VIETNAM NATIONAL UNIVERSITY HO CHI MINH
UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING
DEPARTMENT OF ELECTRONICS

Capstone project 2

Texture classification using MRELBP and Incremental SVM

Supervisor:	PhD. Tran Hoang Linh
	Mr. Nguyen Tuan Hung
Students:	Tran Quang Nguyen Anh
Student ID:	2151047

HO CHI MINH CITY, DECEMBER 2024

ACKNOWLEDGMENTS

I express my sincere gratitude to everyone who has supported and guided me throughout this capstone project on the design of MRELBP algorithm for image extraction.

First of all, I am deeply indebted to Ph.D. Tran Hoang Linh for their exceptional supervision, insightful feedback, and unwavering encouragement. Their expertise in computer architecture and digital design principles has been instrumental in shaping this project and helping me overcome various challenges.

I also extend my heartfelt appreciation to Mr.Hung for his invaluable assistance, constructive discussions, and patient guidance. His contributions and support have been invaluable throughout this journey.

Ho Chi Minh City, December 2024.

ABSTRACT

This project presents the design and implementation of a Median Robust Extended Local Binary Pattern (MRELBP) module using SystemVerilog for advanced image extraction and classification. The MRELBP algorithm enhances traditional Local Binary Patterns (LBP) by incorporating median and extended feature extraction techniques, significantly improving robustness and accuracy in texture analysis. Our design leverages SystemVerilog's advanced capabilities to ensure efficient hardware implementation, optimized for FPGA deployment. The primary objective is to achieve high-speed processing with minimal resource utilization, making it suitable for real-time image processing applications. Comprehensive simulation and validation are conducted to verify the module's functionality and performance against established benchmarks, demonstrating its superior effectiveness and reliability in diverse texture classification tasks.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

The visual system is essential for intelligent robots to gather external information. As machine learning evolves, target classification systems based on machine learning are increasingly integrated into the visual systems of intelligent robots. These systems classify objects in images collected by the acquisition system to guide the robot's actions. Texture, a fundamental characteristic of most natural surfaces, plays a crucial role in this process due to its ubiquity and variability among different objects.

A typical texture classification system comprises three functional modules: texture feature extraction, training, and classification. The Local Binary Pattern (LBP) algorithm is widely used for texture feature extraction because of its robustness to illumination changes and low computational complexity. Various LBP variants have been developed to enhance its feature extraction capabilities and robustness, including the Median Robust Extended Local Binary Pattern (MRELBP). MRELBP combines a median filter with multi resolution support, addressing the limitations of LBP in handling image blur, noise, and high feature dimensionality.

For training and classification, traditional batch algorithms require retraining with the entire dataset when new samples are introduced. Incremental training algorithms, such as incremental random forests, semi-supervised incremental support vector machines (SVMs), incremental fuzzy classifiers, and incremental SVMs, allow for training with new samples while maintaining the performance of batch algorithms.

Method	**Rate accuracy**	**Advantage**	**Disadvantage**	**Reference**
Local Binary Patterns (LBP)	85	Simple, effective for simple patterns	Sensitive to noise and distortion, less effective for complex patterns	A Study on Applying Eigenfaces and LBP Local Characteristics to the Human Facial Recognition Problem
Gray Level Co-occurrence Matrix (GLCM)	88	Captures statistical texture information well	Computationally complex, requires more resources	Overview of Face Recognition Based on Edge Map
Gabor Filters	90	Analyzes texture at multiple scales and orientation	Analyzes texture at multiple scales and orientations	A Comparative Study of Color Spaces for Image Recognition
A Comparative Study of Color Spaces for Image Recognition	95	Automatically learns features, high accuracy	Requires large datasets and computational resources	Face Recognition Using Centralized LBP

Table 1: Comparing table.

From the above comparing table, I choose the LBP algorithms for the feature extraction step. Due to my aim for this project which is that create a real-time classification system, I need a low of complexity of computation and high speed of calculation. This is the reason why I choose a variant of LBP for feature extraction step and SVM for classification step (low size of dataset but still can have an accuracy classification rate, high speed of predication). But at this capstone project 2, I just focus on designing the feature extraction algorithm.

1.2 Project mission

The mission of this project is to design and implement a robust and efficient texture classification system using the Median Robust Extended Local Binary Pattern (MRELBP) algorithm. This project aims to address the limitations of traditional Local Binary Pattern (LBP) algorithms by improving robustness

to image blur and noise, as well as enhancing texture representation accuracy through the use of median filtering and multi resolution support.Below is my task:

+ Research and stuying about the MRELBP algorimth.
+ Design the block diagram.
+ Design following the block diagram and compare with the reference.

2 Theoretical background

2.1 Logic Binary Pattern(LBP)

Local Binary Pattern (LBP) is one of the most widely used methods for feature extraction in pattern recognition and texture analysis. LBP is powerful due to its invariance to monotonic illumination changes and low computational complexity.

Local Binary Pattern operates by comparing the value of a central pixel to its neighboring pixels within a small region. The basic steps of the LBP method are as follows:

+ Select a Central Pixel: Choose a central pixel in a small region (ex: 3x3) of an image.
+ Compare with Neighbors: Compare the value of the central pixel to the values of the neighboring pixels.
+ Assign Binary Labels: Assign a binary label (0 or 1) to each neighbor. If the value of the neighboring pixel is greater than or equal to the central pixel, assign 1; otherwise, assign 0.
+ Form Binary Code: Combine these binary labels to form a binary code.
+ Convert to Decimal Value: Convert the binary code to a decimal value to represent the texture feature at the central pixel.

Mathematical fomular:

Assume the central pixel has a value I_c and the neighboring pixels have values I_n (n = 0, 1, ..., N-1), where N is the number of neighboring pixels. The LBP code at the central pixel is computed as follows:

$$\mathrm{LBP}(x, y) = \sum_{n=0}^{N-1} s(I_n - I_c) \cdot 2^n \tag{1}$$

where the function $s(x)$ is a step function:

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2}$$

From above equations, we can see that LBP is highly sensitive to noise in the image. Small variations in pixel values due to noise can significantly affect the LBP code, leading to less reliable texture representation. Furthermore, the traditional LBP uses a fixed 3x3 neighborhood to calculate the binary pattern. This may not be suitable for all types of textures, particularly those with varying scales and resolutions. When extended to multi-resolution analysis, the LBP features can result in high-dimensional feature vectors, which may require significant computational resources for storage and processing. So I come to a variant of LBP to resolve these problems which has been proven.

