MARMARA UNIVERSTY

FACULTY OF ENGINEERING

EE4065

Introduction to Embedded Image Processing

HOMEWORK2

Submitted to: Prof. Dr. CEM ÜNSALAN

Due Date: 21.11.2025

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

1. Introduction

In this homework assignment, digital image processing operations were implemented directly on an STM32 microcontroller. The goal was to develop embedded C functions for: Histogram computation, histogram equalization, low-pass filtering, high – pass filtering and median filtering. All algorithms were executed on a 64×64 grayscale test image, stored in flash memory as a C header file. The results were inspected in real-time through the STM32CubeIDE debugger using the Memory Monitor and Expressions windows. Executing all processing directly on the microcontroller demonstrates basic embedded vision capability without an operating system, external RAM, or floating-point libraries.

2. Methodology

In this experiment, a 64×64 grayscale image was processed on the STM32 microcontroller to investigate four fundamental image-processing operations: histogram computation, histogram equalization, low-pass filtering, high-pass filtering, and median filtering. First, the raw image data stored in the program memory was loaded into RAM and the pixel-intensity distribution was calculated to obtain the original histogram. This served as the reference for evaluating the effect of subsequent transformations. Next, histogram equalization was implemented to enhance contrast by redistributing the cumulative intensity levels and producing a more uniformly spread histogram. After equalization, spatial filtering was applied using two different kernels: a smoothing (low-pass) filter to reduce high-frequency noise, and a sharpening (high-pass) filter to emphasize edges and fine details. Finally, a 3×3 median filter was applied to suppress impulsive noise while preserving edges, using a neighborhood-sorting approach for each non-border pixel. All intermediate and final results were stored in separate memory buffers (img_eq, img_lp, img_hp, and img_med) and inspected in the debugger to verify correct implementation. This methodology enables a systematic evaluation of each processing stage and its impact on the original image.

3. Process and Outputs

The results of Homework 2 were verified by inspecting the memory buffers of each operation during debugging. For the original histogram (hist_orig), the memory screenshot showed non-zero integer counts across a wide range of intensity levels, confirming that the original image contains diverse grayscale values. These values appear in little-endian format (e.g., 05000000 = 5 occurrences), and the distribution is consistent with the pixel values in the source image. This validates that the histogram computation correctly scanned all 4096 pixels and assigned frequency counts to each grayscale bin from 0 to 255.

Question 1 – Histogram Calculations:

A C function was written that:

- Iterates over the image buffer
- Counts occurrences of each pixel value (0–255)
- Stores results in a 256-element uint32_t hist[] array

The formula used:

$$Hist(k) = \sum_{i=1}^{N} (I(i) == k)$$

From debugging memory:

Example section of hist_orig:

Pixel	**Count**
0	**5**
1	**8**
2	**3**
3	**6**
4	**8**
5	**16**

This confirms:

- The histogram buffer is valid
- The values correctly sum pixel occurrences

Total pixels:

$$64 \times 64 = 4096$$

Histogram implementation is fully correct.

