,

MARMARA UNIVERSTY

FACULTY OF ENGINEERING

EE4065

Introduction to Embedded Image Processing

HOMEWORK 1

Submitted to: Prof. Dr. CEM ÜNSALAN

Due Date: 07.11.2025

	Dept.	**STUDENT ID**	**NAME SURNAME**
1	EE	150720049	RUMEYSA ŞENOL
2	EE	150720996	FATİME EMİN

Table of Contents

1. Introduction _____-3-

2. Methodology _____ -3-

3. Process and Outputs_____-4-

4. Conclusion_____-6-

Table of Figures

Fifure-1_____-4-
Figure-2_____-4-
Figure-3a_____-5-
Figure-3b_____-5-
Figure-4_____-5-

1.Introduction

In this homework, the concept of embedded digital image processing is explored using the STM32 Nucleo-F446RE microcontroller. The objective is to understand how basic image processing operations can be performed directly on an embedded system with limited computational resources. A grayscale image is first generated on a PC, converted into a C header file, and transferred into the STM32's memory. The image is then processed within the microcontroller using four different intensity transformations: negative, thresholding, gamma correction, and piecewise linear transformation. The results of these operations are verified by observing the modified pixel values in the STM32CubeIDE memory window, demonstrating how digital image manipulation can be achieved on an embedded platform without the use of high-level computing tools.

2.Methodology

Transforming the image and turning into a Header File

First, the implementation process began with selecting a sample image on a personal computer and converting it into a grayscale format using Python's Pillow library in VS code. Secondly, the image was resized to a smaller resolution suitable for the STM32's memory limitations (128 – KB SRAM) and saved as a C header file containing an 8-bit array of pixel intensity values. This header file was added to the STM32CubeIDE project and included in the main program file.

Intensity transformations to the header filed image using STM32

Using embedded C code, four intensity transformations—negative, thresholding, gamma correction, and piecewise linear mapping—were applied to the image array. Each operation was performed by iterating through the pixel data and storing the processed results in new arrays. The transformed images were not displayed but verified through the Memory Browser in STM32CubeIDE, where pixel value changes were observed and compared with the original image data. This method allowed direct examination of how pixel intensities are modified by each algorithm within the microcontroller's memory.

4.Process and Outputs

a. Negative Image:

In the negative transformation, each pixel value was subtracted from 255, effectively inverting the brightness of the image. Dark regions became bright, and bright regions turned dark. In the memory window, pixel values that were low in the original image (close to 0) appeared high (close to 255) after processing, and vice versa.

negative[0]	uint8_t	109 'm'
negative[1]	uint8_t	108 'l'
negative[2]	uint8_t	108 'l'
negative[3]	uint8_t	109 'm'
negative[4]	uint8_t	109 'm'
negative[5]	uint8_t	109 'm'
negative[6]	uint8_t	110 'n'
negative[7]	uint8_t	110 'n'
negative[8]	uint8_t	111 'o'
negative[9]	uint8_t	111 'o'
negative[10]	uint8_t	112 'p'
negative[11]	uint8_t	113 'q'
negative[12]	uint8_t	114 'r'
negative[13]	uint8_t	114 'r'
negative[14]	uint8_t	115 's'

Figure 1: Memory values of the negative image 0 to 14 indices

b. Thresholder Image:

The thresholding operation converted the grayscale image into a binary black-and-white image. A threshold value (T = 128) was used to decide whether each pixel would become 0 (black) or 255 (white). In the STM32 memory, the processed array contained only two values — 0 and 255 — clearly showing the segmentation effect.

thresholded[23]	uint8_t	255 'ÿ'
thresholded[24]	uint8_t	255 'ÿ'
thresholded[25]	uint8_t	255 'ÿ'
thresholded[26]	uint8_t	255 'ÿ'
thresholded[27]	uint8_t	255 'ÿ'
thresholded[28]	uint8_t	255 'ÿ'
thresholded[29]	uint8_t	255 'ÿ'
thresholded[30]	uint8_t	255 'ÿ'
thresholded[31]	uint8_t	255 'ÿ'
thresholded[32]	uint8_t	0 '\0'
thresholded[33]	uint8_t	0 '\0'
thresholded[34]	uint8_t	0 '\0'
thresholded[35]	uint8_t	0 '\0'
thresholded[36]	uint8_t	0 '\0'
thresholded[37]	uint8_t	0 '\0'