2.2 Median Robust Extended Logic Binary Pattern(MRELBP)

MRELBP is an advanced texture feature extraction algorithm that builds upon the traditional Local Binary Pattern (LBP) method by incorporating median filtering and multi-resolution support. The key enhancements of MRELBP are designed to improve the robustness and accuracy of texture analysis, especially in the presence of noise and image blur.

The MRELBP was enhanced by Liu from ELBP (Extended Logic Binary Pattern) and RELBP (Robust Extended Logic Binary Pattern) to resolve the problem. Which will be discuss following the below picture.

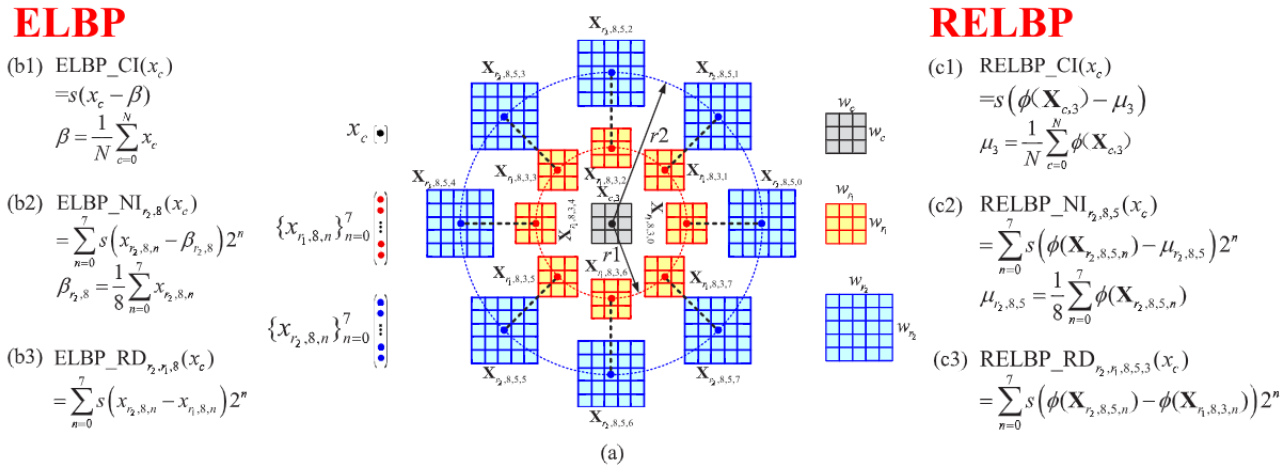

Figure 1: ELBP and RELBP explain computing.

For both enhance methods, they will calculate three variables:

+ CI (Center Intensity): This represents the intensity value of the central pixel in the local neighborhood. The central pixel's intensity is used as a reference point to compare with its neighboring pixels.
+ NI (Neighbor Intensity): These are the intensity values of the pixels surrounding the central pixel within the local neighborhood. These neighboring pixels are compared with the central pixel to determine the local binary pattern.
+ RD (Radius Difference): This denotes the difference in intensity values between the central pixel and its neighboring pixels, calculated across various radius. This multi-resolution approach helps in capturing texture details at different scales, enhancing the robustness and accuracy of the feature extraction process.

Extended Logic Binary Pattern (ELBP):

Extended Local Binary Pattern (ELBP) is a variation of the traditional Local Binary Pattern (LBP) designed to improve texture analysis by incorporating additional features and extending the analysis to multiple resolutions.ELBP enhances the traditional LBP by analyzing the image at different resolutions. This multi-resolution approach allows ELBP to capture texture details at various scales, providing a more comprehensive texture representation (CI, NI). Furthermore, ELBP extends the basic LBP by including additional information from the pixel neighborhood. This can involve using more extensive neighborhoods or considering different statistical measures, which helps in capturing more complex texture patterns.

CI fomular:

$$\text{ELBP\_CI}(x_c) = s(x_c - \beta) \tag{3}$$

Where β is the average value of total calculated center pixel. x_c is the center pixel.

$$\beta = \frac{1}{N}\sum_{c=0}^{N} x_c \tag{4}$$

NI compares each neighboring pixel intensity $x_{r_2,a,n}$ with the average intensity.

$$\text{ELBP\_NI}_{r_2,8}(x_c) = \sum_{n=0}^{7} s\left(x_{r_2,8,n} - \beta_{r_2,8}\right) 2^n \tag{5}$$

Where $x_{r_2,8,n}$ is neighbor pixel surrounding center pixel x_c. β is the average value of 8 neighbor pixels.

$$\beta_{r_2,8} = \frac{1}{8}\sum_{n=0}^{7} x_{r_2,8,n} \tag{6}$$

RD compares the intensities of neighboring pixels at two different radius

$$\text{ELBP\_RD}_{r_2 \cdot r_1,8}(x_c) = \sum_{n=0}^{7} S\left(x_{r_{2,8,n}} - x_{r_{1,8,n}}\right) 2^n \tag{7}$$

Following these above equations, ELBP captures texture features at multiple resolutions, providing a comprehensive and robust representation of textures in images through pixels. But, this still is affected strongly by noise due to using individual pixel of image for computing.

Robust Extended Logic Binary Pattern (RELBP):

RELBP enhances ELBP by incorporating filtering and multi-resolution analysis, which makes it more robust against noise and image blur. The filtering step effectively reduces noise while preserving important texture details. Additionally, the multi-resolution approach captures both fine and coarse texture features, providing a more comprehensive and detailed texture representation. Overall, RELBP offers improved performance in texture analysis, especially in challenging real-world applications, compared to ELBP, which does not include these robust enhancements. From equation (3) to (7), add the filtering step before calculating.

