A FPGA Implementation of Low-Complexity Noise Removal

Takeaki Matsubara, Vasily G. Moshyaga, and Koji Hashimoto
Department of Electronics Engineering and Computer Science Fukuoka University
8-19-1 Nanakuma, Jonan-ku, Fukuoka 814-0180, Japan
{matsubara, vasily, khashi}@fukuoka-u.ac.jp

Abstract—Impulse noise removal is a very important preprocessing operation in many computer vision applications. This paper presents a low-complexity noise removal based on a conditional technique and outlines its FPGA implementation for window sizes of (3x3) and (5x5). As the experiments show, the proposed technique performs significantly better than standard median filter and achieves superior image quality. The FPGA implementations are very compact, fast and consume low-power.

Keywords- Impulse noise, median filter, FPGA implementation, image processing

I. INTRODUCTION

Impulse noise frequently corrupts images during picture acquisition or transmission. Generally, two types of impulse noise exist: fixed-valued impulse noise and random valued impulse noise. The fixed value impulse noise (also known as salt-and-pepper noise [1]) is usually reflected by a pixel which has either a minimum or a maximum value in grey-scale image. In contrast, the values of random-valued noisy pixels are distributed uniformly in the grey-scale image within the range of [0, 255]. The well known median filter [2] removes impulse noise signals by changing the luminance value of the target pixel with the median value of those pixels in the filtering window. However, as the number of corrupted pixels in the image increase, the median filter produces poor results. Namely, it blurs image details and causes loss of the useful information in the image. Besides, since most median filters require sorting or inserting/deleting procedures, they become very computationally expensive.

Recently, various adaptive techniques for impulse noise reduction have been proposed [3-11]. The common idea of these techniques is to split the noise processing into two parts: (1) noise detection; and (2) non-linear filter for impulse noise removal. The incoming data is checked and if a noisy pixel is found, the adaptive filter is applied to repair this corrupted pixel. Otherwise, the original pixel is kept. In comparison to median filtering, the adaptive techniques do not process the noise-free pixels, and so reduce the total computational load. However their efficiency strongly depends on correct detection of noisy pixels and their de-noising, which is not a trivial task. Unfortunately, existing solutions often require extensive computations and large memory space.

In this paper we present a novel adaptive median filtering technique that achieves low complexity and high quality processing under low cost requirements on the image de-noising hardware. We also present its FPGA implementation along with the images and comparisons to the related designs.

The next section presents the proposed technique. Section 3 describes the hardware architecture. Section 4 shows experimental results. Section 5 outlines conclusions and work for the future.

II. THE PROPOSED NOISE REMOVAL TECHNIQUE

A. Main idea

The proposed technique targets at removing random-valued impulse noise. Similarly to the other adaptive filtering methods, it consists of two components: noise detector and filter. The noise detector determines whether the pixels are corrupted by the random-valued impulse noise or not. Each noisy pixel is processed by median filtering for reconstruction. Otherwise, there is no fundamental reason to modify the value of a non-noisy pixel, so the median filtering is skipped. The processing flowchart is shown in Fig.1. Below we discuss the technique in details.

B. The noise detector

The noise detector uses two dynamically defined conditional thresholds \(T_{LOW}, T_{HIGH}\) to distinguish corrupted pixels from the noise-free pixels. Let \(f_0\) be the value of the pixel with coordinates \((i,j)\) and \(W(i,j)\) the set of pixels that surround the \((i,j)\) within the test window of \((N \times N)\) pixels in size. Fig.2 shows the relative position of the pixel \((i,j)\) and its \((3 \times 3)\) test window.

To compute the thresholds, the noise detector sorts the values of pixels within \(W(i,j)\) in ascending order. Let the values of three pixels in the middle be \(f_4, f_5, f_6\), and \(f_4 \leq f_5 \leq f_6\). That is \(f_5\) is the median value of \(N^2\) values in \(W(i,j)\).
Then the thresholds $T_{LOW}$ and $T_{HIGH}$ are defined as:

$$
T_{HIGH} =
\begin{cases} 
\frac{f_{i,j} + \alpha}{2}, & \text{if } f_{i,j} + \alpha < \frac{f_{i,j} - \beta}{2} \\
\frac{f_{i,j} + \beta}{2}, & \text{otherwise}
\end{cases}
$$

