Marmara University

Faculty of Engineering

Electrical and Electronics Engineering

EE4065.1: INTRODUCTION TO EMBEDDED IMAGE PROCESSING#

Homework #2 Report & Results

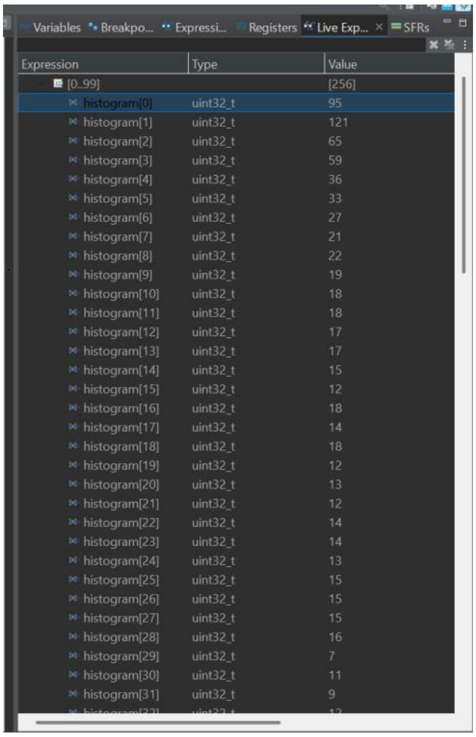
Name Surname	Student ID
Yusuf Yiğit Söyleyici	150723524
Mehmet Açar	150719020

Q1) Histogram Formation

a) C Function for Histogram Calculation The following function was implemented to calculate the histogram of a grayscale image. It initializes a 256-element array to zero and iterates through every pixel in the 128x96 image buffer, incrementing the count for the corresponding intensity value.

```c
void CalculateHistogram(IMAGE_HandleTypeDef *pImg, uint32_t *pHist)
{

    for(int i = 0; i < 256; i++)
    {
        pHist[i] = 0;
    }

    uint32_t totalPixels = pImg->width * pImg->height;

    for(uint32_t i = 0; i < totalPixels; i++)
    {
        uint8_t intensity = pImg->pData[i];

        pHist[intensity]++;
    }
}
```

b) Histogram Results (STM32CubeIDE) The image was transferred from the PC to the STM32 via UART. A breakpoint was placed after the CalculateHistogram function execution. The histogram array was inspected in the "Live Expressions" tab.


Variables Breakpo... Expressi... Registers Live Exp... SFRs

Expression	Type	Value
[0..99]		[256]
histogram[0]	uint32_t	95
histogram[1]	uint32_t	121
histogram[2]	uint32_t	65
histogram[3]	uint32_t	59
histogram[4]	uint32_t	36
histogram[5]	uint32_t	33
histogram[6]	uint32_t	27
histogram[7]	uint32_t	21
histogram[8]	uint32_t	22
histogram[9]	uint32_t	19
histogram[10]	uint32_t	18
histogram[11]	uint32_t	18
histogram[12]	uint32_t	17
histogram[13]	uint32_t	17
histogram[14]	uint32_t	15
histogram[15]	uint32_t	12
histogram[16]	uint32_t	18
histogram[17]	uint32_t	14
histogram[18]	uint32_t	18
histogram[19]	uint32_t	12
histogram[20]	uint32_t	13
histogram[21]	uint32_t	12
histogram[22]	uint32_t	14
histogram[23]	uint32_t	14
histogram[24]	uint32_t	13
histogram[25]	uint32_t	15
histogram[26]	uint32_t	15
histogram[27]	uint32_t	15
histogram[28]	uint32_t	16
histogram[29]	uint32_t	7
histogram[30]	uint32_t	11
histogram[31]	uint32_t	9

Q2) Histogram Equalization

a) Derivation of Method

$$\text{pdf of histogram } (k) = \frac{\#\text{ of } k \text{ valued pixels} \rightarrow n}{\#\text{ of total pixels} \rightarrow M}$$

$$CDF = 255 \sum_{j=0}^{k} \frac{n}{M} \quad \left(\begin{array}{l} \text{new value of } k \text{ valued pixels} \\ \text{after histogram equalization applied} \end{array} \right)$$