CI fomular:

$$\text{RELBP\_CI}(x_c) = s\left(\phi(\mathbf{X}_{c,3}) - \mu_3\right) \tag{8}$$

Where ϕ is the filter applied to image before calculating. μ_3 is the average value of center pixels after filetring.

$$\mu_3 = \frac{1}{N} \sum_{c=0}^{N} \phi(\mathbf{X}_{c,3}) \tag{9}$$

NI fomular:

$$\text{RELBP\_NI}_{r_2,8,5}(x_c) = \sum_{n=0}^{7} s\left(\phi(\mathbf{X}_{r_2,8,5,n}) - \mu_{r_2,8,5}\right) 2^n \tag{10}$$

$$equation \mu_{r_2,8,5} = \frac{1}{8} \sum_{n=0}^{7} \phi(\mathbf{X}_{r_2,8,5,n}) \tag{11}$$

RD fomular:

$$\text{RELBP\_RD}_{t_{2,r_1,8,5,3}}(x_c) = \sum_{n=0}^{7} s\left(\phi(\mathbf{X}_{t_{2,8,5,n}}) - \phi(\mathbf{X}_{t_{1,8,3,n}})\right) 2^n \tag{12}$$

From above, the filter that is applied will be the extended name for RELBP. With MRELBP, this method applies the median filter to the filtering step. The reason why i choose the median filter is that following the actual experiments or other paper, the noise terminating ability of median filter is much more higher than the others. (90-95% especially with salt-and-peper noise)

Filter Type	Noise Reduction	Edge Preservation	Common Applications
Average Filter	Moderate	Poor	Image Blurring, Reducing Gaussian Noise
Median Filter	High	Good	Salt and Pepper Noise Removal, Image Denoising
Max Filter	Low	Poor	Enhancing Bright Spots, Image Highlight Detection

Table 2: Comparison of Noise Reduction Efficiency between Filters

2.3 Bilinear interpolation

Bilinear interpolation is utilized in Median Robust Extended Local Binary Pattern (MRELBP) to enhance the precision and accuracy of feature extraction in texture analysis. In MRELBP, the texture details are captured at different resolutions, and often the values of the pixels at these different scales do not align perfectly with the grid of the original image. Bilinear interpolation helps in estimating pixel values at non-integer coordinates, ensuring that the texture features are accurately represented across various scales. This method calculates the value of a pixel based on a weighted average of the four nearest pixel values, providing a smooth transition between pixel intensities. By incorporating bilinear interpolation, MRELBP achieves a more refined and detailed texture representation, which is crucial for effective noise reduction and blur resistance in real-world applications.

Furthermore, the coordinates of 8 neighbor points surrounding a center pixel are following the formula below:

$$x_{\text{neighbor}}(k) = x_{\text{center}} + r\cos\left(\frac{2k\pi}{8}\right) \tag{13}$$

$$y_{\text{neighbor}}(k) = y_{\text{center}} + r\sin\left(\frac{2k\pi}{8}\right) \tag{14}$$

Where r is the radius and k is the neighbor pixels which belongs to [0:7].

From above equations, with k is odd number, the calculated coordinates is non-integer numbers or be center of pixel. With that points, i need to calculate the values at these points by bilinear interpolation.

Bilinear interpolation is a resampling method used in image processing and computer graphics to determine the value of a pixel based on its neighboring pixels. It provides a smooth transition between pixel values, especially when resizing or transforming images.

Mathematical fomular:

Given four pixel values Q_{11}, Q_{12}, Q_{21}, Q_{22} at coordinates (x_1,y_1), (x_2,y_1), (x_1,y_2), (x_2,y_2) the interpolated value P at (x,y) is computed as:

$$P = \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)}Q_{11} + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)}Q_{21} + \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)}Q_{12} + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)}Q_{22} \tag{15}$$

3 Design strategy

3.1 Block diagram

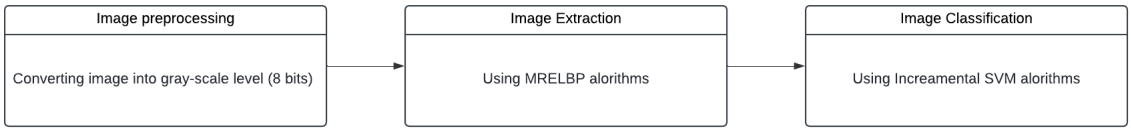

Figure 2: System block diagram.

For classification system, it combines 3 steps:

+ Image processing: Normalize the image by python to gray-scale 8-bit image with size 128x128.
+ Image extraction: Extract features of image after normalization to a histogram. Histogram will be the frequency of 8-bit length values which are the symbolize of a image area (with size 9x9).
+ Image classification: Using data from histogram to classify. At this, I will use a available dataset: Face detector or object detector to a reference data. When the histogram is transferred to classification step, this will compare to the available dataset.

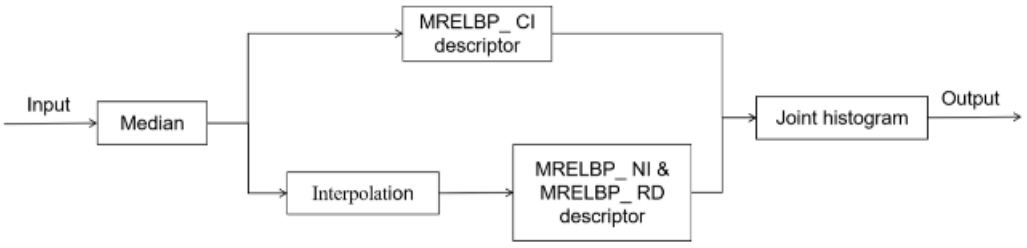

Figure 3: MRELBP module data flow.

For MRELBP module, it combines 3 steps:

+ Median: Filtering the image
+ CI & NI & RD: Calculate the feature of image.
+ Joint histogram: Storing the data.

