# EE4065 – Embedded Digital Image Processing
# Homework 1 Report

**Students Name:** Aylin DOĞAN, Dilek ÇELİK
**Course:** EE4065 Embedded Digital Image Processing

## 1. Objective

The purpose of this experiment is to implement fundamental intensity transformation operations on a grayscale image using an STM32 microcontroller. The processed results will be analyzed through the **Memory Window** in STM32CubeIDE.

The main objectives are: - To handle image data in an embedded system environment. - To apply various pixel-level intensity transformations. - To understand gamma correction and piecewise linear mapping using lookup tables (LUTs).

## 2. Hardware and Software Setup

| Component | Specification |
|---|---|
| Microcontroller | STM32F446RE (ARM Cortex-M4) |
| IDE | STM32CubeIDE |
| Programming Language | C |
| Peripheral Interfaces | UART2 for debug (optional) |
| Observation Tool | STM32CubeIDE Memory Window |

## 3. Project Files Overview

| File Name | Description |
|---|---|
| **main.c** | Contains main application and image processing routines. |
| **image_data.h** | Header file storing grayscale image data array. |
| **stm32f4xx_hal_msp.c**, **stm32f4xx_it.c** | STM32 HAL peripheral support files. |

| File Name | Description |
|---|---|
| **system_stm32f4xx.c** | System clock and initialization code. |

# 4. Implementation Details

## 4.1 Image Input and Parameters

A grayscale image was prepared and stored as a C array in image_data.h. The code processes a limited section of the image to avoid memory overflow.

```
#define IMG_PIXELS (IMG_W * IMG_H)
#define OUT_N   1024
#define OFFSET  0
```

Additionally, the first 20 pixels from the original image were copied to a separate array for visual verification in the Memory Window.

## 4.2 Transformation Functions

### *(a) Negative Transformation*

Each pixel is inverted to create a photographic negative effect.

```
dst[i] = 255 - src[i];
```

### *(b) Thresholding*

Each pixel is compared against a fixed threshold value.

```
dst[i] = (src[i] >= T) ? 255 : 0;
```

This converts the grayscale image into a binary black-and-white image.

### *(c) Gamma Correction (γ = 3 and γ = 1/3)*

A lookup table (LUT) is generated for each gamma value.

```
float y = powf(x / 255.0, gamma) * 255;
```

- **γ = 3:** Enhances darker regions, compresses bright areas.
- **γ = 1/3:** Brightens dark areas, compresses highlights.

### *(d) Piecewise Linear Transformation*

Implements a segmented linear mapping with two slopes:

```
if (x <= T) y = k1 * x; else y = highMin + k2 * (x - T);
```

This allows control over contrast in two distinct intensity regions.

## 5. Execution Flow

1. System peripherals (GPIO, UART) are initialized.
2. run_image_ops() is called once to perform all transformations sequentially.
3. The following buffers are populated:
    - first_pixels[] → Original sample
    - img_neg[] → Negative image
    - img_th[] → Thresholded image
    - img_gam3[] → Gamma 3.0 image
    - img_gam13[] → Gamma 1/3 image
    - img_pw[] → Piecewise linear result

Users can inspect each buffer under **Memory Window** to validate the transformation results.

## 6. Observations and Results

| Expression | Type | Value |
|---|---|---|
| first_pixels | volatile uint8_t [20] | 0x200000c0 <first_pixels> |
| img_neg | volatile uint8_t [1024] | 0x200000d4 <img_neg> |
| img_th | volatile uint8_t [1024] | 0x200004d4 <img_th> |
| img_gam3 | volatile uint8_t [1024] | 0x200008d4 <img_gam3> |
| img_gam13 | volatile uint8_t [1024] | 0x20000cd4 <img_gam13> |
| img_pw | volatile uint8_t [1024] | 0x200010d4 <img_pw> |

- **Imgage:**

0x200000c0

0x200000c0 <Traditional>

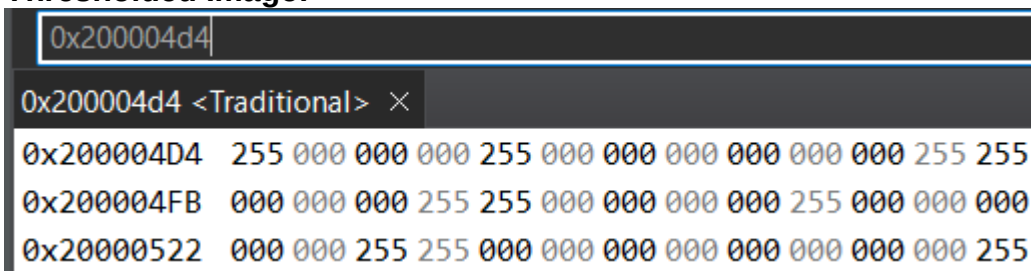
```
0x200000C0  145 056 049 089 137 090 062 033 071 077 092 145 153
0x200000E7  162 182 165 153 189 183 134 134 184 198 109 082 189
0x2000010E  087 099 169 091 062 131 086 122 114 107 071 098 149
```