→ ~~lets say~~ from prob theory, if we have transformation $s = T(r)$, the probability density of the output variable s, denoted as $P_s(s)$ is related to input by

$$P_s(s) = P_r(r) \left| \frac{dr}{ds} \right|$$

$$\frac{ds}{dr} = \frac{d}{dr}\left[255 \int_0^r P_r(w)\,dw \right] = 255\, P_r(r)$$

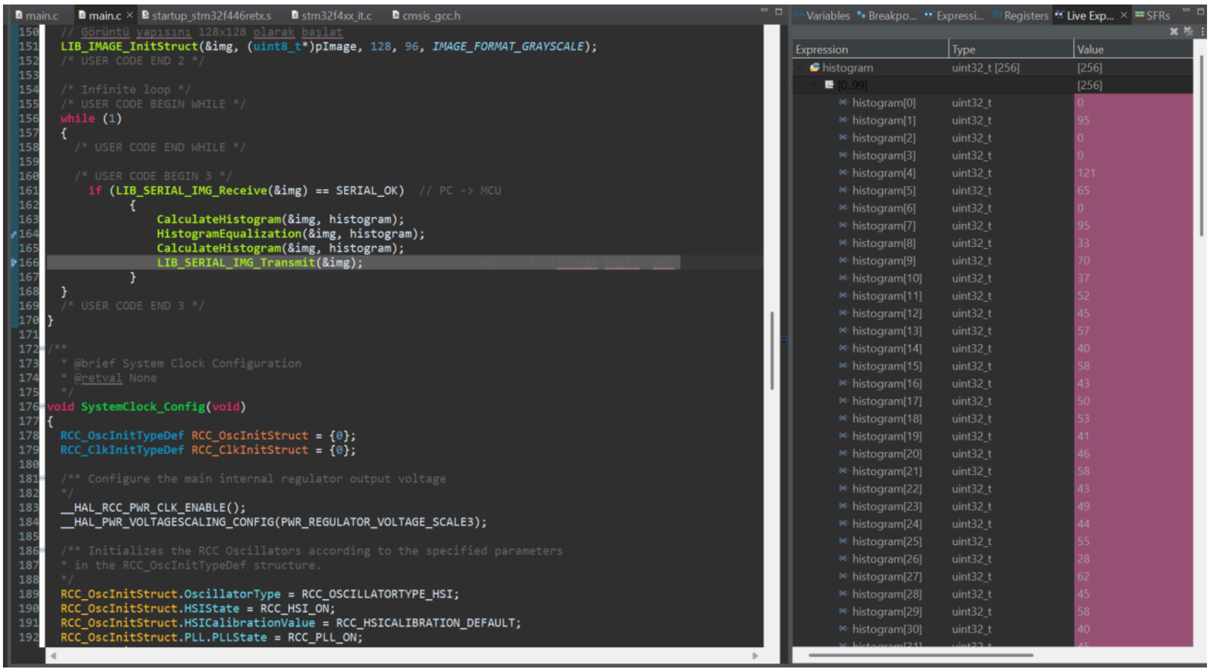
$$\Rightarrow P_s(s) = \cancel{P_r(r)} \left| \frac{1}{255\, \cancel{P_r(r)}} \right| = \frac{1}{255}$$

→ since $P_s(s)$ is a constant, the probability of every value in the output is equal. This means the output histogram is perfectly flat (uniform).