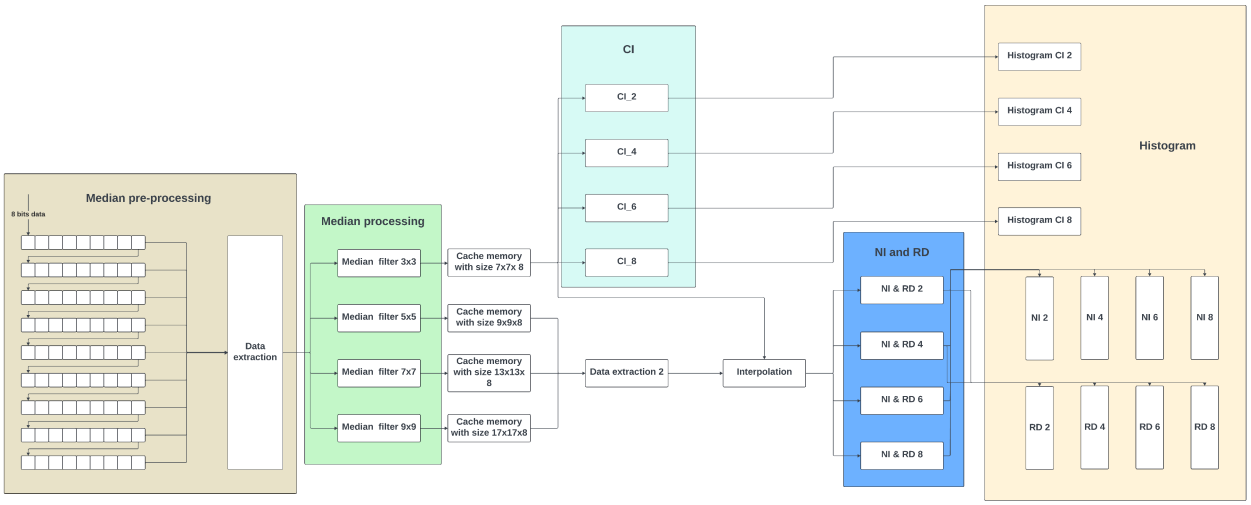

Figure 4: MRELBP module hardware architecture.

Liu conducted a comparison of the effects of different filter types and their parameters on texture extraction performance, examining filters like the Gaussian filter, average filter, and median filter within the RELBP algorithm. After extensive testing and validation of numerous parameters, it was determined that median filtering yielded the best performance parameters.

Parameter	Value
Filter type	MRELBP
Radius	(2, 4, 6, 8)
Number of samples	8
Median size	3x3, 5x5, 7x7, 9x9

Table 3: Parameters.

The choice of filter sizes for different components in MRELBP (Median Robust Extended Local Binary Pattern) is based on the specific requirements for capturing texture features:

+ **CI (Center Intensity):** The CI component focuses on the intensity of the central pixel and its immediate neighbors. A 3x3 filter is sufficient for this purpose because it captures local intensity variations within a small neighborhood, providing a good balance between noise reduction and edge preservation.

+ **NI (Neighborhood Intensity) and RD (Radial Difference):** These components require a broader analysis of the local texture. The larger median filters (5x5, 7x7, 9x9) are used to capture more extensive neighborhood information and radial differences. These larger kernels help in encompassing a wider area around each pixel, which is essential for accurately representing complex texture features and ensuring robust noise and blur resistance

3.2 Median pre-processing

I/O ports:

Explain:

The memory cache stores the sub-image input (9x9 image input) for computing. Using the BRAM architecture for design, nine 72-bit shift registers will be stored for matrix input.

for each clock, the the data 9x9 matrix will be push out for median processing. In calculation, the total clock of filling fully BRAM is 81 clocks witch each clock will fill a pixel of matrix 9x9.

Signal name	Width	Direction	Description
i_clk	1	Input	Global clock
i_rst	1	Input	Global reset
i_data	8	Input	Pixel value
i_wren	1	Input	Write enable
o_data	648	Output	Data output

Table 4: Median processing I/O table.

3.3 Data extraction for median processing (Control unit)

3.3.1 Control unit (3x3, 5x5 ,7x7 median filters)

I/O port:

Signal name	Width	Direction	Description
i_clk	1	Input	Global clock
i_rst	1	Input	Global reset
o_state	3	Output	State output for test
o_median_3x3	1	Output	Enable calculate median 3x3
o_median_5x5	1	Output	Enable calculation median 5x5
o_median_7x7	1	Output	Enable calculation median 7x7

Table 5: Control unit I/O table.

State diagram:

Current state	Next state	Condition
IDLE	IDLE	Delay for initial fill in BRAM
IDLE	EN_FIRST_COL	Done delay
EN_FIRST_COL	EN_COL	Always jump
EN_COL	EN_COL	Delay following the computing times
EN_COL	EN_NEW_ROW	Done delay
EN_NEW_ROW	EN_NEW_ROW	Delay for terminating the first row
EN_NEW_ROW	EN_FIRST_COL	Done terminating delay
EN_NEW_ROW	WAIT_NEW_MATRIX	Taking out all elements of old matrix
WAIT_NEW_MATRIX	WAIT_NEW_MATRIX	Wait to the new matrix fill in
WAIT_NEW_MATRIX	EN_FIRST_COL	Comback the first step

Table 6: State table.

Explain:

With BRAM has 9 shift registers to store the data of input image, corresponding to each radius end median filter size, the data of BRAM will be taken out following below flow:

+ Median filter 3x3: Taking data with size 24 bits of shift registers: 4,5,6.
+ Median filter 5x5: Taking data with size 40 bits of shift registers: 3,4,5,6,7.
+ Median filter 7x7: Taking data with size 56 bits of shift registers: 2,3,4,5,6,7,8.
+ Median filter 9x9: Taking data with size 81 bits of all shift registers.

Furthermore, with different computing times with the 9x9 input matrix of each median filter, I need a some control unit for enable computing task of each median filter.

Median filter 3x3:

+ Fill the data to register 6 for the first time computing (IDLE state): At this state, I will delay 9 * 6 = 54 clocks to wait the first data to be written to the first element of register number 6.
+ Take the first time for computing (EN_FIRST_COL state): At this state, I divide the total time taken (7 times for each row: 7x7 matrix output) into 2 parts: first taken and the other taken times.

+ Take the other time (EN_COL state): take the data from shift registers 6 times continuously. For total clock of enable computing task is 7 clocks.
+ Check terminating the data (EN_NEW_ROW) of shift register 6 in 7 times. If the terminating is 7 times, come to state that wait to fill in new matrix. In contrary, come to state that terminates the oldest row of data. (in 3 clocks).
+ Check wait to fill in the new matrix (WAIT_NEW_MATRIX), this will be delay 21 clocks in total.

With the same priority to 5x5 and 7x7 median filters and they just are different from the clock delay.

Median filter 5x5:

+ IDLE to EN_FIRST_COL: 63 clocks.
+ EN_FIRST_COL to EN_COL: 1 clock.
+ EN_COL to EN_NEW_ROW: 4 clocks.
+ EN_NEW_ROW to EN_FIRST_COL: 5 clocks.
+ WAIT_NEW_MATRIX to EN_FIRST_COL: 41 clocks.