Figure 2: Memory values of the thresholder image 23 to 37 indices

c. Gamma Correlation:

Gamma correction adjusted the image contrast non-linearly. When γ = 3, the image became darker because lower intensity values were further reduced; when γ = 1/3, the image became brighter since dark areas were enhanced. In the memory data, the pixel values after γ = 3 shifted closer to 0, while those for γ = 1/3 shifted toward higher intensity levels (close to 255).

gamma3[0]	uint8_t	47 '/'
gamma3[1]	uint8_t	48 '0'
gamma3[2]	uint8_t	48 '0'
gamma3[3]	uint8_t	47 '/'
gamma3[4]	uint8_t	47 '/'
gamma3[5]	uint8_t	47 '/'
gamma3[6]	uint8_t	46 '.'
gamma3[7]	uint8_t	46 '.'
gamma3[8]	uint8_t	45 '-'
gamma3[9]	uint8_t	45 '-'
gamma3[10]	uint8_t	44 ','
gamma3[11]	uint8_t	44 ','
gamma3[12]	uint8_t	43 '+'
gamma3[13]	uint8_t	43 '+'

Figure 3a: Gamma Correlation with gamma = 3

gamma13[0]	uint8_t	211 'Ó'
gamma13[1]	uint8_t	212 'Ô'
gamma13[2]	uint8_t	212 'Ô'
gamma13[3]	uint8_t	211 'Ó'
gamma13[4]	uint8_t	211 'Ó'
gamma13[5]	uint8_t	211 'Ó'
gamma13[6]	uint8_t	211 'Ó'
gamma13[7]	uint8_t	211 'Ó'
gamma13[8]	uint8_t	210 'Ò'
gamma13[9]	uint8_t	210 'Ò'
gamma13[10]	uint8_t	210 'Ò'
gamma13[11]	uint8_t	209 'Ñ'
gamma13[12]	uint8_t	209 'Ñ'
gamma13[13]	uint8_t	209 'Ñ'

Figure 3b: Gamma Correlation with gamma = 1/3

d. Piecewise Linear Transform:

In part (d), a piecewise linear transformation was applied to the thresholded image obtained in part (b). The purpose of this step was to enhance the binary result by introducing gradual intensity variations rather than abrupt transitions between black and white regions. By applying different linear mappings to specific intensity intervals, the contrast around the threshold level was adjusted, resulting in smoother brightness transitions. In the STM32 memory, the pixel values of the piecewise-transformed image no longer contained only 0 and 255, but intermediate values depending on the defined linear segments, showing the effectiveness of sequential intensity transformation in embedded processing.

piecewise[26]	uint8_t	200 'È'
piecewise[27]	uint8_t	200 'È'
piecewise[28]	uint8_t	200 'È'
piecewise[29]	uint8_t	200 'È'
piecewise[30]	uint8_t	200 'È'
piecewise[31]	uint8_t	200 'È'
piecewise[32]	uint8_t	50 '2'
piecewise[33]	uint8_t	50 '2'
piecewise[34]	uint8_t	50 '2'
piecewise[35]	uint8_t	50 '2'
piecewise[36]	uint8_t	50 '2'
piecewise[37]	uint8_t	50 '2'
piecewise[38]	uint8_t	50 '2'

Figure 4: Memory values of the P.L.T image 26 to 38 indices

Overall, the transformations demonstrated how pixel-level operations can be directly executed on an embedded platform, with each method producing a distinct pattern of intensity changes observable through the STM32CubeIDE memory window.

3.Conclusion

In this homework, we implemented several basic image processing operations—negative transformation, thresholding, gamma correction, and piecewise linear adjustment—on the STM32 Nucleo-F446RE microcontroller. The processed image data was transferred and handled entirely within the STM32's memory, without displaying the actual image output. Instead, the results were interpreted and verified by examining the numerical pixel values stored in memory through STM32CubeIDE. These observations clearly showed how each transformation affected the pixel intensity distribution, even without visual rendering. This experiment provided valuable insight into how image processing principles can be executed and analyzed at the embedded hardware level, strengthening our understanding of memory-based data manipulation and embedded image computation.