b) C Function for Histogram Equalization The equalization function first calculates the Cumulative Distribution Function (CDF) of the histogram. It then creates a lookup table (map) to normalize the CDF values to the range [0, 255]. Finally, it replaces every pixel in the original image with its mapped value.

```c
void HistogramEqualization(IMAGE_HandleTypeDef *pImg, uint32_t *pHist)
{
    uint32_t totalPixels = pImg->width * pImg->height;

    uint32_t cdf[256];

    cdf[0] = pHist[0];
    for(int i = 1; i < 256; i++)
    {
        cdf[i] = cdf[i-1] + pHist[i];
    }

    uint8_t map[256];
    for(int i = 0; i < 256; i++)
    {

        map[i] = (uint8_t)( (cdf[i] * 255) / totalPixels );
    }

    for(uint32_t i = 0; i < totalPixels; i++)
    {
        uint8_t oldPixel = pImg->pData[i];
        pImg->pData[i] = map[oldPixel];
    }
}
```

c) Equalized Histogram Results After applying the equalization function, the histogram was recalculated. As seen in the screenshot below, the pixel counts have shifted. Note that some indices now have a count of 0 (gaps), while others have increased counts. This confirms that the contrast stretching algorithm successfully redistributed the intensity values.



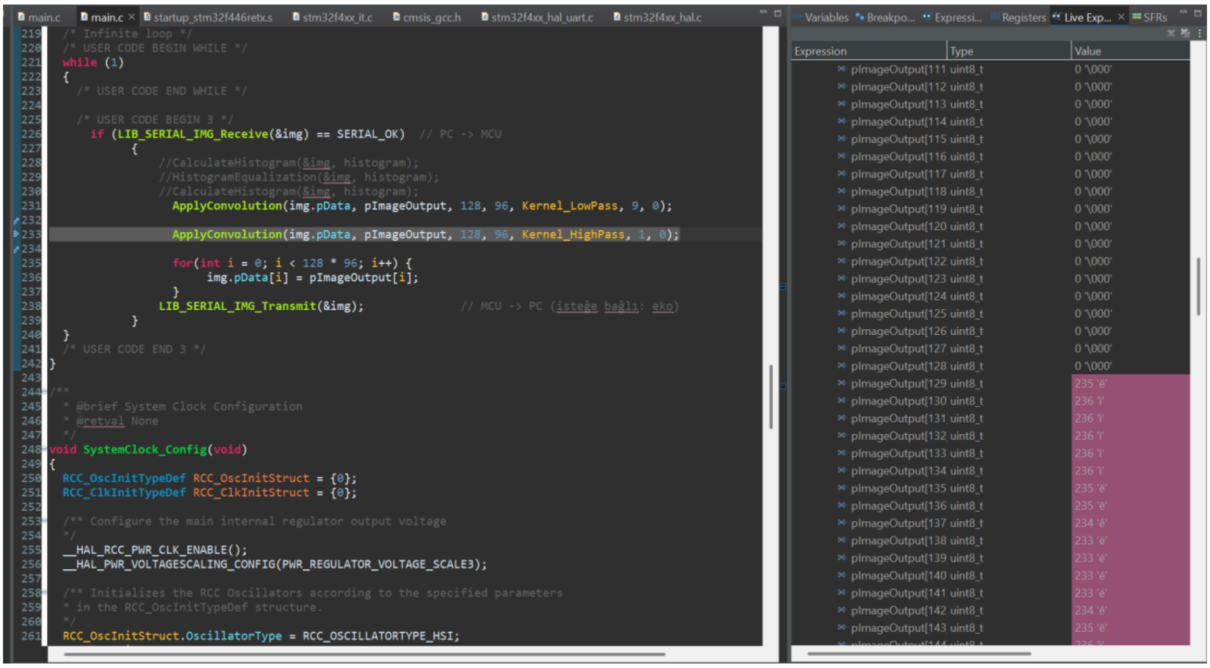
Q3) 2D Convolution and Filtering

a) C Function for 2D Convolution A generic convolution function was created that accepts a kernel, a scaling factor, and an offset. The function uses a secondary buffer (pImageOutput) to store results to avoid modifying the source data during calculation. The borders (Row 0/127 and Col 0/95) are skipped.

```c
void ApplyConvolution(uint8_t* pIn, uint8_t* pOut, int width, int height, const int8_t* kernel, int scale, int offset)
{

    {
        for (int x = 1; x < width - 1; x++)
        {
            int sum = 0;
            // Kernel Index: 0 1 2
            //               3 4 5
            //               6 7 8

            int k = 0;
            for (int ky = -1; ky <= 1; ky++)
            {
                for (int kx = -1; kx <= 1; kx++)
                {
                    int pIdx = (y + ky) * width + (x + kx);

                    sum += pIn[pIdx] * kernel[k];
                    k++;
                }
            }

            if (scale != 0) sum /= scale;

            sum += offset;

            if (sum < 0) sum = 0;
            if (sum > 255) sum = 255;

            pOut[y * width + x] = (uint8_t)sum;
        }
    }
}
```