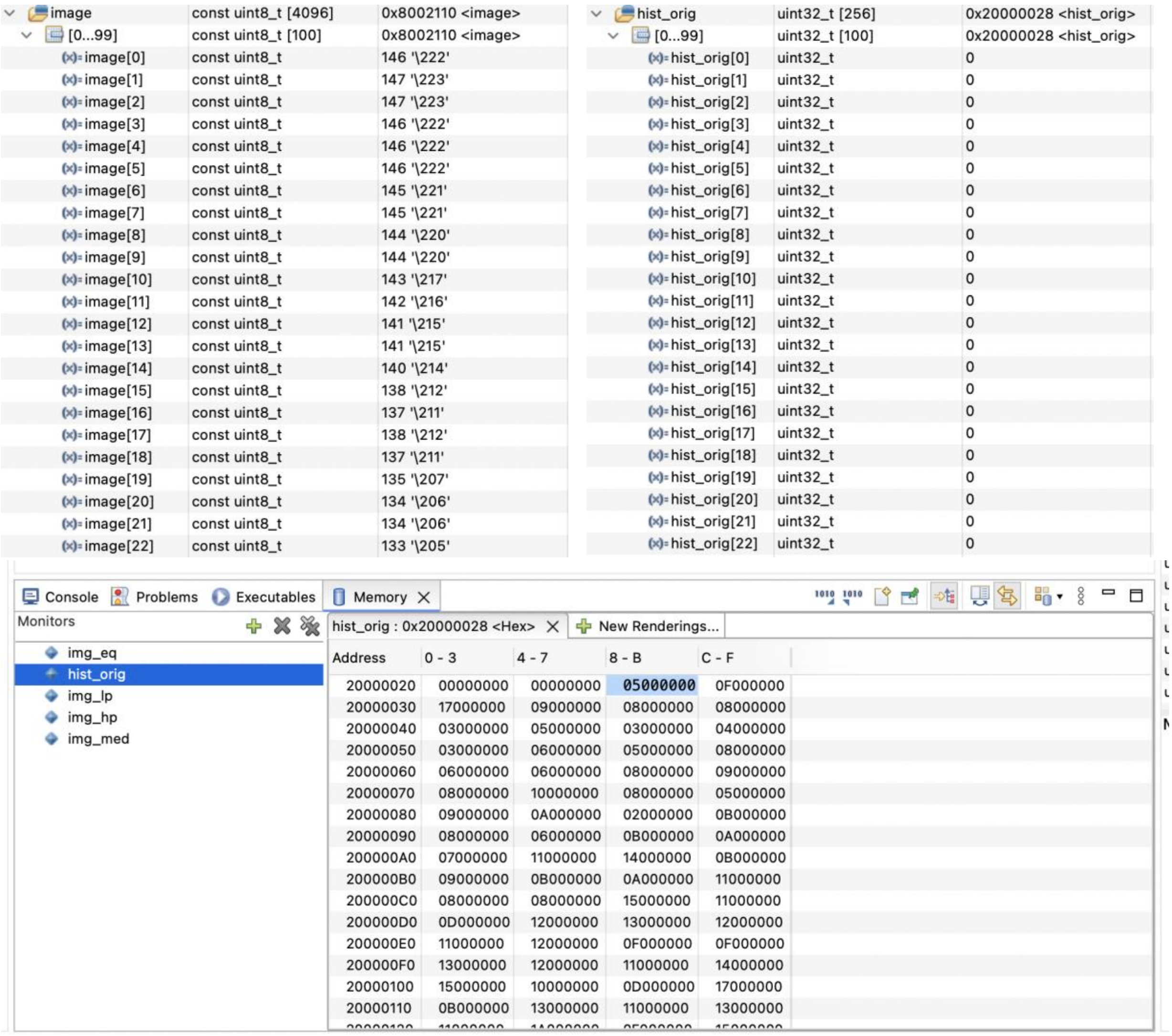


image	const uint8_t [4096]	0x8002110 <image>
[0...99]	const uint8_t [100]	0x8002110 <image>
image[0]	const uint8_t	146 '\222'
image[1]	const uint8_t	147 '\223'
image[2]	const uint8_t	147 '\223'
image[3]	const uint8_t	146 '\222'
image[4]	const uint8_t	146 '\222'
image[5]	const uint8_t	146 '\222'
image[6]	const uint8_t	145 '\221'
image[7]	const uint8_t	145 '\221'
image[8]	const uint8_t	144 '\220'
image[9]	const uint8_t	144 '\220'
image[10]	const uint8_t	143 '\217'
image[11]	const uint8_t	142 '\216'
image[12]	const uint8_t	141 '\215'
image[13]	const uint8_t	141 '\215'
image[14]	const uint8_t	140 '\214'
image[15]	const uint8_t	138 '\212'
image[16]	const uint8_t	137 '\211'
image[17]	const uint8_t	138 '\212'
image[18]	const uint8_t	137 '\211'
image[19]	const uint8_t	135 '\207'
image[20]	const uint8_t	134 '\206'
image[21]	const uint8_t	134 '\206'
image[22]	const uint8_t	133 '\205'

hist_orig	uint32_t [256]	0x20000028 <hist_orig>
[0...99]	uint32_t [100]	0x20000028 <hist_orig>
hist_orig[0]	uint32_t	0
hist_orig[1]	uint32_t	0
hist_orig[2]	uint32_t	0
hist_orig[3]	uint32_t	0
hist_orig[4]	uint32_t	0
hist_orig[5]	uint32_t	0
hist_orig[6]	uint32_t	0
hist_orig[7]	uint32_t	0
hist_orig[8]	uint32_t	0
hist_orig[9]	uint32_t	0
hist_orig[10]	uint32_t	0
hist_orig[11]	uint32_t	0
hist_orig[12]	uint32_t	0
hist_orig[13]	uint32_t	0
hist_orig[14]	uint32_t	0
hist_orig[15]	uint32_t	0
hist_orig[16]	uint32_t	0
hist_orig[17]	uint32_t	0
hist_orig[18]	uint32_t	0
hist_orig[19]	uint32_t	0
hist_orig[20]	uint32_t	0
hist_orig[21]	uint32_t	0
hist_orig[22]	uint32_t	0

