EE4065 – Embedded Digital Image Processing

Homework 2 Report

Students Name: Aylin DOĞAN, Dilek ÇELİK

Course: EE4065 Embedded Digital Image Processing

1. Objective

The objective of this experiment is to implement histogram formation, histogram equalization, 2D convolution filtering (low-pass and high-pass), and median filtering on a 128×128 grayscale image using STM32F446RE microcontroller. The results are analyzed using STM32CubeIDE Memory Window.

2. Hardware and Software Setup

Microcontroller: STM32F446RE (ARM Cortex-M4)

IDE: STM32CubeIDE

Programming Language: C

Observation Tool: STM32CubeIDE Expressions & Memory Window

3. Project Files Overview

main.c – Main application and image processing routines

mouse_image.h – Image stored as a C array

stm32f4xx_it.c, stm32f4xx_hal_msp.c – HAL and interrupt handlers

system_stm32f4xx.c – System clock configuration

4. Implementation Details

4.1 Histogram Formation (Q1)

A 256-bin histogram is computed by counting each pixel value.

4.1.a

The form of the c function to calculate histogram is given below.

```c
static void compute_histogram(const uint8_t* img, int w, int h, uint32_t hist[256]) {
  for (int i = 0; i < 256; ++i) hist[i] = 0;

  for (int y = 0; y < h; ++y) {
    for (int x = 0; x < w; ++x) {
      uint8_t p = img[y * w + x];
      hist[p]++;
    }
  }
}
```

4.1.b

histogram entries from STM32Cube IDE are given below.

```
hist_original
        Details:{0 <repeats 16 times>, 1, 1, 1, 2, 2, 6, 13, 16, 19, 27, 55, 62, 93, 165, 122, 131,
128, 126, 108, 75, 62, 53, 62, 56, 37, 53, 51, 31, 36, 36, 40, 29, 26, 26, 32, 27, 21, 27, 13, 23, 31,
18, 20, 26, 17, 22, 29, 23, 19, 20, 25, 19, 19, 15, 14, 18, 26, 22, 21, 14, 7, 17, 15, 18, 18, 6, 13, 11,
13, 8, 11, 10, 12, 10, 13, 11, 9, 6, 9, 13, 9, 14, 15, 8, 11, 11, 9, 8, 11, 17, 12, 14, 7, 13, 8, 11, 10, 8,
11, 14, 16, 14, 7, 16, 8, 9, 5, 10, 14, 12, 24, 26, 27, 66, 92, 118, 96, 85, 89, 157, 165, 162, 201,
214, 268, 284, 436, 499, 534, 917, 738, 479, 469, 269, 180, 209, 165, 180, 186, 141, 188, 277,
119, 147, 214, 186, 365, 344, 840, 1022, 1002, 327, 244, 164, 226, 82, 39, 23, 35, 31, 9, 14, 6, 1,
0, 1, 0 <repeats 34 times>, 3, 2, 3, 3, 3, 9, 13, 16, 16, 26, 34, 0 <repeats 29 times>}
        Default:0x200000d4 <hist_original>
        Decimal:536871124
        Hex:0x200000d4
        Binary:10000000000000000000011010100
        Octal:04000000324
```

