



MARMARA UNIVERSITY
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.

The screenshot shows a debugger's memory dump and monitors view.

Memory Dump:

Symbol	Type	Value	Address	Type	Value	Address
image	const uint8_t [4096]	0x8002110 <image>	0x8002110	const uint8_t [100]	0x8002110 <image>	0x8002110
image[0]	const uint8_t	146 '\222'	0x8002111	const uint8_t	147 '\223'	0x8002112
image[1]	const uint8_t	147 '\223'	0x8002113	const uint8_t	147 '\223'	0x8002114
image[2]	const uint8_t	146 '\222'	0x8002115	const uint8_t	146 '\222'	0x8002116
image[3]	const uint8_t	146 '\222'	0x8002117	const uint8_t	146 '\222'	0x8002118
image[4]	const uint8_t	146 '\222'	0x8002119	const uint8_t	146 '\222'	0x800211A
image[5]	const uint8_t	146 '\222'	0x800211B	const uint8_t	145 '\221'	0x800211C
image[6]	const uint8_t	145 '\221'	0x800211D	const uint8_t	145 '\221'	0x800211E
image[7]	const uint8_t	145 '\221'	0x800211F	const uint8_t	144 '\220'	0x800211G
image[8]	const uint8_t	144 '\220'	0x800211H	const uint8_t	144 '\220'	0x800211I
image[9]	const uint8_t	144 '\220'	0x800211J	const uint8_t	143 '\217'	0x800211K
image[10]	const uint8_t	143 '\217'	0x800211L	const uint8_t	142 '\216'	0x800211M
image[11]	const uint8_t	142 '\216'	0x800211N	const uint8_t	141 '\215'	0x800211O
image[12]	const uint8_t	141 '\215'	0x800211P	const uint8_t	141 '\215'	0x800211Q
image[13]	const uint8_t	140 '\214'	0x800211R	const uint8_t	138 '\212'	0x800211S
image[14]	const uint8_t	140 '\214'	0x800211T	const uint8_t	137 '\211'	0x800211U
image[15]	const uint8_t	138 '\212'	0x800211V	const uint8_t	138 '\212'	0x800211W
image[16]	const uint8_t	137 '\211'	0x800211X	const uint8_t	137 '\211'	0x800211Y
image[17]	const uint8_t	138 '\212'	0x800211Z	const uint8_t	137 '\211'	0x800211AA
image[18]	const uint8_t	137 '\211'	0x800211AB	const uint8_t	135 '\207'	0x800211AC
image[19]	const uint8_t	135 '\207'	0x800211AD	const uint8_t	134 '\206'	0x800211AE
image[20]	const uint8_t	134 '\206'	0x800211AF	const uint8_t	134 '\206'	0x800211AG
image[21]	const uint8_t	134 '\206'	0x800211AH	const uint8_t	133 '\205'	0x800211AI
image[22]	const uint8_t	133 '\205'	0x800211AJ	hist_orig	uint32_t [256]	0x20000028 <hist_orig>
			0x800211AK	[0...99]	uint32_t [100]	0x20000028 <hist_orig>
			0x800211AL	(*)=hist_orig[0]	uint32_t	0
			0x800211AM	(*)=hist_orig[1]	uint32_t	0
			0x800211AN	(*)=hist_orig[2]	uint32_t	0
			0x800211AO	(*)=hist_orig[3]	uint32_t	0
			0x800211AP	(*)=hist_orig[4]	uint32_t	0
			0x800211AQ	(*)=hist_orig[5]	uint32_t	0
			0x800211AR	(*)=hist_orig[6]	uint32_t	0
			0x800211AS	(*)=hist_orig[7]	uint32_t	0
			0x800211AT	(*)=hist_orig[8]	uint32_t	0
			0x800211AU	(*)=hist_orig[9]	uint32_t	0
			0x800211AV	(*)=hist_orig[10]	uint32_t	0
			0x800211AW	(*)=hist_orig[11]	uint32_t	0
			0x800211AX	(*)=hist_orig[12]	uint32_t	0
			0x800211AY	(*)=hist_orig[13]	uint32_t	0
			0x800211AZ	(*)=hist_orig[14]	uint32_t	0
			0x800211B0	(*)=hist_orig[15]	uint32_t	0
			0x800211B1	(*)=hist_orig[16]	uint32_t	0
			0x800211B2	(*)=hist_orig[17]	uint32_t	0
			0x800211B3	(*)=hist_orig[18]	uint32_t	0
			0x800211B4	(*)=hist_orig[19]	uint32_t	0
			0x800211B5	(*)=hist_orig[20]	uint32_t	0
			0x800211B6	(*)=hist_orig[21]	uint32_t	0
			0x800211B7	(*)=hist_orig[22]	uint32_t	0

