EE 4065 – Embedded Digital Image Processing
Homework 3 Report

Course: EE 4065 – Embedded Digital Image Processing

Dilek Çelik 150721853
Aylin Doğan 150720048

1. Introduction

This report presents the implementation of basic image processing algorithms on an STM32 microcontroller as part of Homework 3. The study focuses on Otsu's thresholding method for both grayscale and color images, as well as fundamental morphological operations. All image processing tasks are executed on the embedded system, and the results are transferred to a PC for visualization using Python.

2. System Overview

This section describes the hardware and software environment.

2.1 Hardware Platform

Development Board: STM32F446RE

Communication Interface: USART2

2.2 Software Tools

The embedded software was developed in C using an appropriate IDE. Python was used on the PC side to visualize and verify the results obtained from the microcontroller.

3. Question 1 – Otsu's Thresholding (Grayscale Image)

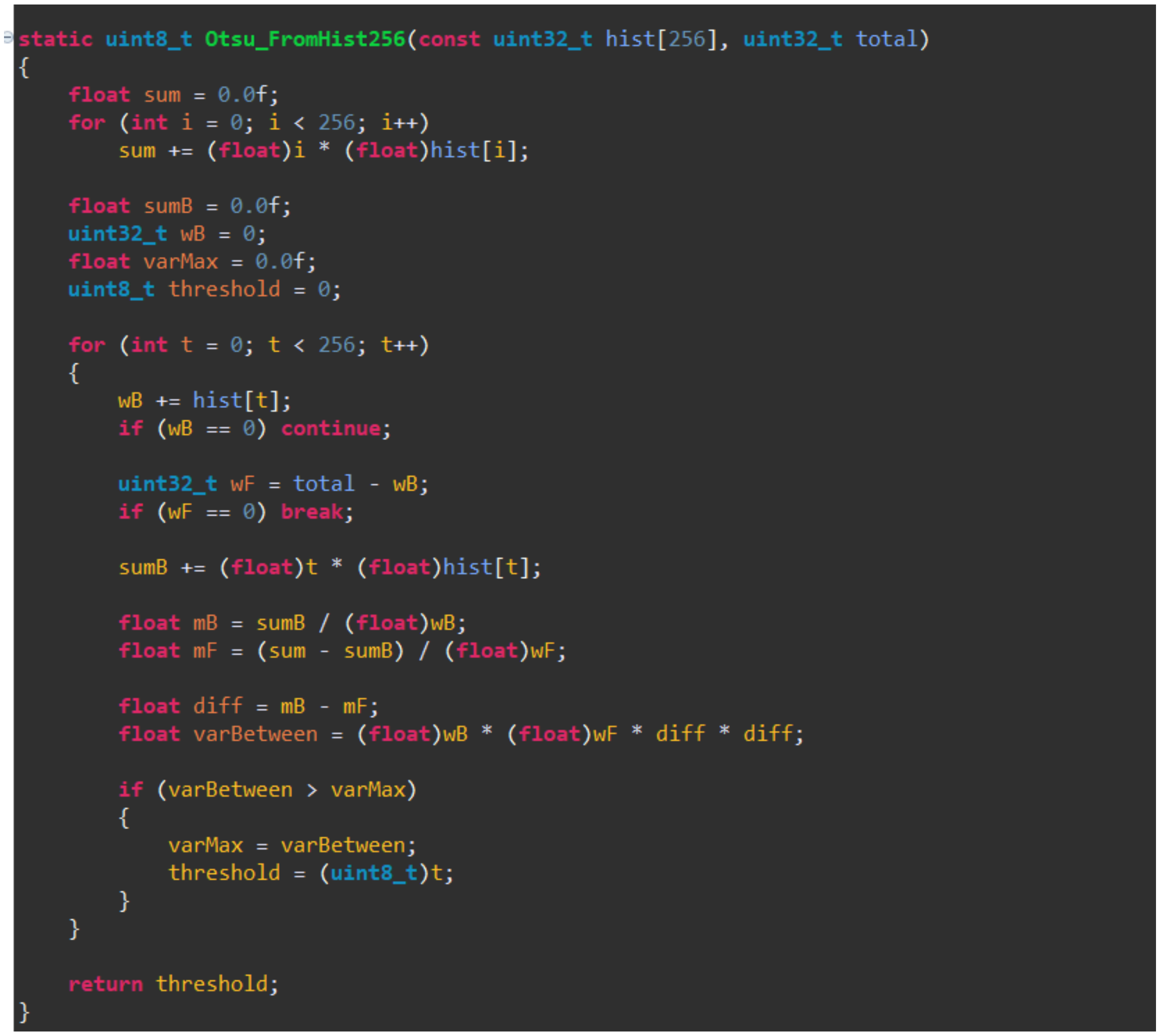
Otsu's thresholding method is implemented for grayscale images. The goal is to automatically determine an optimal threshold value that separates foreground and background regions in an image.