4.2 Histogram Equalization (Q2)

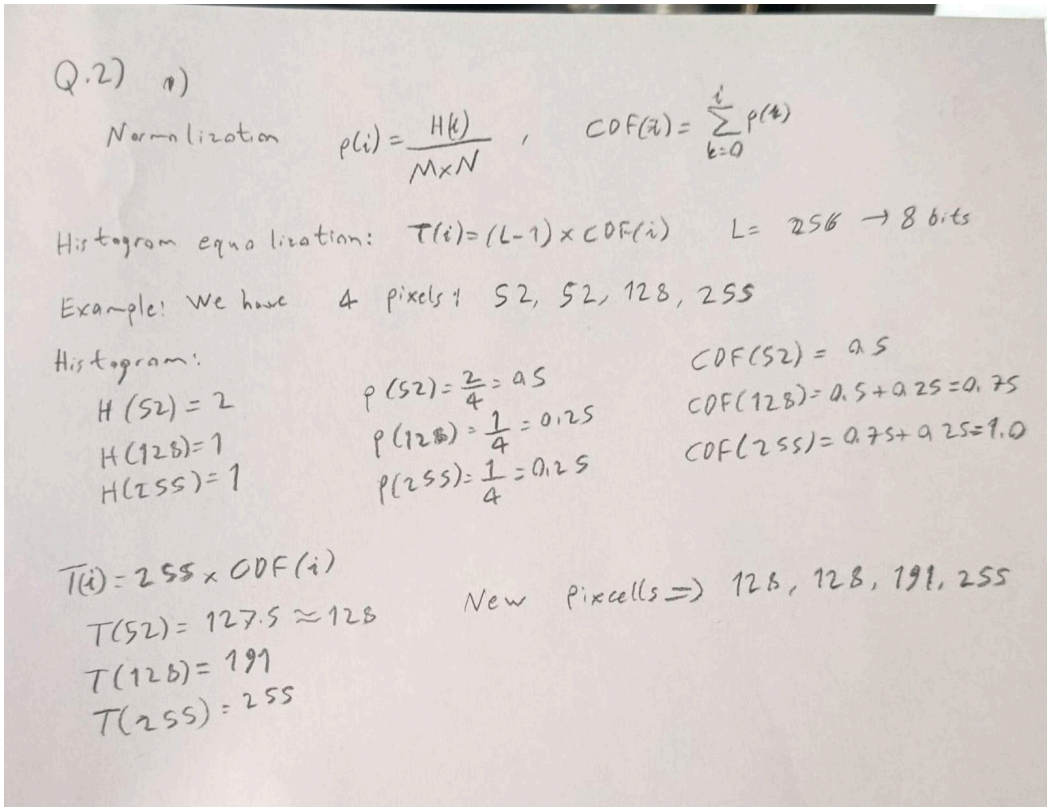
CDF-based lookup table generated and applied to enhance contrast.

```
void build_equalization_lut(const uint32_t hist[256], int total, uint8_t lut[256]
{
    uint32_t cdf = 0;
    uint32_t cdf_min = 0;

    for (int i = 0; i < 256; i++)
    {
        cdf += hist[i];

        if (cdf_min == 0 && cdf != 0)
            cdf_min = cdf;
```

```
        uint32_t v = (cdf - cdf_min) * 255u / (total - cdf_min);
        lut[i] = (v > 255) ? 255 : v;
    }
}
```

4.2.a

The Histogram Eq. sample is given below.



Q.2) a)

Normalization $p(i) = \frac{H(i)}{M \times N}$, $COF(i) = \sum_{k=0}^{i} p(k)$

Histogram equalization: $T(i) = (L-1) \times COF(i)$ $L = 256 \rightarrow 8$ bits

Example: We have 4 pixels: 52, 52, 128, 255

Histogram:

$H(52) = 2$
$H(128) = 1$
$H(255) = 1$

$p(52) = \frac{2}{4} = 0.5$
$p(128) = \frac{1}{4} = 0.25$
$p(255) = \frac{1}{4} = 0.25$

$COF(52) = 0.5$
$COF(128) = 0.5 + 0.25 = 0.75$
$COF(255) = 0.75 + 0.25 = 1.0$

$T(i) = 255 \times COF(i)$

$T(52) = 127.5 \approx 128$
$T(128) = 191$
$T(255) = 255$

New pixcells => 128, 128, 191, 255

4.2.b

The C function on the microcontroller to apply histogram equalization is given below.

```c
static void build_equalization_lut(const uint32_t hist[256], int total,
                                   uint8_t lut[256]) {
  uint32_t cumulative = 0;
  uint32_t cdf_min = 0;
  uint32_t total_u = (uint32_t)total;

  for (int i = 0; i < 256; ++i) {
    cumulative += hist[i];
    if (cdf_min == 0 && cumulative != 0) {
      cdf_min = cumulative;
    }

    uint32_t v = 0;
    uint32_t denom = total_u - cdf_min;
    if (denom != 0U) {
      v = (cumulative - cdf_min) * 255u / denom;
    }
    if (v > 255u) v = 255u;
    lut[i] = (uint8_t)v;
  }
}
```