Monitors:

Symbol	Address	0 - 3	4 - 7	8 - B	C - F
img_eq	20000020	00000000	00000000	05000000	0F000000
hist_orig	20000030	17000000	09000000	08000000	08000000
img_lp	20000040	03000000	05000000	03000000	04000000
img_hp	20000050	03000000	06000000	05000000	08000000
img_med	20000060	06000000	06000000	08000000	09000000

Question 2 – Histogram Equazation:

Steps implemented:

1. Compute histogram
2. Compute CDF:

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

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	(*)=img_eq[40]	uint8_t	104 'h'
[...99]	uint8_t [100]	0x2001eff0	(*)=img_eq[41]	uint8_t	103 'g'
(*)=img_eq[0]	uint8_t	150 '226'	(*)=img_eq[42]	uint8_t	103 'g'
(*)=img_eq[1]	uint8_t	152 '230'	(*)=img_eq[43]	uint8_t	103 'g'
(*)=img_eq[2]	uint8_t	152 '230'	(*)=img_eq[44]	uint8_t	103 'g'
(*)=img_eq[3]	uint8_t	150 '226'	(*)=img_eq[45]	uint8_t	102 'f'
(*)=img_eq[4]	uint8_t	150 '226'	(*)=img_eq[46]	uint8_t	100 'd'
(*)=img_eq[5]	uint8_t	150 '226'	(*)=img_eq[47]	uint8_t	100 'd'
(*)=img_eq[6]	uint8_t	148 '224'	(*)=img_eq[48]	uint8_t	99 'c'
(*)=img_eq[7]	uint8_t	148 '224'	(*)=img_eq[49]	uint8_t	99 'c'
(*)=img_eq[8]	uint8_t	146 '222'	(*)=img_eq[50]	uint8_t	99 'c'
(*)=img_eq[9]	uint8_t	146 '222'	(*)=img_eq[51]	uint8_t	99 'c'
(*)=img_eq[10]	uint8_t	144 '220'	(*)=img_eq[52]	uint8_t	103 'g'
(*)=img_eq[11]	uint8_t	142 '216'	(*)=img_eq[53]	uint8_t	124 'l'
(*)=img_eq[12]	uint8_t	139 '213'	(*)=img_eq[54]	uint8_t	142 '216'
(*)=img_eq[13]	uint8_t	139 '213'	(*)=img_eq[55]	uint8_t	106 'j'
(*)=img_eq[14]	uint8_t	137 '211'	(*)=img_eq[56]	uint8_t	109 'm'
(*)=img_eq[15]	uint8_t	131 '203'	(*)=img_eq[57]	uint8_t	110 'n'
(*)=img_eq[16]	uint8_t	129 '201'	(*)=img_eq[58]	uint8_t	103 'g'
(*)=img_eq[17]	uint8_t	131 '203'	(*)=img_eq[59]	uint8_t	108 'l'
(*)=img_eq[18]	uint8_t	129 '201'	(*)=img_eq[60]	uint8_t	99 'c'
(*)=img_eq[19]	uint8_t	124 'l'			
(*)=img_eq[20]	uint8_t	122 'z'			
(*)=img_eq[21]	uint8_t	122 'z'			
(*)=img_eq[22]	uint8_t	120 'x'			