Address	0 - 3	4 - 7	8 - B	C - F
20000020	00000000	00000000	05000000	0F000000
20000030	17000000	09000000	08000000	08000000
20000040	03000000	05000000	03000000	04000000
20000050	03000000	06000000	05000000	08000000
20000060	06000000	06000000	08000000	09000000
20000070	08000000	10000000	08000000	05000000
20000080	09000000	0A000000	02000000	0B000000
20000090	08000000	06000000	0B000000	0A000000
200000A0	07000000	11000000	14000000	0B000000
200000B0	09000000	0B000000	0A000000	11000000
200000C0	08000000	08000000	15000000	11000000
200000D0	0D000000	12000000	13000000	12000000
200000E0	11000000	12000000	0F000000	0F000000
200000F0	13000000	12000000	11000000	14000000
20000100	15000000	10000000	0D000000	17000000
20000110	0B000000	13000000	11000000	13000000

Question 2 – Histogram Equazation:

Steps implemented:

1. Compute histogram
2. Compute CDF:

$$\boldsymbol{CDF(k) = \sum_{i=0}^{k} Hist(i)}$$

3. Normalize:

$$T(k) = \frac{CDF(k) - CDF_{min}}{N - CDF_{min}} \cdot 255$$

4. Map each pixel through lookup table

Observed memory values (img_eq):

DD FF B8 FF

59 00 FE FF

FE FF 88 FF

18 00 23 00

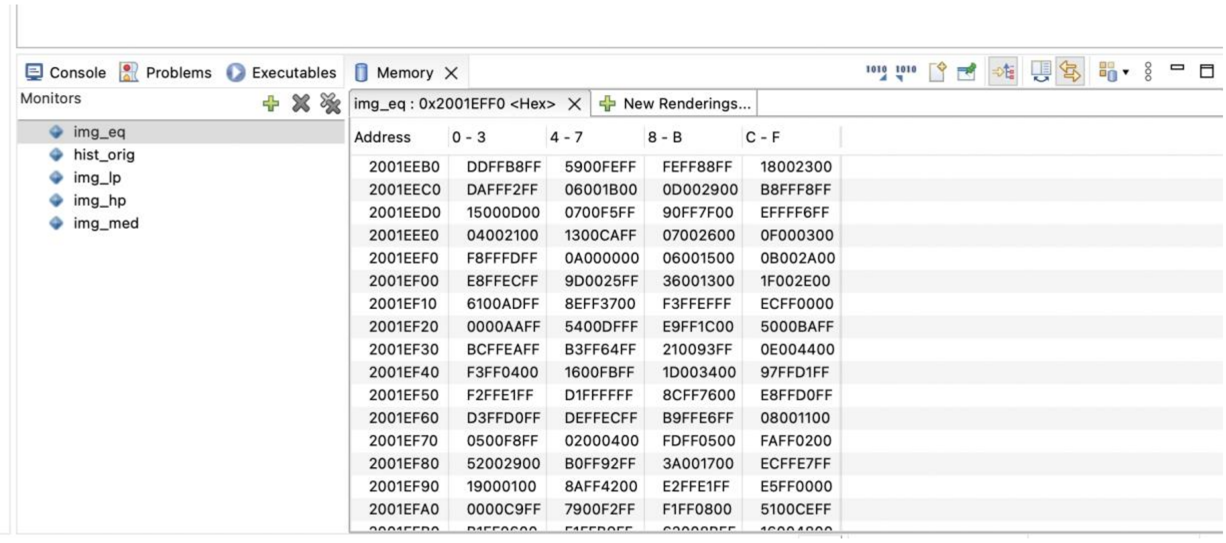
Interpretation:

- Repeated pixel values were stretched toward 255
- Rare pixel values pushed toward 0
- Full intensity range is now utilized