(1)

$$
T_{LOW} =
\begin{cases} 
\frac{f_{i,j} - \alpha}{2}, & \text{if } f_{i,j} - \alpha > \frac{f_{i,j} - \beta}{2} \\
\frac{f_{i,j} - \beta}{2}, & \text{otherwise}
\end{cases}
$$

(2)

If the detector finds that the value $f_{i,j}$ of pixel $(i,j)$ is larger than $T_{HIGH}$ or lower than $T_{LOW}$, it sets a binary flag $S$ to one. Otherwise, the $S$ is 0. That is,

$$
S =
\begin{cases} 
1, & \text{if } (f_{i,j} \leq T_{LOW}) \text{ or } (T_{HIGH} \leq f_{i,j}) \\
0, & \text{otherwise}
\end{cases}
$$

(3)

Thus, the flag ($S=1$) points out that the pixel $(i,j)$ is corrupted. If $S=0$, the pixel has a noise-free value.

C. The Median Filter

The median filter is activated if and only if the flag $S$ is set. The filter replaces the corrupted value of current processing pixel $(i,j)$ with the median value of those pixels in $W(i,j)$. In contrast to traditional median filters[2] as well as the adaptive center-weight filters[3-4], which replace the central pixel intensity value $f_{i,j}$ in the window $N$ times, our filter replaces $f_{i,j}$ only once, namely if $S=1$. Otherwise, the pixel value remains unchanged. This allows the noise filter to be tuned to the expected noise characteristics (unlike the center weighted median filter) and also assumes that significantly less change is made to the original noiseless pixels (unlike the traditional median filter). It should be noticed that the proposed conditional median filter turns to the standard median filter for $T_{LOW} = T_{HIGH}$.

III. EXPERIMENTAL EVALUATION

To examine the properties of the proposed technique, we performed several tests. The first test was dedicated to finding the parameters ($\alpha$ and $\beta$) of the thresholds ($T_{HIGH}$ and $T_{LOW}$), which provide the best picture quality in terms of Peak-Signal to Noise ratio (PSNR) and mean squared error (MSE). We used a standard (512x512) 8-bit grey-scale test image (Lena) and the same image corrupted with noise of 10\% and 20\% density. The noisy image was then processed by the proposed technique (3x3 window) with $\alpha$ varying from 0 to 60, while for each value of $\alpha$, the value of $\beta$ was changing from 0 to 100. Those values of $\alpha$ and $\beta$, which maximized PSNR were selected. Thus we obtained $\alpha=16$, $\beta=60$.

Table I shows the PSNR and MSE of the noisy image, the image filtered by standard median filter (MF) and the image filtered by the proposed technique obtained for two window sizes: (3x3) and (5x5), respectively. Fig. 3 illustrates the visual quality of corresponding images for $N=3$, 30\% noise density. As we see the results produced by the proposed technique are considerably better than those of the standard median filter.

Next, we compared the results obtained by the proposed technique with those produced by the related approaches. Totally five de-noising methods have been tested: 1) standard median filter (MF) of size (3x3) [2]; 2) the differential rank impulse detector (DRID) [6]; 3) the alpha-trimmed mean-based technique (ATMBM) [8], 4) the edge-preserving image de-noising technique (EPID) [11] and 4) the proposed
technique. The threshold parameters of the related techniques were set as described in the papers. Table II shows the PSNR values of the images with noise densities varying from 10% to 40%. As one see our technique achieves better results than the DRID, ATMBM, and EPID, i.e. has less difference is almost invisible (see fig.4).

IV. HARDWARE IMPLEMENTATION

Table III compares the techniques in terms of required operations, processing time and the buffer size. Here, comp shows comparisons; abs - absolute value computations; add – additions; sub – subtractions, and mult – multiplications. The processing time was obtained from the PC with 2.8GHz Pentium CPU and 512MB memory. Based on the results we conclude that our technique requires less arithmetic operations than the DRID, ATMBM, and EPID, i.e. has less computational complexity, while utilizing the minimum number of memory buffers.

IV. HARDWARE IMPLEMENTATION

Fig.4 outlines hardware architecture for implementing the proposed technique. The architecture consists of three major blocks: register bank, sorting unit, and threshold generation and noise detection unit. Below we briefly describe each of them in details.