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	F8FFFFFF	0A000000	06001500	0B002AA0
2001EF00	E8FFECFF	9D0025FF	36001300	1F002E00
2001EF10	6100ADFF	8EFF3700	F3FFEFFF	ECFF0000
2001EF20	0000AAFF	5400DFFF	E9FF1C00	5000BAFF
2001EF30	BCFFE AFF	B3FF64FF	210093FF	0E004400
2001EF40	F3FF0400	1600FBFF	1D003400	97FFD1FF
2001EF50	F2FFE1FF	D1FFFFFF	8CF7600	E8FFD0FF
2001EF60	D3FFD0FF	DEFFECFF	B9FFE6FF	08001100
2001EF70	0500F8FF	02000400	FDF0500	FAFF0200
2001EF80	52002900	B0FF92FF	3A001700	ECFFE7FF
2001EF90	19000100	8AFF4200	E2FFE1FF	E5FF0000
2001EFA0	0000C9FF	7900F2FF	F1FF0800	5100CEFF
2001EEF00	00000000	000000FF	000000FF	10001000

Question 3 – Lowpass filtering:

A 3×3 averaging kernel was applied:

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

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

↳ img_lp	uint8_t [4096]	0x2001dff0		↳ img_lp[200]	uint8_t	138 '212'
↳ [0..99]	uint8_t [100]	0x2001dff0		↳ img_lp[201]	uint8_t	138 '212'
↳ img_lp[0]	uint8_t	0 '0'		↳ img_lp[202]	uint8_t	138 '212'
↳ img_lp[1]	uint8_t	0 '0'		↳ img_lp[203]	uint8_t	138 '212'
↳ img_lp[2]	uint8_t	0 '0'		↳ img_lp[204]	uint8_t	138 '212'
↳ img_lp[3]	uint8_t	0 '0'		↳ img_lp[205]	uint8_t	138 '212'
↳ img_lp[4]	uint8_t	0 '0'		↳ img_lp[206]	uint8_t	138 '212'
↳ img_lp[5]	uint8_t	0 '0'		↳ img_lp[207]	uint8_t	138 '212'
↳ img_lp[6]	uint8_t	0 '0'		↳ img_lp[208]	uint8_t	138 '212'
↳ img_lp[7]	uint8_t	0 '0'		↳ img_lp[209]	uint8_t	139 '213'
↳ img_lp[8]	uint8_t	0 '0'		↳ img_lp[210]	uint8_t	139 '213'
↳ img_lp[9]	uint8_t	0 '0'		↳ img_lp[211]	uint8_t	139 '213'
↳ img_lp[10]	uint8_t	0 '0'		↳ img_lp[212]	uint8_t	138 '212'
↳ img_lp[11]	uint8_t	0 '0'		↳ img_lp[213]	uint8_t	138 '212'
↳ img_lp[12]	uint8_t	0 '0'		↳ img_lp[214]	uint8_t	138 '212'
↳ img_lp[13]	uint8_t	0 '0'		↳ img_lp[215]	uint8_t	137 '211'
↳ img_lp[14]	uint8_t	0 '0'		↳ img_lp[216]	uint8_t	132 '204'
↳ img_lp[15]	uint8_t	0 '0'		↳ img_lp[217]	uint8_t	132 '204'
↳ img_lp[16]	uint8_t	0 '0'		↳ img_lp[218]	uint8_t	132 '204'
↳ img_lp[17]	uint8_t	0 '0'		↳ img_lp[219]	uint8_t	137 '211'
↳ img_lp[18]	uint8_t	0 '0'		↳ img_lp[220]	uint8_t	137 '211'
↳ img_lp[19]	uint8_t	0 '0'		↳ img_lp[221]	uint8_t	136 '210'
↳ img_lp[20]	uint8_t	0 '0'		↳ img_lp[222]	uint8_t	135 '207'
↳ img_lp[21]	uint8_t	0 '0'		↳ img_lp[223]	uint8_t	135 '207'
↳ img_lp[22]	uint8_t	0 '0'				