Median filter 7x7:

+ IDLE to EN_FIRST_COL: 72 clocks.
+ EN_FIRST_COL to EN_COL: 1 clock.
+ EN_COL to EN_NEW_ROW: 2 clocks.
+ EN_NEW_ROW to EN_FIRST_COL: 7 clocks.
+ WAIT_NEW_MATRIX to EN_FIRST_COL: 61 clocks.

3.3.2 Control unit (9x9 median filter)

I/O table:

Signal name	Width	Direction	Description
i_clk	1	Input	Global clock
i_rst	1	Input	Global reset
o_state	1	Output	State output for test
o_median_9x9	1	Output	Enable calculate median 9x9

Table 7: Control unit of median 9x9 I/O table.

Explain:

With the median filter 9x9, I just need to delay 81 clocks at each computing enable median filter.

3.4 Median processing

The median filtering sampling ranges corresponding to radius combinations (2, 4, 6,8) are 3×3, 5×5, 7×7 and 9×9.

3.4.1 Median filter 3x3

I/O port:

Signal name	Width	Direction	Description
i_pixel	72	Input	Input matrix 3x3
o_pixel _median	1	Output	Median value

Table 8: Median 3x3 I/O table.

Explain:

For matrix input with size 3x3, the steps to finding median value:

+ Sorting each row in ascending.
+ Sorting the middle column of input matrix after sorting row.
+ Median value is the middle value of sorted middle column. (Output a 7x7 matrix)

3.4.2 Median filter 5x5

I/O port:

Signal name	Width	Direction	Description
i_pixel	200	Input	Input matrix 5x5
o_pixel _median	1	Output	Median value

Table 9: Median 5x5 I/O table.

Explain:

Take the reference to the fast algorithm from Yang's proposed, I've implemented the median filter 5x5 by workflow:

+ Sorting each row in ascending
+ Sort each column in ascending.
+ Sort three 45 degree diagonal strings.
+ Sort 3 diagonal string.
+ Get the median value. (output a 5x5 matrix per 9x9 matrix input)

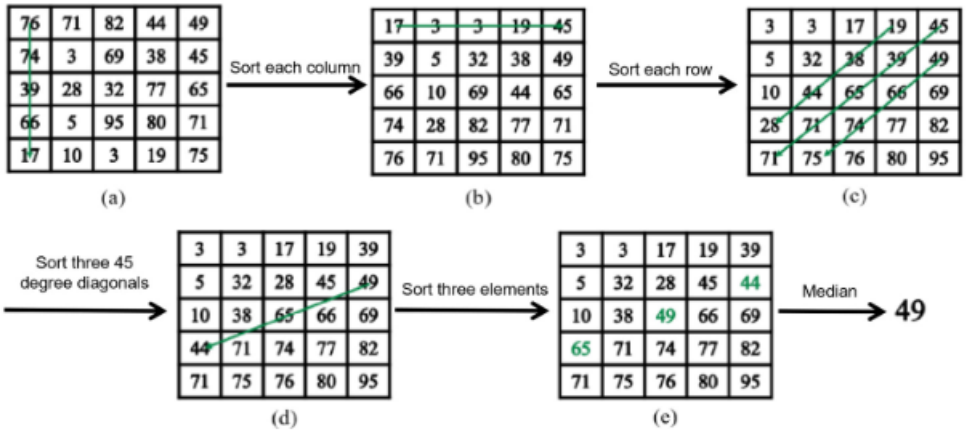

Figure 5: Median filter 5x5 work flow.

3.4.3 Median filter 7x7

I/O port:

Signal name	Width	Direction	Description
i_clk	1	Input	Global clock
i_pixel	392	Input	Input matrix 7x7
o_pixel_median	1	Output	Median value

Table 10: Median 7x7 I/O table.

Explain:
Take the reference to the fast algorithm from Liu's proposed, I've implemented the median filter 7x7 by workflow:

+ Sorting each row in ascending
+ Sort each column in ascending.

+ Sort 3 elements in upper and lower corner.
+ Delete 24 elements.
+ Decrease dimension to 5x5.
+ Sort like previous part: median filter 5x5.
+ Get the median value.

For a matrix input 9x9 goes through median filter 7x7, I get 3x3 matrix result. And to enhance the speed of computing. I use the pipeline manner, add register between step delete element and reduce dimension.

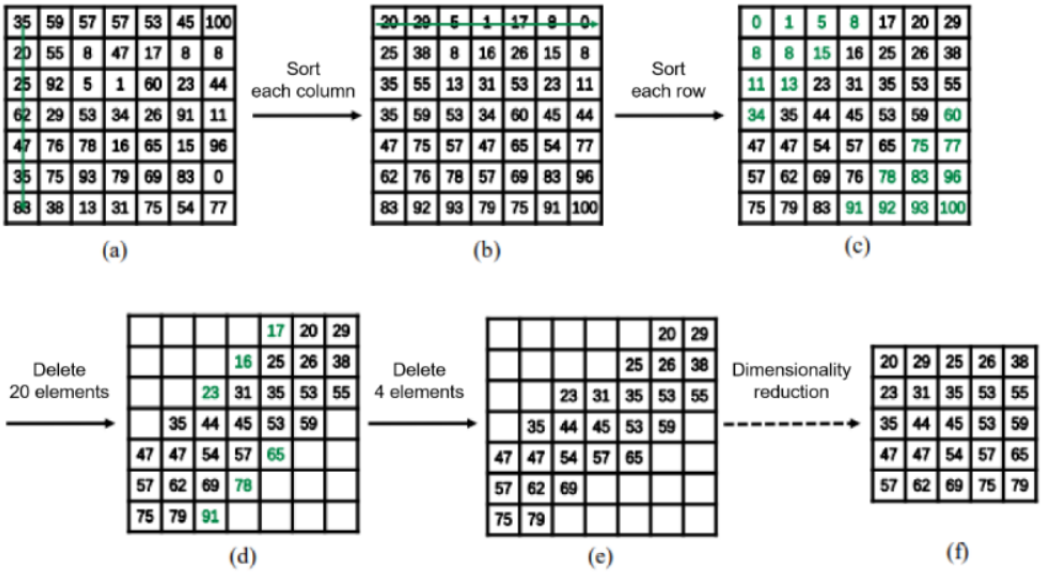

Figure 6: Median filter 7x7 work flow.