3.1 Otsu Thresholding Algorithm

Otsu's method is a global thresholding technique that selects the threshold value by maximizing the between-class variance of pixel intensities. This method does not require any manual threshold selection and is widely used in image segmentation applications.

3.2 Microcontroller Implementation

A C function was developed on the microcontroller to compute Otsu's threshold for a given grayscale image. The function calculates the histogram of the image, evaluates class probabilities, and determines the threshold that maximizes the between-class variance.

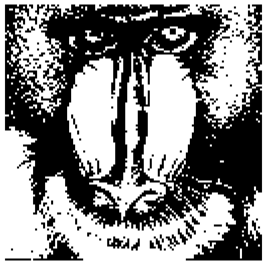


```c
static uint8_t Otsu_FromHist256(const uint32_t hist[256], uint32_t total)
{
    float sum = 0.0f;
    for (int i = 0; i < 256; i++)
        sum += (float)i * (float)hist[i];

    float sumB = 0.0f;
    uint32_t wB = 0;
    float varMax = 0.0f;
    uint8_t threshold = 0;

    for (int t = 0; t < 256; t++)
    {
        wB += hist[t];
        if (wB == 0) continue;

        uint32_t wF = total - wB;
        if (wF == 0) break;

        sumB += (float)t * (float)hist[t];

        float mB = sumB / (float)wB;
        float mF = (sum - sumB) / (float)wF;

        float diff = mB - mF;
        float varBetween = (float)wB * (float)wF * diff * diff;

        if (varBetween > varMax)
        {
            varMax = varBetween;
            threshold = (uint8_t)t;
        }
    }

    return threshold;
}
```

3.3 Grayscale Image Preparation

A grayscale image of appropriate resolution was generated on the PC and transferred to the microcontroller. The image was selected to clearly demonstrate the effectiveness of Otsu's thresholding method.

Grayscale image.

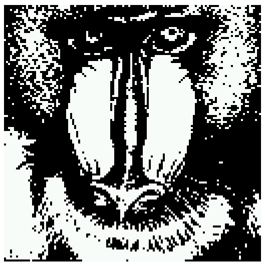


Result of Otsu's Method from grayscale image.

4. Question 2 – Otsu's Thresholding (Color Image)

In this part, Otsu's thresholding method was extended to color images. The color image was processed channel-wise before thresholding and converted to grayscale .

Color İmage.



Result of Otsu's Method from color image.

5. Question 3 – Morphological Operations

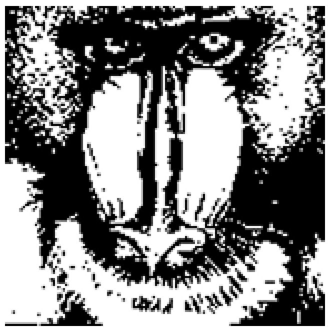
This section focuses on basic morphological operations applied to a binary image. These operations are commonly used to remove noise, fill gaps, and refine object shapes in binary images.

5.1 Morphological Operations Overview

The following morphological operations were implemented on the microcontroller: dilation, erosion, opening, and closing. Each operation was implemented as a separate C function.

5.2 Binary Image Source

The binary image obtained from Question 1 was used as the input for all morphological operations.



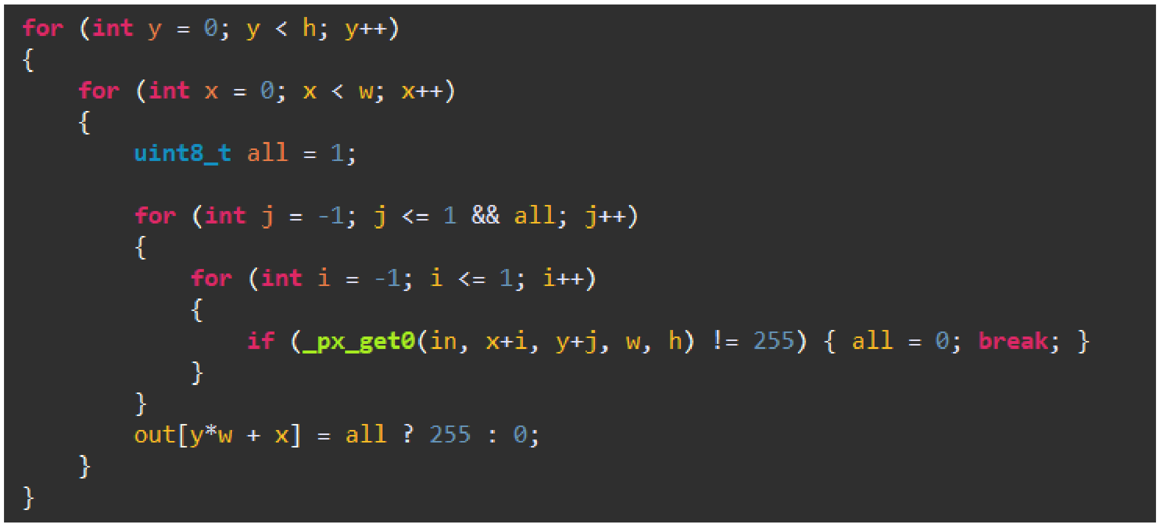
Result of Otsu's Method from grayscale image.