b) Low Pass Filtering Results A standard Box Blur (Average) kernel was applied.

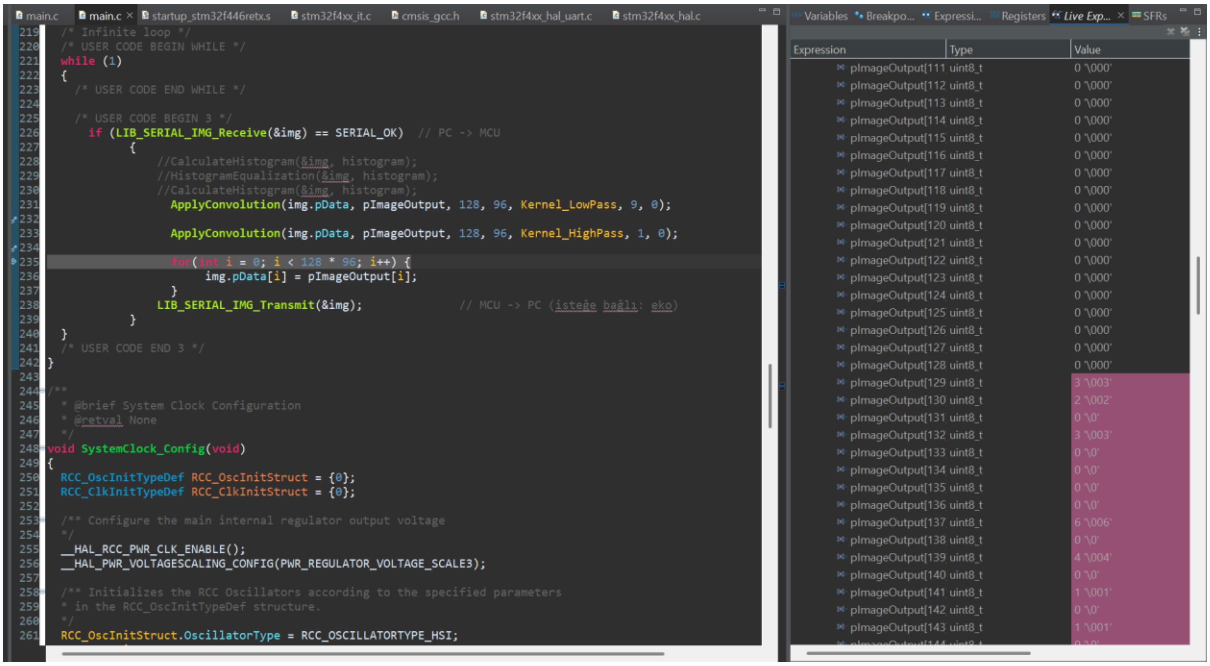
$$Kernel = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Value of 0 until 129th index shows that we applied convolution correctly since the first row doesn't included in convolution.

c) High Pass Filtering Results A Laplacian Edge Detection kernel was applied.