Memory					
Monitors	img_lp : 0x2001DF00 <Hex>				New Renderings...
	Address	0 - 3	4 - 7	8 - B	C - F
img_eq	2001DEB0	00000905	14200000	08000000	00151E09
hist_orig	2001DEC0	080F1715	00032900	00020B00	00444323
img_lp	2001DED0	0B2B2B07	25470515	00002500	0F00110A
img_hp	2001DEE0	160032C5	12002600	0B000304	00020400
img_med	2001DEF0	0019000C	000C0A0B	00150202	04000009
	2001DF00	10000000	00000000	00002000	04000000
	2001DF10	0000001E	00270026	20341700	04000000
	2001DF20	00000000	28DF0120	50DF0120	F9FFFFFF
	2001DF30	A4200008	A0DF0120	00000000	FFFFFFF
	2001DF40	00000000	BFO90008	E0070008	00000061
	2001DF50	40000000	40000000	A0DF0120	A4200008
	2001DF60	181E0002	FF0F0000	3F000000	70DF0120
	2001DF70	A0FF0120	F9FFFFFF	A4200008	A0DF0120
	2001DF80	00000000	B7340020	00000000	BF090008
	2001DF90	EC090008	00000081	B0FF0120	0B140E06
	2001DFA0	00000000	00000000	00000000	00000000
	2001DFB0	00000000	00000000	00000000	00000000

Question 3 – Highpass filtering:

A Laplacian kernel was used:

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

Example (img_hp):

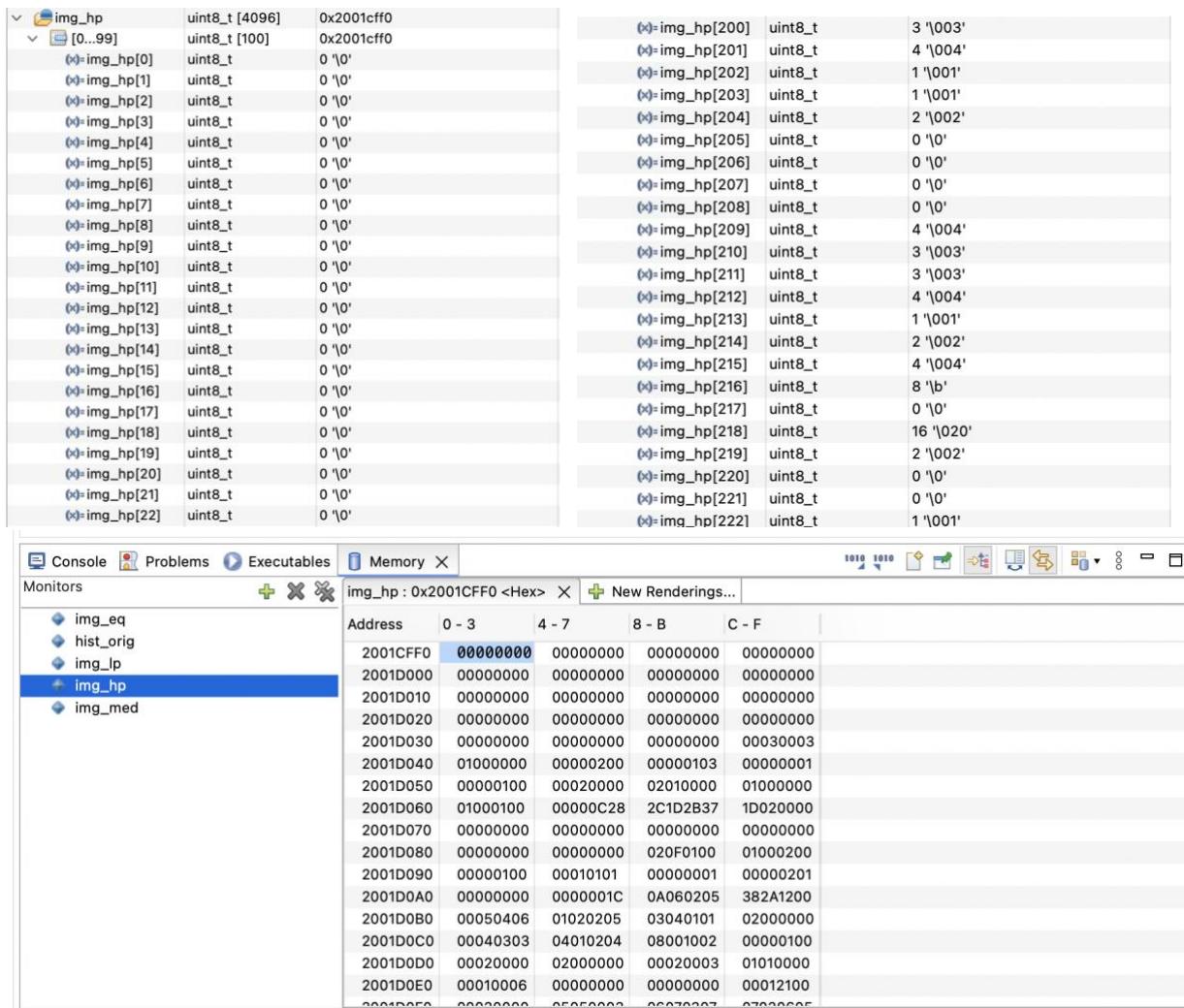
00000000
00000000
00000000
...
00000103
01000200