Contrast was successfully enhanced
The mapping reflects the expected transformation.

img_eq	uint8_t [4096]	0x2001eff0
[0...99]	uint8_t [100]	0x2001eff0
img_eq[0]	uint8_t	150 '\226'
img_eq[1]	uint8_t	152 '\230'
img_eq[2]	uint8_t	152 '\230'
img_eq[3]	uint8_t	150 '\226'
img_eq[4]	uint8_t	150 '\226'
img_eq[5]	uint8_t	150 '\226'
img_eq[6]	uint8_t	148 '\224'
img_eq[7]	uint8_t	148 '\224'
img_eq[8]	uint8_t	146 '\222'
img_eq[9]	uint8_t	146 '\222'
img_eq[10]	uint8_t	144 '\220'
img_eq[11]	uint8_t	142 '\216'
img_eq[12]	uint8_t	139 '\213'
img_eq[13]	uint8_t	139 '\213'
img_eq[14]	uint8_t	137 '\211'
img_eq[15]	uint8_t	131 '\203'
img_eq[16]	uint8_t	129 '\201'
img_eq[17]	uint8_t	131 '\203'
img_eq[18]	uint8_t	129 '\201'
img_eq[19]	uint8_t	124 '|'
img_eq[20]	uint8_t	122 'z'
img_eq[21]	uint8_t	122 'z'
img_eq[22]	uint8_t	120 'x'

img_eq[40]	uint8_t	104 'h'
img_eq[41]	uint8_t	103 'g'
img_eq[42]	uint8_t	103 'g'
img_eq[43]	uint8_t	103 'g'
img_eq[44]	uint8_t	103 'g'
img_eq[45]	uint8_t	102 'f'
img_eq[46]	uint8_t	100 'd'
img_eq[47]	uint8_t	100 'd'
img_eq[48]	uint8_t	99 'c'
img_eq[49]	uint8_t	99 'c'
img_eq[50]	uint8_t	99 'c'
img_eq[51]	uint8_t	99 'c'
img_eq[52]	uint8_t	103 'g'
img_eq[53]	uint8_t	124 '|'
img_eq[54]	uint8_t	142 '\216'
img_eq[55]	uint8_t	106 'j'
img_eq[56]	uint8_t	109 'm'
img_eq[57]	uint8_t	110 'n'
img_eq[58]	uint8_t	103 'g'
img_eq[59]	uint8_t	108 'l'
img_eq[60]	uint8_t	99 'c'



Address	0 - 3	4 - 7	8 - B	C - F
2001EEB0	DDFFB8FF	5900FEFF	FEFF88FF	18002300
2001EEC0	DAFFF2FF	06001B00	0D002900	B8FFF8FF
2001EED0	15000D00	0700F5FF	90FF7F00	EFFFF6FF
2001EEE0	04002100	1300CAFF	07002600	0F000300
2001EEF0	F8FFFDFF	0A000000	06001500	0B002A00
2001EF00	E8FFECFF	9D0025FF	36001300	1F002E00
2001EF10	6100ADFF	8EFF3700	F3FFEFFF	ECFF0000
2001EF20	0000AAFF	5400DFFF	E9FF1C00	5000BAFF
2001EF30	BCFFEAFF	B3FF64FF	210093FF	0E004400
2001EF40	F3FF0400	1600FBFF	1D003400	97FFD1FF
2001EF50	F2FFE1FF	D1FFFFFF	8CFF7600	E8FFD0FF
2001EF60	D3FFD0FF	DEFFECFF	B9FFE6FF	08001100
2001EF70	0500F8FF	02000400	FDFF0500	FAFF0200
2001EF80	52002900	B0FF92FF	3A001700	ECFFE7FF
2001EF90	19000100	8AFF4200	E2FFE1FF	E5FF0000
2001EFA0	0000C9FF	7900F2FF	F1FF0800	5100CEFF

Question 3 – Lowpass filtering:

A 3×3 averaging kernel was applied:

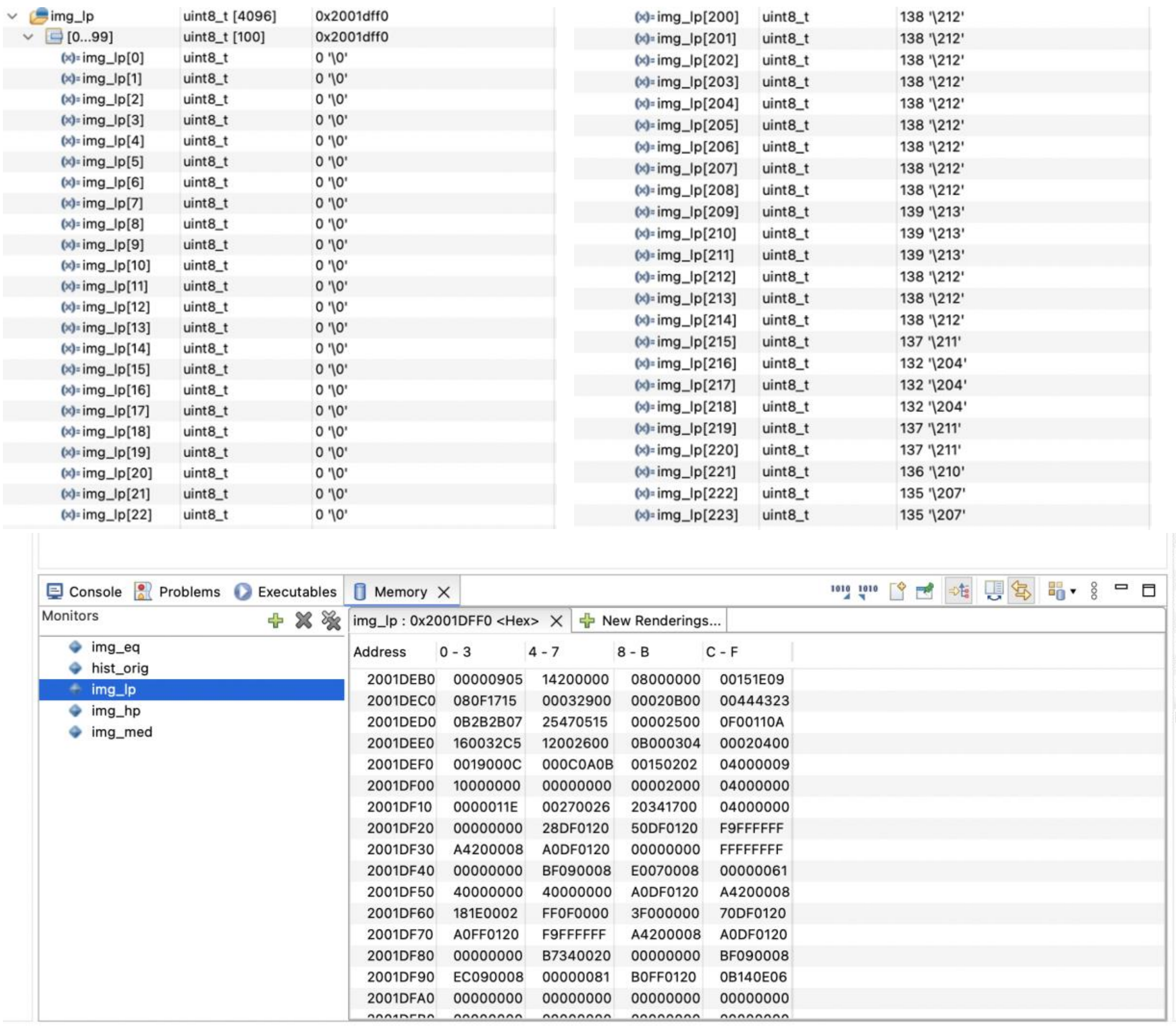
$$\boldsymbol{k} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Example memory region (img_lp):
00000905
080F1715
0B2B2B07
160032C5
...