3.4.4 Median filter 9x9

I/O port:

Signal name	Width	Direction	Description
i_clk	1	Input	Global clock
i_pixel	648	Input	Input matrix 7x7
o_pixel_median	1	Output	Median value

Table 11: Median 9x9 I/O table.

Explain:

The same as work flow of median filter 7x7, but adding one more step reduction from 9x9 to 7x7.

+ Sorting each row in ascending
+ Sort each column in ascending.
+ Sort 3 elements in upper and lower corner.
+ Delete 22 elements.
+ Reduce dimension to 7x7.
+ Sort like previous part: median filter 7x7.
+ Get the median value.

For a matrix input 9x9 goes through median filter 9x9, I get a median value. And to enhance the speed of computing. I use the pipeline manner like median filter 7x7, add register before steps that reduction dimension.

3.4.5 New sort method for 9 elements

To enhance the speed of computing and reduce the resource of design, I use a new of sort method for step sorting.

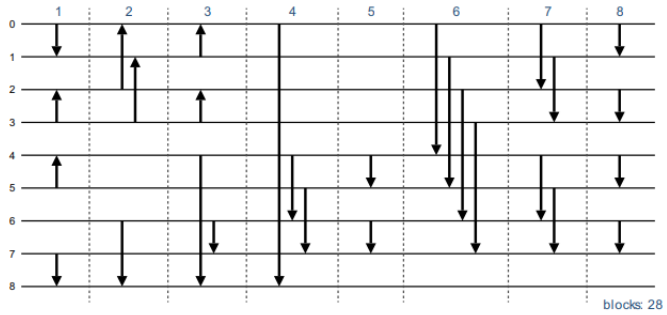

Figure 7: Sorting work flow.

Following the Figure 7, the steps of sorting 9 elements is below:

+ Initial setup: The elements to be sorted are placed on the horizontal lines at the top. The numbers at the top (0 to 8) represent the initial positions of these elements.

+ Comparison and swap: Vertical arrows indicate the comparisons and possible swaps between elements. Each arrow represents a comparison between two elements. If the elements are out of order, they are swapped to ensure the greater element moves to the top position.

+ Sorting steps:The numbers on the left (0 to 8) indicate the steps in the sorting process. At each step, specific pairs of elements are compared and swapped if necessary. The arrows guide the sequence of operations needed to sort the elements. Each step ensures that certain elements are in the correct relative order.

And following the below comparing table, I see that the new method takes a fewer used resources and higher run frequency with others sort methods.

Sorting Algorithm	**Frequency (MHz)**	**Logic Elements**
New Sorting Method	80	672
Bubble Sort	55	700
Bitonic Sort	50	750
Quick Sort	45	800
Merge Sort	40	850
Sorting Network	35	725

Table 12: Comparison of Sorting Algorithms

3.5 Center intensity descriptor

I/O port:

Signal name	**Width**	**Direction**	**Description**
i_pixel	392	Input	Input result matrix of median filter 3x3
o_count_1	6	Output	Count logic 1
o_count_0	6	Output	Count logic 0

Table 13: CI I/O table.

Explain:

For each radius, they need a (2*R + 1)x(2*R+1) input data for calculating the CI. (R=2: 5x5; R=4: 9x9; R=6: 13x13; R=9; 17x17). Due to using result matrix of median filter 3x3, so the input for CI calculation is just 7x7.

First, the reasons why CI just takes the result of median filter 3x3 is that first of all, the 3x3 median filter focuses on a small neighborhood centered on the pixel of interest. This size is sufficient to capture local intensity variations and reduce noise without affecting the immediate texture details around the central pixel. A 3x3 filter is computationally efficient compared to larger filters. It requires less processing power and memory, making it faster and more suitable for real-time applications.Using a 3x3 filter ensures that the CI component remains robust against small-scale noise without introducing significant blurring. This balance is crucial for maintaining the integrity of texture features in the image.

Secondly, to enhance the speed of calculating, the size of needed input data will be reduced or zoom the image. This is proven that decreases the complexity of CI calculation but still hold the features. This will be synchronous with calculating the NI and RD at R=2 with out the delays, this means using all data of matrix output of median filter 3x3. The steps of calculating CI at R = {2,4,6,8}:

+ Calculate the average Value of matrix size (2*R+1)x(2*R+1) with R={4,6,8}, the computing input matrix is always 7x7 (zoom to the center). The result is stored in fixed point form: Q14.2. Calculate by the equaltion: average = sum / $(2 * R + 1)^2$.
+ Using that value and apply to equation (8) to transfer the value to binary.
+ Count the number of logic 1 and logic 0.
+ Transfer to histogram.

3.6 Bilinear interpolation

I/O portt:

Signal name	Width	Direction	Description
i_pixel	168	Input	Input the pixel after calculate the coordinate
o_neighbor_odd	96	Output	Value of coordinate at k is odd (13) & (14)
o_neighbor_even	32	Output	Value of coordinate at k is even (13) & (14)
o_pixel_center	8	Output	Value of center pixel(just for R=2)

Table 14: Interpolation I/O table.

Explain:

Using equation (13) & (14) with the coordinate of the center pixel of each input matrix (2*R+1)x(2*R+1): R=2:(2,2); R=4:(4,4); R=6:(6,6); R=8:(8,8), I can calculate the coordinates of the neighbors pixels.

With the coordinates at k are odd number, the value will be stored in fixed point form; Q16.8. And take the integral of value to be the coordinate for bilinear interpolation.

With bilinear interpolation, 4 pixels per k(odd number) will be taken to calculate: (x_{ne}, y_{ne}), (x_{ne} + 1, y_{ne}), (x_{ne}, y_{ne} + 1) and (x_{ne} + 1, y_{ne} + 1). Then output the result for RD and Ni to calculate next task in a series of output:

+ R=2: 4 points at k is odd number after interpolation, 4 points at k is even number and a center pixel for RD.
+ R=4,6,8: 4 points at k is odd number after interpolation, 4 points at k is even number.

3.7 Neighbor intensity descriptor

I/O port:

Explain:

First the i_neighbor_even will be converted to a fixed point to synthesize the form of i_neighbor_odd. Then calculate the average of 8 points and convert the 8 points to binary. At the result, I get a 8-bit number. This will be transferred to histogram.

Signal name	Width	Direction	Description
i_neighbor_odd	96	Input	Value of coordinate at k is odd (13) & (14)
i_neighbor_even	32	Input	Value of coordinate at k is even (13) & (14)
o_ni_result 7	8	Output	Result of (10)

Table 15: NI I/O table.