Interpretation:

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

High-pass behavior is correct

Only edges remain



The screenshot shows the Eclipse IDE interface with two tables side-by-side and a monitor table below them.

Left Table:

Address	Type	Value
0x2001cff0	uint8_t [4096]	0x2001cff0
[0...99]	uint8_t [100]	0x2001cff0
(0):img_hp[0]	uint8_t	0 '\0'
(0):img_hp[1]	uint8_t	0 '\0'
(0):img_hp[2]	uint8_t	0 '\0'
(0):img_hp[3]	uint8_t	0 '\0'
(0):img_hp[4]	uint8_t	0 '\0'
(0):img_hp[5]	uint8_t	0 '\0'
(0):img_hp[6]	uint8_t	0 '\0'
(0):img_hp[7]	uint8_t	0 '\0'
(0):img_hp[8]	uint8_t	0 '\0'
(0):img_hp[9]	uint8_t	0 '\0'
(0):img_hp[10]	uint8_t	0 '\0'
(0):img_hp[11]	uint8_t	0 '\0'
(0):img_hp[12]	uint8_t	0 '\0'
(0):img_hp[13]	uint8_t	0 '\0'
(0):img_hp[14]	uint8_t	0 '\0'
(0):img_hp[15]	uint8_t	0 '\0'
(0):img_hp[16]	uint8_t	0 '\0'
(0):img_hp[17]	uint8_t	0 '\0'
(0):img_hp[18]	uint8_t	0 '\0'
(0):img_hp[19]	uint8_t	0 '\0'
(0):img_hp[20]	uint8_t	0 '\0'
(0):img_hp[21]	uint8_t	0 '\0'
(0):img_hp[22]	uint8_t	0 '\0'

Right Table:

Address	Type	Value
(0):img_hp[200]	uint8_t	3 '\003'
(0):img_hp[201]	uint8_t	4 '\004'
(0):img_hp[202]	uint8_t	1 '\001'
(0):img_hp[203]	uint8_t	1 '\001'
(0):img_hp[204]	uint8_t	2 '\002'
(0):img_hp[205]	uint8_t	0 '\0'
(0):img_hp[206]	uint8_t	0 '\0'
(0):img_hp[207]	uint8_t	0 '\0'
(0):img_hp[208]	uint8_t	0 '\0'
(0):img_hp[209]	uint8_t	4 '\004'
(0):img_hp[210]	uint8_t	3 '\003'
(0):img_hp[211]	uint8_t	3 '\003'
(0):img_hp[212]	uint8_t	4 '\004'
(0):img_hp[213]	uint8_t	1 '\001'
(0):img_hp[214]	uint8_t	2 '\002'
(0):img_hp[215]	uint8_t	4 '\004'
(0):img_hp[216]	uint8_t	8 '\b'
(0):img_hp[217]	uint8_t	0 '\0'
(0):img_hp[218]	uint8_t	16 '\020'
(0):img_hp[219]	uint8_t	2 '\002'
(0):img_hp[220]	uint8_t	0 '\0'
(0):img_hp[221]	uint8_t	0 '\0'
(0):img_hp[222]	uint8_t	1 '\001'

Monitor Table:

Monitors	Address	0 - 3	4 - 7	8 - B	C - F
img_eq	2001CFF0	00000000	00000000	00000000	00000000
hist_orig	2001D000	00000000	00000000	00000000	00000000
img_lp	2001D010	00000000	00000000	00000000	00000000
img_hp	2001D020	00000000	00000000	00000000	00000000
	2001D030	00000000	00000000	00000000	00030003
	2001D040	01000000	00000200	00000103	00000001
	2001D050	00000100	00020000	02010000	01000000
	2001D060	01000100	00000C28	2C1D2B37	1D020000
	2001D070	00000000	00000000	00000000	00000000
	2001D080	00000000	00000000	020F0100	01000200
	2001D090	00000100	00010101	00000001	00000201
	2001D0A0	00000000	00000001C	0A060205	382A1200
	2001D0B0	00050406	01020205	03040101	02000000
	2001D0C0	00040303	04010204	08001002	00000100
	2001D0D0	00020000	02000000	00020003	01010000
	2001D0E0	00010006	00000000	00000000	00012100
	2001D0F0	00000000	00000000	00000000	00000000

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

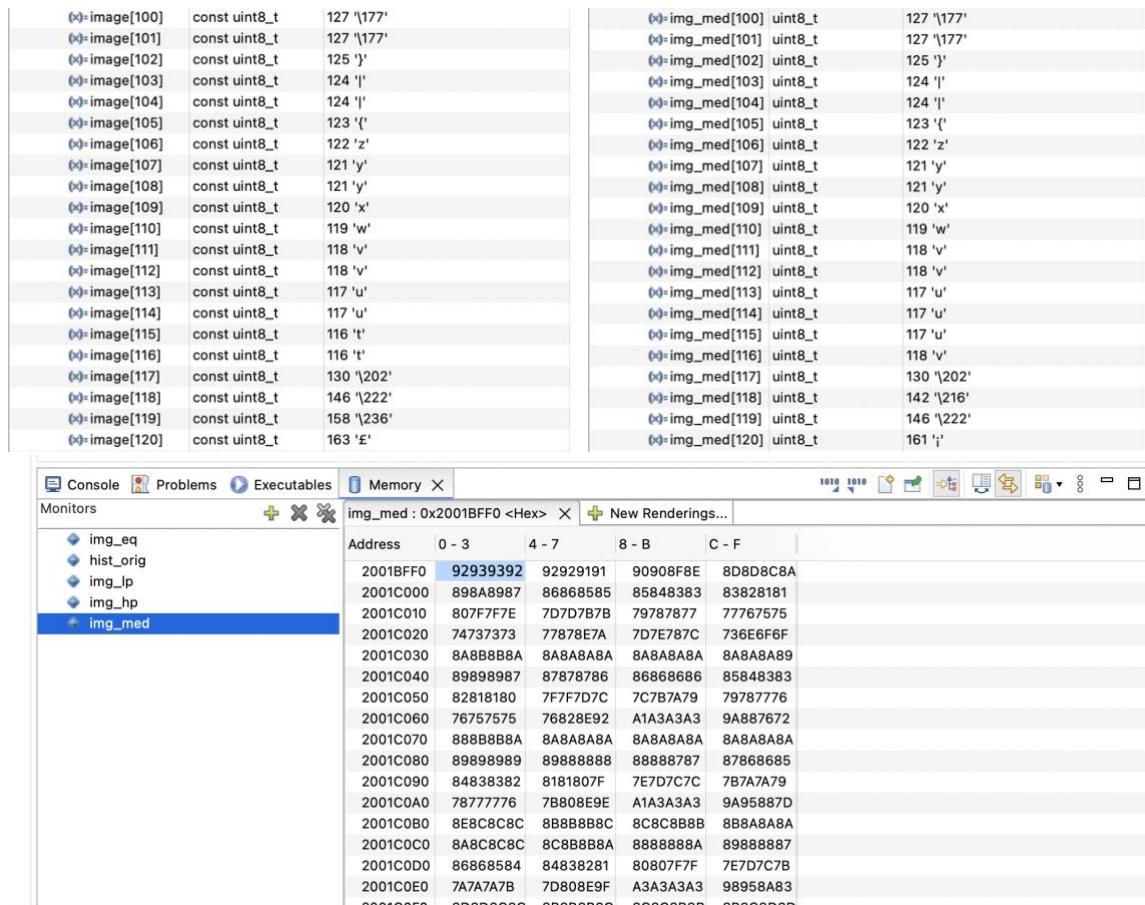
...