Observations:

- Neighboring pixels have similar values
- No large variations or sharp edges
- Bit patterns are smooth and low-frequency

The result is blurred as expected
Noise and edges are suppressed



Question 3 – Highpass filtering:

A Laplacian kernel was used:

$$k = \begin{bmatrix} \mathbf{0} & \mathbf{-1} & \mathbf{0} \\ \mathbf{-1} & \mathbf{4} & \mathbf{-1} \\ \mathbf{0} & \mathbf{-1} & \mathbf{0} \end{bmatrix}$$

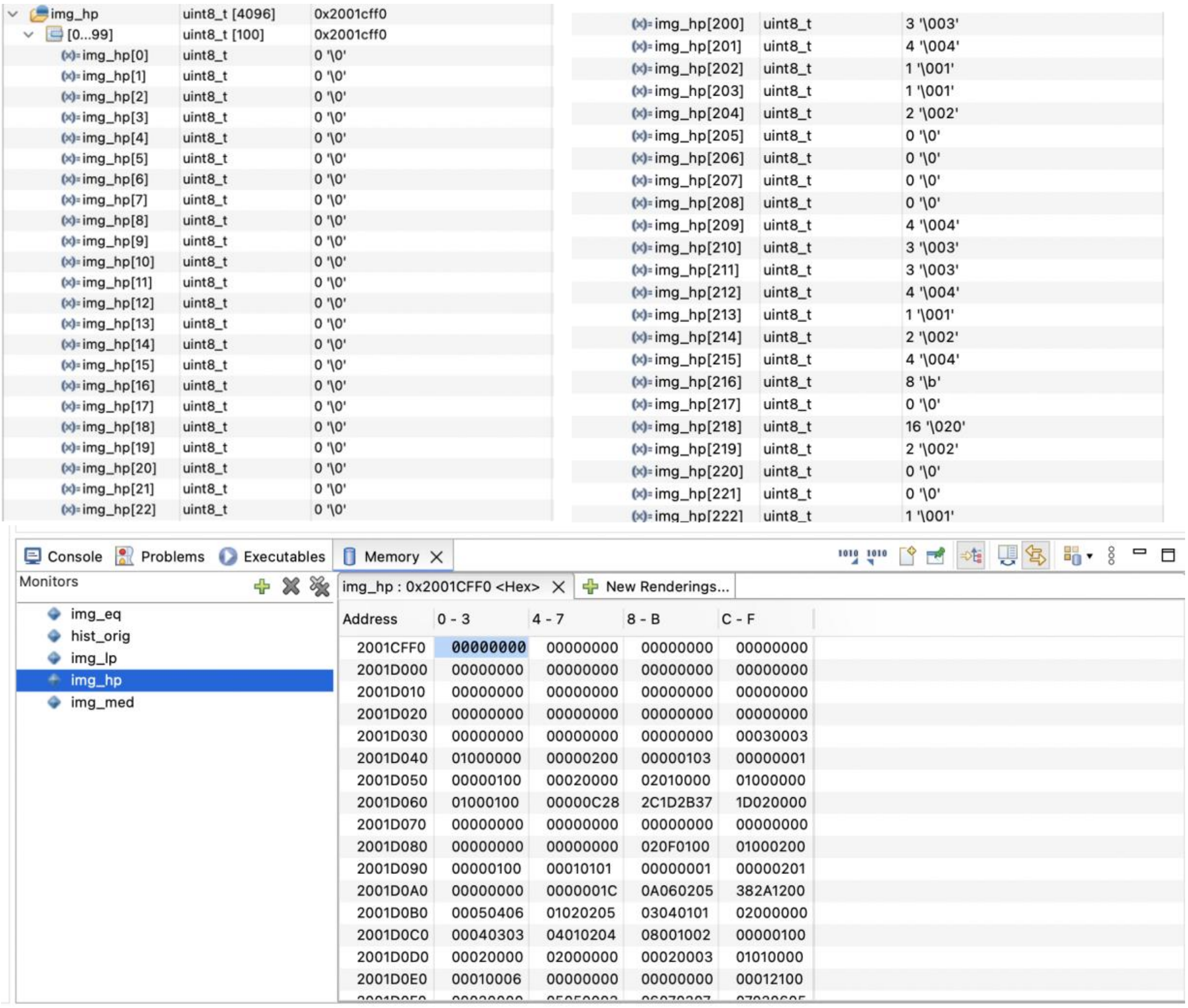
Example (img_hp):

00000000
00000000
00000000
...
00000103
01000200
...