5.3 Microcontroller Implementation

*Dilation : if any 255 exist in 3*3 , we make them all 255.*

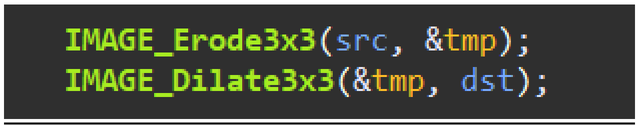
```c
for (int y = 0; y < h; y++)
{
    for (int x = 0; x < w; x++)
    {
        uint8_t any = 0;

        for (int j = -1; j <= 1 && !any; j++)
        {
            for (int i = -1; i <= 1; i++)
            {
                if (_px_get0(in, x+i, y+j, w, h) == 255) { any = 1; break; }
            }
        }
        out[y*w + x] = any ? 255 : 0;
    }
}
```

*Erosion : if any 0 exist in 3*3 , we make them all 0.*



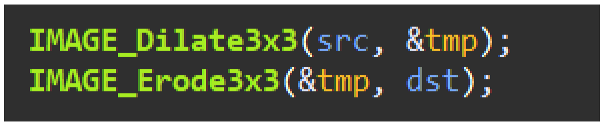
```
for (int y = 0; y < h; y++)
{
    for (int x = 0; x < w; x++)
    {
        uint8_t all = 1;

        for (int j = -1; j <= 1 && all; j++)
        {
            for (int i = -1; i <= 1; i++)
            {
                if (_px_get0(in, x+i, y+j, w, h) != 255) { all = 0; break; }
            }
        }
        out[y*w + x] = all ? 255 : 0;
    }
}
```

Opening = erosion -> dilation



```
IMAGE_Erode3x3(src, &tmp);
IMAGE_Dilate3x3(&tmp, dst);
```

Closing : dilation -> erosion

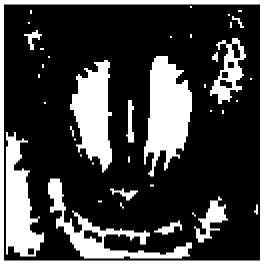


```
IMAGE_Dilate3x3(src, &tmp);
IMAGE_Erode3x3(&tmp, dst);
```

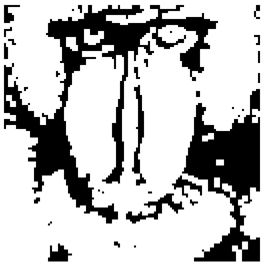
5.4 Morphological Results

The results of dilation, erosion, opening, and closing operations are shown below. All results were generated on the microcontroller and visualized on the PC.

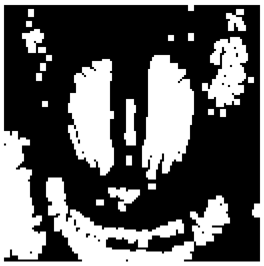
Erosion



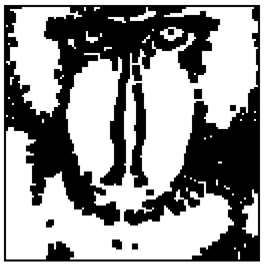
Dilation



Opening



Closing



6. Discussion

For color images, converting the RGB565 image into a grayscale representation before applying Otsu’s method provides a practical solution. Although some color information is lost during this conversion, the resulting binary images are still suitable for further processing. The segmentation quality remains acceptable for the purposes of morphological analysis.

The application of morphological operations significantly improves the quality of the binary images. Dilation helps to expand foreground regions and close small gaps, while erosion removes isolated noise pixels. Opening and closing operations combine these effects to eliminate small objects and fill holes inside foreground regions. The observed behavior of these operations matches their theoretical definitions.

7. Conclusion

In this homework, basic image processing algorithms were successfully implemented on an STM32 microcontroller. Otsu’s thresholding method was applied to both grayscale and color images, enabling automatic and effective conversion of images into binary form. Additionally, fundamental morphological operations including dilation, erosion, opening, and closing were implemented and tested using binary images.