4.2.c

Histogram equalization from STM32Cube IDE are given below. Some bins of the equalized histogram appear as zero because histogram equalization maps many original intensity values into a smaller number of output intensity levels. This is expected behavior and not an error.

```
hist_equalized
        Details:{61, 27, 55, 62, 93, 0, 0, 165, 0, 122, 0, 131, 0, 128, 0, 126, 108, 75, 62, 53, 62, 56, 90,
51, 67, 76, 55, 58, 75, 67, 64, 39, 91, 63, 47, 83, 57, 69, 67, 60, 62, 61, 75, 61, 50, 77, 66, 0, 92, 0, 118,
96, 0, 85, 89, 0, 157, 0, 0, 165, 0, 162, 0, 0, 0, 201, 0, 0, 214, 0, 0, 0, 268, 0, 0, 0, 284, 0, 0, 0, 0, 0, 0, 436,
0, 0, 0, 0, 0, 0, 0, 499, 0, 0, 0, 0, 0, 0, 0, 534, 0 <repeats 14 times>, 917, 0, 0, 0, 0, 0, 0, 0, 0, 0, 738, 0, 0,
0, 0, 0, 0, 0, 479, 0, 0, 0, 0, 0, 0, 469, 0, 0, 0, 269, 0, 0, 180, 0, 0, 209, 0, 0, 165, 0, 180, 0, 0, 186, 0, 0, 141,
0, 0, 188, 0, 0, 0, 277, 0, 119, 0, 147, 0, 0, 214, 0, 0, 186, 0, 0, 0, 0, 0, 365, 0, 0, 0, 0, 344, 0 <repeats 12
times>, 840, 0 <repeats 15 times>...}
        Default:0x200004d4 <hist_equalized>
        Decimal:536872148
        Hex:0x200004d4
        Binary:10000000000000000010011010100
        Octal:04000002324
```

4.3 Low-Pass & High-Pass Filters (Q3)

Low-pass filter smooths the image using a 3×3 averaging kernel.

High-pass filter enhances edges using a 3×3 Laplacian kernel.

2D convolution performed by sliding window and weighted sum.

Filtering starts only when the 3×3 window is fully available.

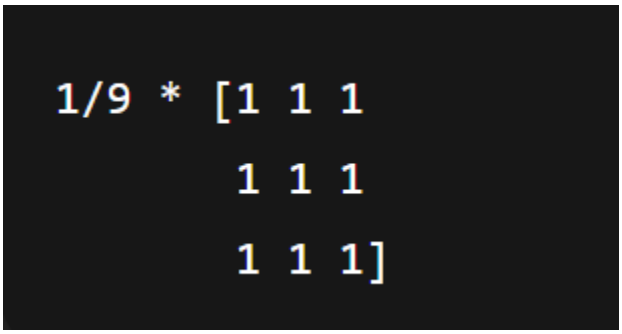
Therefore, rows y = 0 and y = 1 are not filtered because they do not have enough previous rows to form the window.

4.3.a

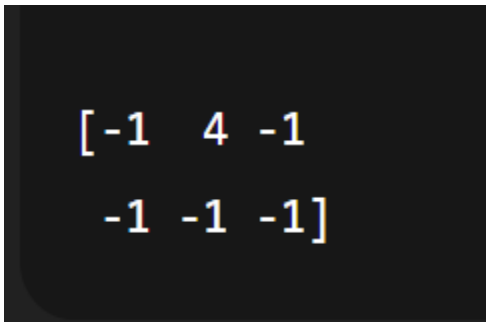
Filtering begins at row $y \geq 2$ because a complete 3×3 window is required.

The algorithm slides a 3×3 kernel across the equalized image and computes the average (low-pass) or Laplacian (high-pass).

Low-pass kernel:



High-pass kernel (Laplacian):



The C function on the microcontroller to apply 2D convolution is given below.