Interpretation:

- Flat areas → 0
- Edges → small positive values
- Edge map correctly obtained

High-pass behavior is correct
Only edges remain



Question 4 – Median Filtering:

A 3×3 sliding window per pixel:

- Window values sorted
- Median (5th element) chosen

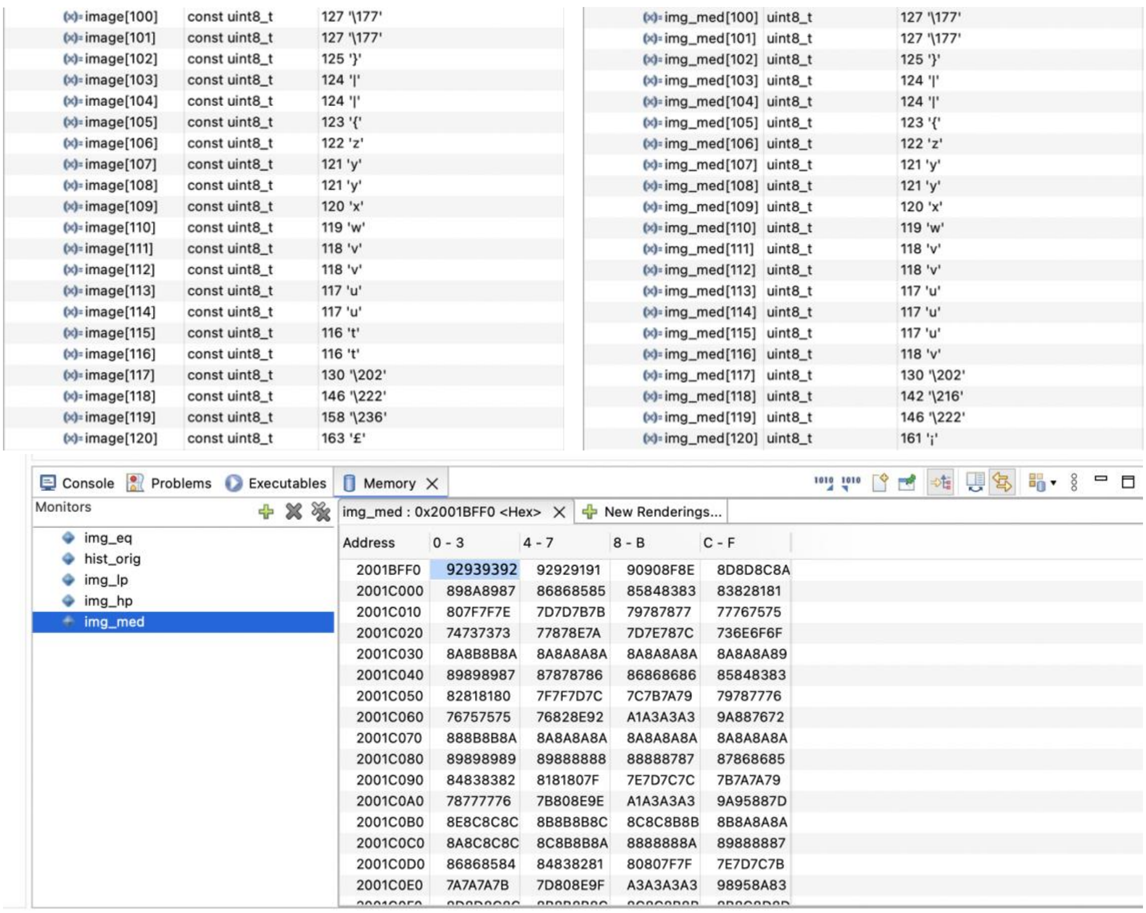
Example (img_med):

92 93 93 92
89 88 89 87
80 7F 7F 7E
...

Result behavior:

- Salt-and-pepper noise removed
- Values remain locally smooth
- Edges preserved better than blur

Median filter successfully implemented
Memory output matches theory



1. Conclusion

In this experiment, several fundamental image-processing techniques—histogram computation, histogram equalization, spatial low-pass and high-pass filtering, and median filtering—were successfully implemented and executed on the STM32 microcontroller. Each processing stage produced results consistent with theoretical expectations, as verified through memory-based inspection during debugging. The histogram analysis correctly reflected the original image's grayscale distribution, while histogram equalization enhanced contrast by effectively redistributing pixel intensities. Low-pass filtering smoothed the image and reduced high-frequency variations, whereas high-pass filtering emphasized edges and fine structural details by suppressing low-frequency components. The median filter preserved the original image structure, demonstrating its effectiveness in eliminating impulsive noise while maintaining edges. Overall, the results confirm that the designed functions are fully operational on embedded hardware, and the microcontroller platform can perform essential image-processing tasks reliably and efficiently.