Result behavior:

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

Median filter successfully implemented

Memory output matches theory



(*)>image[100]	const uint8_t	127 '\177'	(*)>img_med[100]	uint8_t	127 '\177'
(*)>image[101]	const uint8_t	127 '\177'	(*)>img_med[101]	uint8_t	127 '\177'
(*)>image[102]	const uint8_t	125 '\'	(*)>img_med[102]	uint8_t	125 '\'
(*)>image[103]	const uint8_t	124 '\ '	(*)>img_med[103]	uint8_t	124 '\ '
(*)>image[104]	const uint8_t	124 '\ '	(*)>img_med[104]	uint8_t	124 '\ '
(*)>image[105]	const uint8_t	123 '\{'	(*)>img_med[105]	uint8_t	123 '\{'
(*)>image[106]	const uint8_t	122 '\z'	(*)>img_med[106]	uint8_t	122 '\z'
(*)>image[107]	const uint8_t	121 '\y'	(*)>img_med[107]	uint8_t	121 '\y'
(*)>image[108]	const uint8_t	121 '\y'	(*)>img_med[108]	uint8_t	121 '\y'
(*)>image[109]	const uint8_t	120 '\x'	(*)>img_med[109]	uint8_t	120 '\x'
(*)>image[110]	const uint8_t	119 '\w'	(*)>img_med[110]	uint8_t	119 '\w'
(*)>image[111]	const uint8_t	118 '\v'	(*)>img_med[111]	uint8_t	118 '\v'
(*)>image[112]	const uint8_t	118 '\v'	(*)>img_med[112]	uint8_t	118 '\v'
(*)>image[113]	const uint8_t	117 '\u'	(*)>img_med[113]	uint8_t	117 '\u'
(*)>image[114]	const uint8_t	117 '\u'	(*)>img_med[114]	uint8_t	117 '\u'
(*)>image[115]	const uint8_t	116 '\t'	(*)>img_med[115]	uint8_t	117 '\u'
(*)>image[116]	const uint8_t	116 '\t'	(*)>img_med[116]	uint8_t	118 '\v'
(*)>image[117]	const uint8_t	130 '\202'	(*)>img_med[117]	uint8_t	130 '\202'
(*)>image[118]	const uint8_t	146 '\222'	(*)>img_med[118]	uint8_t	142 '\216'
(*)>image[119]	const uint8_t	158 '\236'	(*)>img_med[119]	uint8_t	146 '\222'
(*)>image[120]	const uint8_t	163 '\e'	(*)>img_med[120]	uint8_t	161 '\i'

img_med : 0x2001BFF0 <Hex> X New Renderings...					
Address	0 - 3	4 - 7	8 - B	C - F	
2001BFF0	92939392	92929191	90908F8E	8D8D8C8A	
2001C000	898A8987	86868585	85848383	83828181	
2001C010	807F7F7E	7D7D7B7B	79787877	77767575	
2001C020	74737373	77878E7A	7D7E787C	736E6F6F	
2001C030	8A8BBB8A	8A8ABA8A	8A8ABA8A	8A8ABA89	
2001C040	89898987	87878786	86868686	85848383	
2001C050	82818180	7F7F7D7C	7C7B7A79	79787776	
2001C060	76757575	76828E92	A1A3A3A3	9A887672	
2001C070	888BBB8A	8A8ABA8A	8A8ABA8A	8A8ABA8A	
2001C080	89898989	89888888	88888787	87868685	
2001C090	84838382	8181807F	7E7D7C7C	7B7A7A79	
2001C0A0	78777776	7B808E9E	A1A3A3A3	9A95887D	
2001C0B0	8E8C8C8C	8B8BBB8C	8C8C8B8B	8B8A8A8A	
2001C0C0	8A8C8C8C	8C888B8A	8888888A	89888887	
2001C0D0	86868584	84838281	80807F7F	7E7D7C7B	
2001C0E0	7A7A7A7B	7D808E9F	A3A3A3A3	98958A83	
2001C0F0	88888888	88888888	88888888	88888888	

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.