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



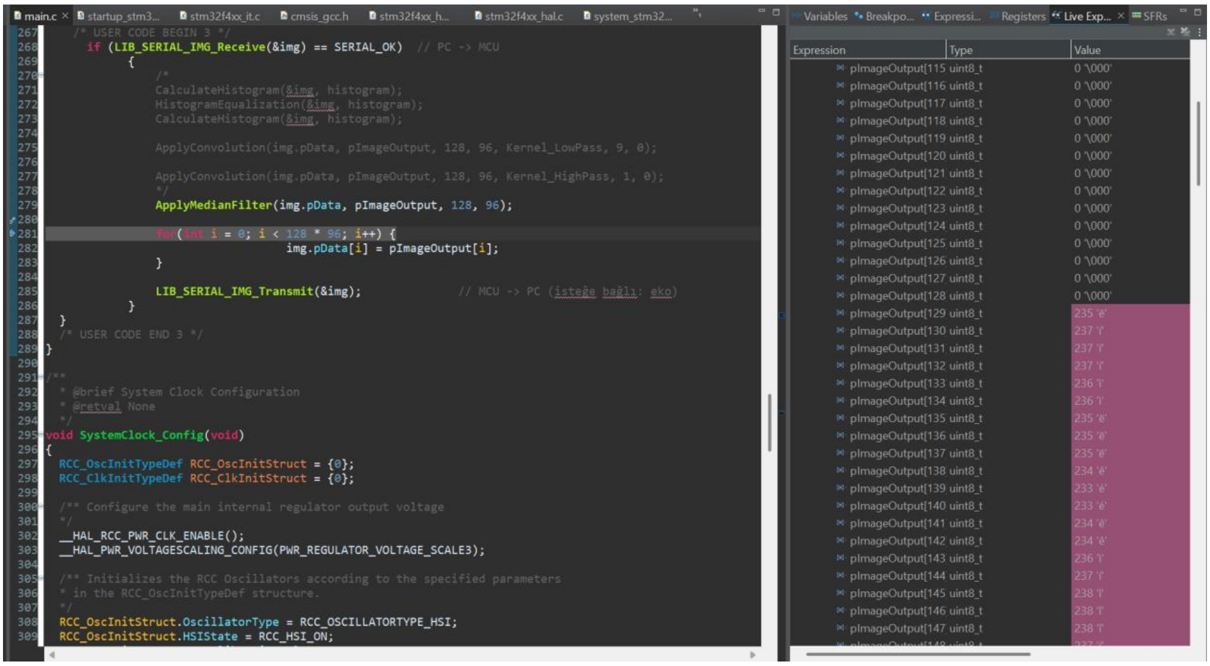
Value of 0 until 129th index shows that we applied convolution correctly since the first row doesn’t included in convolution.

Q4) Median Filtering

a) C Function for Median Filtering The Median Filter collects the 9 neighbors of a pixel into a temporary array, sorts them using a Bubble Sort algorithm, and selects the middle value (index 4). This method is non-linear and is effective for removing salt-and-pepper noise.

```c
void ApplyMedianFilter(uint8_t* pIn, uint8_t* pOut, int width, int height)
{
    uint8_t window[9];

    for(int y = 1; y < height - 1; y++)
    {
        for(int x = 1; x < width - 1; x++)
        {

            int k = 0;
            for(int ky = -1; ky <= 1; ky++)
            {
                for(int kx = -1; kx <= 1; kx++)
                {
                    window[k] = pIn[ (y + ky) * width + (x + kx) ];
                    k++;
                }
            }

            for(int i = 0; i < 9; i++)
            {
                for(int j = i + 1; j < 9; j++)
                {
                    if(window[i] > window[j])
                    {
                        uint8_t temp = window[i];
                        window[i] = window[j];
                        window[j] = temp;
                    }
                }
            }

            pOut[y * width + x] = window[4];
        }
    }
}
```

b) Median Filter Results The filter was applied to the image. The output buffer was inspected.



Conclusion

This project successfully established a bidirectional image processing pipeline between a PC and the STM32 Nucleo-F446RE, validating the microcontroller's capability to perform both statistical and spatial image manipulations. Through the implementation of histogram equalization, the system demonstrated effective contrast enhancement, which was mathematically verified by the redistribution of intensity values observed in the debugger. Furthermore, the successful execution of 2D convolution (Low/High Pass) and Median filtering using a secondary output buffer highlighted the importance of correct memory management in embedded C programming, ultimately confirming the system's reliability in handling fundamental digital image processing algorithms in a resource-constrained environment.