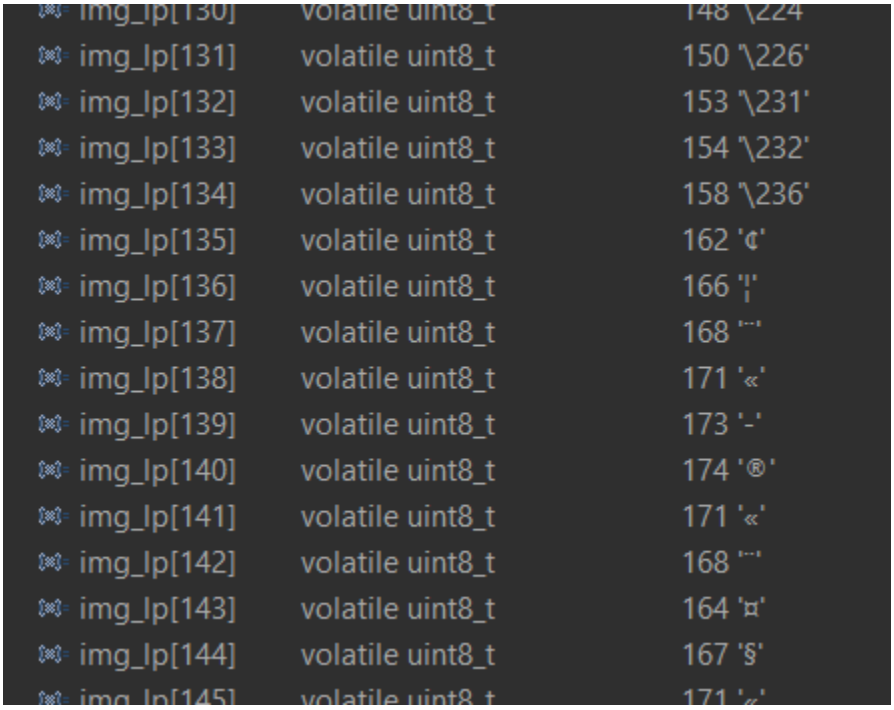
```
for (int y = 0; y < IMG_H; ++y) {
  uint8_t* row = row_buf[y % 3];


  if (y >= 2) {
    uint8_t* row_top = row_buf[(y - 2) % 3];
    uint8_t* row_mid = row_buf[(y - 1) % 3];
    uint8_t* row_bot = row_buf[y % 3];
    size_t row_base = (size_t)(y - 1) * IMG_W;

    for (int x = 1; x < IMG_W - 1; ++x) {
      int sum_lp =
          row_top[x - 1] + row_top[x] + row_top[x + 1] +
          row_mid[x - 1] + row_mid[x] + row_mid[x + 1] +
          row_bot[x - 1] + row_bot[x] + row_bot[x + 1];
      uint8_t lp_val = (uint8_t)(sum_lp / 9);

      int hp_sum = 0;
      hp_sum += -row_mid[x - 1];
      hp_sum += -row_mid[x + 1];
      hp_sum += -row_top[x];
      hp_sum += -row_bot[x];
      hp_sum += row_mid[x] * 4;
      uint8_t hp_val = clamp_u8(hp_sum);

      uint8_t window[9] = {
          row_top[x - 1], row_top[x], row_top[x + 1],
          row_mid[x - 1], row_mid[x], row_mid[x + 1],
          row_bot[x - 1], row_bot[x], row_bot[x + 1],
      };
      for (int i = 0; i < 9; ++i) {
        for (int j = i + 1; j < 9; ++j) {
```

```
      for (int i = 0; i < 9; ++i) {
        for (int j = i + 1; j < 9; ++j) {
          if (window[j] < window[i]) {
            uint8_t tmp = window[i];
            window[i] = window[j];
            window[j] = tmp;
          }
        }
      }
      uint8_t med_val = window[4];

      size_t idx = row_base + (size_t)x;
      store_window_value(idx, OFFSET, safeN, img_lp, lp_val);
      store_window_value(idx, OFFSET, safeN, img_hp, hp_val);
      store_window_value(idx, OFFSET, safeN, img_med, med_val);
    }
  }
}

for (int i = 0; i < 256; ++i) {
  hist_equalized[i] = hist_eq_local[i];
}
```