- **Negative Image:**

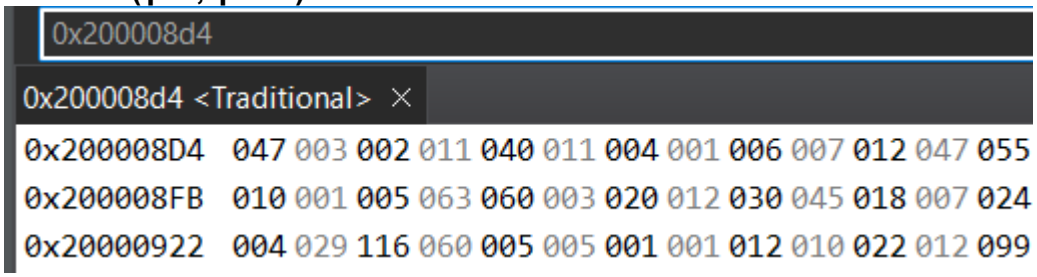
0x200000d4 <Traditional>

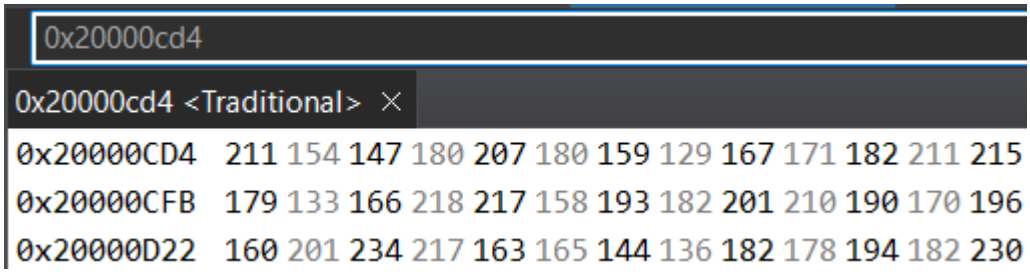
```
0x200000D4  110 199 206 166 118 165 193 222 184 178 163 110 102
0x200000FB  167 219 185 095 098 194 145 162 130 112 149 179 139
0x20000122  192 131 059 098 188 186 209 216 163 168 143 163 069
```

- **Thresholded Image:**

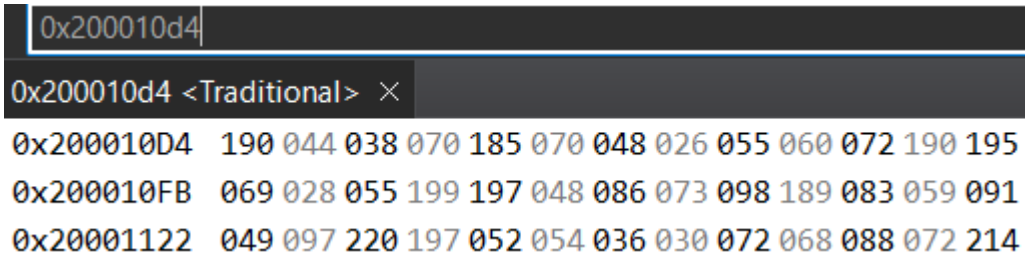



- **Gamma (γ=3, γ=1/3):**




- **Piecewise Linear:**



## 7. Discussion

The obtained results clearly demonstrate how different intensity transformations affect image perception in the embedded environment. The negative transformation successfully inverted brightness levels, providing a complementary visual representation. Thresholding effectively converted the grayscale image into a binary one, enhancing edge details but sacrificing gradual intensity transitions. Gamma correction showed a strong non-linear effect — with γ = 3 compressing highlights and γ = 1/3 amplifying darker regions — proving how power-law mapping controls contrast distribution. Finally, the piecewise linear transformation offered a flexible way to enhance contrast selectively in different brightness zones, confirming its practical use in embedded display optimization.

## 8. Conclusion

In conclusion, the experiment validated that embedded systems like STM32 can efficiently perform real-time pixel-level image transformations with limited memory and processing power. Each technique revealed distinct effects: negative transformation emphasized inversion accuracy, thresholding enabled segmentation, gamma correction demonstrated non-linear control, and piecewise linear mapping balanced brightness adaptively. These results collectively highlight how mathematical transformations can be implemented even on low-resource microcontrollers, making them valuable for applications in embedded vision, adaptive contrast control, and lightweight image preprocessing.

## 9. Appendix

- Source code: main.c, image_data.h
- IDE configuration screenshots (optional)
- Memory Window captures for each transformation