3.8 Radius different descriptor

I/O port:

Signal name	Width	Direction	Description
i_neighbor_odd	192	Input	Value of coordinate at k is odd (13) & (14)
i_neighbor_even	64	Input	Value of coordinate at k is even (13) & (14)
i_pixel_center	8	Input	Central pixel of matrix output of median filter 3x3
o_ni_result 7	8	Output	Result of (10)

Table 16: NI I/O table.

Explain:

For the RD calculation, I compare 8 points of two different radius. Means that R=2 with central pixel, R=4 with R=2, R=6 with R=4, R=8 with R=6. Using equation (12) for computation, the output result of this calculation is an 8-bit number and then transferred to histogram.

3.9 Histogram

For histogram at each radius: 2,4,6,8, I divide the histogram into 3 part. CI: two address (0 and 1), NI and RD: 255 address per parts. To tranfer data to histogram I follow by the below workflow:

+ CI: For each radius: the count logic 1 and count logic 0 values will be add to the value of address in the CI histogram.
+ NI: Both 4 values of each NI values corresponding to different radius will be the address and the value of that will increase by 1 each time that address is detect.
+ RD: Get the same workflow to NI.

4 Result and verification

4.1 Verification

For verification plan, I will work in flow that: design each blocks, verify their functions, connect them together, verify all. For the design, i just into step 3 due to transfer from using register to BRAM.

Block	Status
BRAM for input 9x9 image	PASSED
Control unit for enable median processing	PASSED
Median processing (3x3,5x5,7x7,9x9)	PASSED
CI calculator	PASSED
Cache from median process to interpolation	PASSED
Control unit for cache from median process to interpolation	RUNNING
Interpolation calculator	PASSED
NI & RD calculator	PASSED
Histogram for CI	PASSED
Histogram for NI and RD	PASSED

Table 17: Verification plan table.

4.2 Result on Quartus

I design and synthesize on Quartus 13.01 and simulate the signals and print the results on the Modelsim. For easy check the result when writing testbench, I usually use the $display and $monitor to print on the transcipt console.

BRAM for input 9x9 image:

```
Test 1: check reset
[time 0] o_data[0]=000000000000, o_data[1]=000000000000, o_data[2]=000000000000
Test 2: check data with i_wren = 1
[time 25] o_data[0]=0a0000000000, o_data[1]=000000000000, o_data[2]=000000000000
Test 3: Check shift data
[time 35] o_data[0]=105000000000, o_data[1]=000000000000, o_data[2]=000000000000
[time 45] o_data[0]=138280000000, o_data[1]=000000000000, o_data[2]=000000000000
[time 55] o_data[0]=139c14000000, o_data[1]=000000000000, o_data[2]=000000000000
[time 65] o_data[0]=139ce0a00000, o_data[1]=000000000000, o_data[2]=000000000000
[time 75] o_data[0]=139ce7050000, o_data[1]=000000000000, o_data[2]=000000000000
[time 85] o_data[0]=139ce7382800, o_data[1]=000000000000, o_data[2]=000000000000
[time 95] o_data[0]=139ce739c140, o_data[1]=000000000000, o_data[2]=000000000000
[time 105] o_data[0]=139ce739ce0a, o_data[1]=000000000000, o_data[2]=000000000000
[time 115] o_data[0]=139ce739ce70, o_data[1]=0a0000000000, o_data[2]=000000000000
[time 125] o_data[0]=139ce739ce73, o_data[1]=105000000000, o_data[2]=000000000000
[time 135] o_data[0]=139ce739ce73, o_data[1]=138280000000, o_data[2]=000000000000
```

Figure 8: Bram test on quartus.

Explain:

Above is the test result of a BRAM for image input:

+ Test 1: Check Reset: To ensure that the BRAM is correctly reset.
+ Test 2: Check data with i_wren = 1: To verify that data can be written to the BRAM.
+ Test 3: Check shift data: To observe how data shifts within the BRAM.

Control unit for median process:

```
[Time 0] rst=1:  state=0 output=0
[Time 20] rst=0:  state=0 output=0
[Time 740] rst=0:  state=1 output=1
[Time 750] rst=0:  state=2 output=1
[Time 770] rst=0:  state=3 output=0
[Time 840] rst=0:  state=1 output=1
[Time 850] rst=0:  state=2 output=1
[Time 870] rst=0:  state=3 output=0
[Time 940] rst=0:  state=1 output=1
[Time 950] rst=0:  state=2 output=1
[Time 970] rst=0:  state=3 output=0
[Time 980] rst=0:  state=4 output=0
[Time 1580] rst=0:  state=1 output=1
[Time 1590] rst=0:  state=2 output=1
```

Figure 9: Control unit for median 7x7 test on quartus.

Explain:

For control unit, with output = 1 means that median filter 7x7 are enable to compute. In this case output equal 1 in 3 time each time is in 3 clocks. This means median filter 7x7 has been calculating 9 times (3x3 output matrix).

Median filter processing

```
Input Matrix:
[ 36 129   9  99  13]
[141 101  18   1  13]
[118  61 237 140 249]
[198 197 170 229 119]
[ 18 143 242 206 232]
Median Value: 129
```

Figure 10: Median filter 5x5.

```
Input Matrix:
[ 36 129   9  99  13 141 101]
[ 18   1  13 118  61 237 140]
[249 198 197 170 229 119  18]
[143 242 206 232 197  92 189]
[ 45 101  99  10 128  32 170]
[157 150  19  13  83 107 213]
[  2 174  29 207  35  10 202]
Median Value: 118
```

Figure 11: Median filter 7x7.

```
Input Matrix:
[ 4  1  9  3 13 13  5 18  1 ]
[13 22 29 13 12 25  6  5 10 ]
[ 5 23 18 15 18 14  8  5 28 ]
[29 13  5  3 10  0  0 10 29 ]
[22 19 13 19 11 21  2 14 29 ]
[15  3 10 10 28 18 10  1 24 ]
[24  9 11 22  6 14 28 10 11 ]
[17  5 15 27 26 30 21 17 25 ]
[ 2 12 31 15 24 23 31 28 27 ]
Median Value: 14
```

Figure 12: Median filter 9x9.