4.3.b

The filtered low pass image entries from STM32Cube IDE are given below.

```
img_lp
        Details:'\0' <repeats 129 times>,
"\223\224\226\231\232\236¢¦¨«®«¨¤§«°³±°⁻°²³´µ¶¸¹º»¾ÁÂÅÁÁÂÃÅÇÈÉÊÊÉÈÈÉÊÌÍÍÌÌÌÌÏÌÎÑÔÖÖÔ
ÑÍÍ"...
        Default:0x20000cd4 <img_lp>
        Decimal:536874196
        Hex:0x20000cd4
        Binary:10000000000000000110011010100
        Octal:04000006324
```



4.3.c

The filtered high pass image entries from STM32Cube IDE are given below.

```
img_lp
        Details:'\0' <repeats 129 times>,
"\223\224\226\231\232\236¦¨«®«¨¤§«°³±°⁻°²³´µ¶¸¹º»¾ÁÂÃÁÁÂÃÅÇÈÉÊÈÈÉÊÌÍÍÌÌÌÌÏÌÎÑÔÖÔÔÑÍÍ"...
        Default:0x20000cd4 <img_lp>
        Decimal:536874196
        Hex:0x20000cd4
        Binary:10000000000000000110011010100
        Octal:04000006324
```

img_hp[263]	volatile uint8_t	77 'M'
img_hp[264]	volatile uint8_t	59 ';'
img_hp[265]	volatile uint8_t	71 'G'
img_hp[266]	volatile uint8_t	43 '+'
img_hp[267]	volatile uint8_t	55 '7'
img_hp[268]	volatile uint8_t	52 '4'
img_hp[269]	volatile uint8_t	47 '/'
img_hp[270]	volatile uint8_t	64 '@'
img_hp[271]	volatile uint8_t	67 'C'
img_hp[272]	volatile uint8_t	69 'E'
img_hp[273]	volatile uint8_t	49 '1'
img_hp[274]	volatile uint8_t	37 '%'
img_hp[275]	volatile uint8_t	44 ','
img_hp[276]	volatile uint8_t	38 '&'
img_hp[277]	volatile uint8_t	38 '&'
img_hp[278]	volatile uint8_t	51 '3'

4.4 Median Filtering (Q4)

A 3×3 window is sorted and the median value is assigned for noise removal.

```
uint8_t median9(uint8_t w[9])
{
    for (int i = 0; i < 9; i++)
        for (int j = i + 1; j < 9; j++)
            if (w[j] < w[i])
            {
                uint8_t t = w[i];
                w[i] = w[j];
                w[j] = t;
            }

    return w[4];
}
```

4.4.a

The C function on the microcontroller to apply median filtering is given below.

4.4.b

The filtered image entries from STM32Cube IDE are given below.

```
img_med
        Details:'\0' <repeats 129 times>,
"}}\205\214\214\214\223\223\226\226\231\231\231\223\223\226\231\233\236\236\233\2
33\233\236\236\236\236¡¡¡¤¤¤ª¯¯¯¯¯¯ ,,,½½,,," , '½' <repeats 13 times>, "ÊÊÚÚÚ½,,"...
        Default:0x200014d4 <img_med>
        Decimal:536876244
        Hex:0x200014d4
        Binary:10000000000000000010010011010100
        Octal:04000012324
```

5. Execution Flow

1. Initialize peripherals.

2. Run all image operations sequentially.

3. Store first 1024 pixels of each output buffer.

4. Inspect buffers using STM32CubeIDE Memory Window.

Only the first 1024 pixels (OUT_N = 1024) are stored and inspected in the Memory Window to reduce RAM usage and simplify debugging on the microcontroller.

6. Discussion

Histogram equalization improved contrast significantly. Low-pass filtering reduced noise but blurred edges. High-pass filtering enhanced object boundaries. Median filtering effectively removed salt-and-pepper noise while preserving edge details. Median filtering preserved edges significantly better than the low-pass filter.

The low-pass filter blurred edges due to uniform averaging, while the median filter removed impulse (salt-and-pepper) noise without smoothing the edges.

7. Conclusion

The STM32F446RE successfully processed a 128×128 grayscale image using multiple filtering and transformation techniques. The experiment validated real-time embedded image processing feasibility. The STM32F446RE demonstrated that real-time embedded digital image processing is feasible even on a resource-limited microcontroller.

8. Appendix

Source code: main.c, mouse_image.h