and visualization.  
cv2 enables image reading and transformations, while matplotlib.pyplot allows for plotting results. The os and sys modules manage file paths and handle program exits during execution.

*# Importing required modules for handling images and visualizations  
import* cv2 *as* vision\_lib *# Handles image reading and transformations  
import* matplotlib.pyplot *as* plot\_lib *# Enables creating and saving plots  
import* os *# Assists with file path management  
import* sys *# Allows for program exit on errors*

The below function loads an image file using its name and constructs the full path relative to the script’s location. It uses OpenCV’s imread() to read the image into memory. If the image cannot be loaded, it prints an error message indicating the missing or incorrect path. The program then terminates safely using sys.exit(1) to prevent further execution.

*def* load\_source\_image(file\_name):  
 *# Build the complete path relative to this script's location* full\_path = os.path.join(os.path.dirname(\_\_file\_\_), file\_name)  
 loaded\_img = vision\_lib.imread(full\_path)  
 *# Check if loading failed and exit with a message if so  
 if* loaded\_img *is None*:  
 print(f"Error: Could not load the image at '{full\_path}'. "  
 f"Verify the file exists and path is correct.")  
 sys.exit(1) *# Terminate the script on failure  
 return* loaded\_img

This function converts a color image into a single grayscale channel for simplified analysis.It uses OpenCV’s cvtColor() to transform the image from BGR to grayscale format.

*def* convert\_to\_monochrome(source\_img):  
 *# Reduce color channels to one for simplified analysis  
 return* vision\_lib.cvtColor(source\_img, vision\_lib.COLOR\_BGR2GRAY)

This function applies Gaussian blurring to reduce noise in a grayscale image.  
It uses a (7 x 7) kernel to smooth pixel intensity variations and enhance segmentation quality.

*def* apply\_noise\_reduction(mono\_img):  
 *# Use a larger kernel for broader smoothing effect  
 return* vision\_lib.GaussianBlur(mono\_img, (7, 7), 0)

This function performs adaptive thresholding to convert a filtered grayscale image into a binary image. It uses the **Gaussian-weighted method** to calculate local thresholds, making it robust to lighting variations. Each pixel’s threshold is determined from its (11 X 11) neighborhood with a bias adjustment factor of 2.

*# Function to binarize the image based on local variations  
def* apply\_local\_binarization(filtered\_img):  
 *# Adjust parameters for custom threshold behavior  
 return* vision\_lib.adaptiveThreshold(  
 filtered\_img, *# Filtered input* 255, *# Peak intensity value* vision\_lib.ADAPTIVE\_THRESH\_GAUSSIAN\_C, *# Weighted local method* vision\_lib.THRESH\_BINARY, *# Output as black/white* 11, *# Local area size* 2 *# Adjustment factor* )

**Figure 2 -** Python Script for Generating and Saving Adaptive Thresholding Outputs.

A screenshot of a computer program

AI-generated content may be incorrect.

This code in Figure 2, compiles the image processing stages [original, grayscale, filtered, and binarized] into a visual summary and saves both the combined view and the final binary image, demonstrating automated handling of uneven illumination using OpenCV and Matplotlib.

This function creates and saves a visual summary showing each stage of image processing side by side. It arranges the images horizontally, rendering the original in color and subsequent stages in grayscale. Titles are added for clarity, axes are removed for a clean layout, and the figure is saved at high resolution using Matplotlib.

*# Function to generate and store a combined view of processing stages  
def* generate\_and\_store\_visual\_summary(stages\_list, labels\_list, output\_file):  
 *# Initialize a wide figure for horizontal arrangement* plot\_lib.figure(figsize=(18, 6))  
 *for* pos *in* range(len(stages\_list)):  
 plot\_lib.subplot(1, len(stages\_list), pos + 1)  
 *if* pos == 0:  
 *# Adjust color order for accurate rendering* plot\_lib.imshow(vision\_lib.cvtColor(stages\_list[pos], vision\_lib.COLOR\_BGR2RGB))  
 *else*:  
 *# Render in monochrome for processed stages* plot\_lib.imshow(stages\_list[pos], cmap='gray')  
 plot\_lib.title(labels\_list[pos])  
 plot\_lib.axis('off')  
 *# Optimize spacing and save with high resolution* plot\_lib.tight\_layout()  
 plot\_lib.savefig(output\_file, dpi=400)  
 *# No on-screen display to keep script lightweight* plot\_lib.close()

This function saves the final binarized image as a separate output file.  
It uses OpenCV’s imwrite() to write the processed binary image to disk for later analysis or visualization.

*# Function to store the binarized result separately  
def* store\_final\_binary(result\_img, output\_file):  
 vision\_lib.imwrite(output\_file, result\_img)