CI with R=2,4,6,8 (Symobolize at R=2):

```
Matrix input:
[ 25  35  45  55  65]
[ 30  40  50  60  70]
[ 35  45  55  65  75]
[ 40  50  60  70  80]
[ 45  55  65  75  85]
expected average:    55 00
PASS.
Matrix output:
[0 0 0 0 1]
[0 0 0 1 1]
[0 0 0 1 1]
[0 0 1 1 1]
[0 0 1 1 1]
```

Figure 13: CI calculation at R=2.

Explain:

The test image of CI, represent the matrix output before counting logic 1 and logic 0 and then transfering to histogram. Furthermore, It represents the integral of average value in decimal and the other part in binary for check.

Interpolation at R=2,4,6,8 (symbolize at R=2, R=8

```
Case k = 0
x = 001000, y = 000800
Case k = 1
x = 000da8, y = 000da8
Case k = 2
x = 000800, y = 001000
Case k = 3
x = 000258, y = 000da8
Case k = 4
x = 000000, y = 000800
Case k = 5
x = 000258, y = 000258
Case k = 6
x = 000800, y = 000000
Case k = 7
x = 000da8, y = 000258
```

Figure 14: Calculate coordinate for interpolation at R=8

```
Initial Matrix:
0a 14 1e 28 32
3c 46 50 5a 64
6e 78 82 8c 96
a0 aa b4 be c8
d2 dc e6 f0 fa
Center Pixel: 82
Q_NE_0: 96, Q_NE_1: 00d6d7, Q_NE_2: e6, Q_NE_3: 00ba8f
Q_NE_4: 6e, Q_NE_5: 002d27, Q_NE_6: 1e, Q_NE_7: 00496f
```

Figure 15: Interpolation at R=2

Explain:

At R=2, the output is not only the 8 neighbor pixel points but also the center pixel of matrix input.

```
Initial Matrix:
 Pixel_08:  20
 Pixel_22:  40, Pixel_23:  50, Pixel_210:  70, Pixel_211:  80
 Pixel_32: 100, Pixel_33: 110, Pixel_310: 130, Pixel_311: 140
 Pixel_80: 160, Pixel_88: 170, Pixel_816: 180
 Pixel_102: 200, Pixel_103: 210, Pixel_1313: 230, Pixel_1314: 240
 Pixel_112: 250, Pixel_113: 255, Pixel_1413: 220, Pixel_1414: 230
 Pixel_168: 240
Q_NE_0: b4, Q_NE_1: 00e5ff, Q_NE_2: f0, Q_NE_3: 00eb1e
Q_NE_4: a0, Q_NE_5: 00400f, Q_NE_6: 14, Q_NE_7: 00612f
```

Figure 16: Interpolation at R=8

RD and NI calculation

```
Input Pixel Center: 128
Input Q_NE_0:  50, Q_NE_1: 000f00, Q_NE_2: 100, Q_NE_3: 001400
Input Q_NE_4: 150, Q_NE_5: 001e00, Q_NE_6: 200, Q_NE_7: 002800
NI Result: 10101011
RD Result: 10101111
```

Figure 17: NI and RD at R=2

```
Input for R_1:
Q_NE_0_1:  30, Q_NE_1_1: 000900, Q_NE_2_1:  60, Q_NE_3_1: 001200
Q_NE_4_1:  90, Q_NE_5_1: 001b00, Q_NE_6_1: 120, Q_NE_7_1: 002400
Input for R_2:
Q_NE_0_2:  40, Q_NE_1_2: 000d00, Q_NE_2_2:  80, Q_NE_3_2: 001600
Q_NE_4_2: 120, Q_NE_5_2: 001f00, Q_NE_6_2: 160, Q_NE_7_2: 002800
NI Result: 10101011
RD Result: 11111111
```

Figure 18: NI and RD at R=4,6,8

Above images are the result of calculating NI and RD block with expect data and case at R=2,4,6,8. RD at R=2 is a special case with the 8 neighbor points will compare with central pixel.

5 Conclusion and future work

5.1 Conclusion

In conclusion, the integration of the Median Robust Extended Local Binary Pattern (MRELBP) with SystemVerilog in my capstone project has demonstrated significant improvements in texture analysis and hardware implementation. By leveraging the noise reduction capabilities of median filtering and the robust texture representation of MRELBP, I achieved enhanced accuracy and reliability in texture feature extraction. The use of SystemVerilog for hardware description allowed for efficient mapping onto FPGA, ensuring high-speed processing and real-time performance. This project not only highlights the effectiveness of combining advanced algorithms with hardware design but also paves the way for future developments in real-time image processing and pattern recognition applications.

Through this project, I have gained valuable insights into both algorithmic and hardware optimization. I learned the intricacies of median filtering and its impact on texture analysis, as well as the practical challenges of implementing complex algorithms in hardware. This experience has equipped us with a deeper understanding of system design, performance optimization, and the importance of balancing computational efficiency with real-world applicability. These skills and knowledge will be instrumental in my future endeavors in the field of computer vision and hardware development.

5.2 Future work

In next 4 months, I will complete the system and implement on board DE2 or DE10 at lab. Now about the next work, it is that complete the control unit between interpolation and median processing, verify the MRELBP module, start the design of the SVM module. Furthermore, at task 5, for dataset, I will use the Face detector or Object detector datasets for system to compare the result data of extraction step with available dataset for classification. The type of dataset is based on the application of system.

Task	Description	Start Date	End Date	Note
1	Verify module MRELBP	26/12/2024	05/01/2025	
2	Design SVM module	01/2025	01/2025	
3	Verify module SVM	02/2025	02/2025	
4	Verify system (connect two modules)	02/2025	02/2025	
5	Implement on board with a available dataset	03/2025	03/2025	Image and video
6	Design module transfer data from Camera to board	04/2025	04/2025	
7	Verify system with camera	04/2025	05/2025	

Table 18: Timeline of Tasks

References

[1] Lei Sun Kang Yang Min Wei. “Design of median filtering system based on FPGA for large windows”. In: (2018).

[2] Matti Pietika Timo Ojala. “Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns”. In: (July 2002).

[3] Jie Zhou Xingzheng Wang and Jane You. “Robust Texture Image Representation by Scale Selective Local Binary Patterns”. In: (February 2016).

[4] Lukas Sekanina Zdenek Vasicek. “Novel Hardware Implementation of Adaptive Median Filters”. In: (May 2008).