A. Register bank

The register bank consists of $N \times N$ registers to store all pixel values of the current window. When the window is shifted from the current location to the next one, only $N$ new values are read into the Register Bank (RB) and the rest $(N-1) \times N$ pixel values are shifted to their right registers, respectively. Eventually, $N$ values are loaded into the shifter in parallel. The shifter operates at $N$ time’s higher clock frequency than the register file, moving the received data one position to the right in every clock cycle. With each data move one pixel sample enters the sorting unit. Thus its takes $N$ machine cycles (or $N$ clock cycles of the shifter) to process all pixels in the window.

B. Sorting Unit

To implement sorting in hardware, we use 1D-structure proposed in [12]. The circuit consists of $M=N^2$ cascaded blocks, one for each window rank, as shown in Fig.6. Each block $i$ is composed of one $n$-bit register ($R_i$), one $k$-bit $(k=\log_2M)$ counter ($P_i$), and a simple data-transfer logic. All blocks are connected to the Reset line and the global input, $X$, through which they receive the incoming sample. The data-transfer logic allows blocks to receive and transmit the content of their $R$ and $P$ values from/to their neighbors. The registers $R_1, R_2, ..., R_M$ store samples in descending order; so at time $t$ the maximum value is always at the left (in $R_1$); the minimum value at the right (in $R_M$), and the median is in the register $R_m$. We assume the sorting unit runs at the twice higher clock frequency than the shifter: i.e. it performs two clock cycles for each clock cycle of the shifter. Eventually, $N$ values are loaded into the shifter in parallel. The register bank consists of $N \times N$ registers to store all pixel values of the current window. When the window is shifted from the current location to the next one, only $N$ new values are read into the Register Bank (RB) and the rest $(N-1) \times N$ pixel values are shifted to their right registers, respectively. Eventually, $N$ values are loaded into the shifter in parallel. The shifter operates at $N$ time’s higher clock frequency than the register file, moving the received data one position to the right in every clock cycle. With each data move one pixel sample enters the sorting unit. Thus its takes $N$ machine cycles (or $N$ clock cycles of the shifter) to process all pixels in the window.

IV. HARDWARE IMPLEMENTATION

Fig.4 outlines hardware architecture for implementing the proposed technique. The architecture consists of three major blocks: register bank, sorting unit, and threshold generation and noise detection unit. Below we briefly describe each of them in details.

Table III compares the techniques in terms of required operations, processing time and the buffer size. Here, comp shows comparisons; abs - absolute value computations; add – additions; sub – subtractions, and mult – multiplications. The processing time was obtained from the PC with 2.8GHz Pentium CPU and 512MB memory. Based on the results we conclude that our technique requires less arithmetic operations than the DRID, ATMBM, and EPID, i.e. has less computational complexity, while utilizing the minimum number of memory buffers.

IV. HARDWARE IMPLEMENTATION

Fig.4 outlines hardware architecture for implementing the proposed technique. The architecture consists of three major blocks: register bank, sorting unit, and threshold generation and noise detection unit. Below we briefly describe each of them in details.

Table III compares the techniques in terms of required operations, processing time and the buffer size. Here, comp shows comparisons; abs - absolute value computations; add – additions; sub – subtractions, and mult – multiplications. The processing time was obtained from the PC with 2.8GHz Pentium CPU and 512MB memory. Based on the results we conclude that our technique requires less arithmetic operations than the DRID, ATMBM, and EPID, i.e. has less computational complexity, while utilizing the minimum number of memory buffers.

IV. HARDWARE IMPLEMENTATION

Fig.4 outlines hardware architecture for implementing the proposed technique. The architecture consists of three major blocks: register bank, sorting unit, and threshold generation and noise detection unit. Below we briefly describe each of them in details.

Table III compares the techniques in terms of required operations, processing time and the buffer size. Here, comp shows comparisons; abs - absolute value computations; add – additions; sub – subtractions, and mult – multiplications. The processing time was obtained from the PC with 2.8GHz Pentium CPU and 512MB memory. Based on the results we conclude that our technique requires less arithmetic operations than the DRID, ATMBM, and EPID, i.e. has less computational complexity, while utilizing the minimum number of memory buffers.

IV. HARDWARE IMPLEMENTATION

Fig.4 outlines hardware architecture for implementing the proposed technique. The architecture consists of three major blocks: register bank, sorting unit, and threshold generation and noise detection unit. Below we briefly describe each of them in details.
odd clock cycle, the counters are incremented. The counter \( P_2 \) reaches 5, which is the limit for \( M=5 \), and overflows sending the request signal \( z_3 \) to all the blocks on its right. By receiving this signal, the blocks 3,4 and 5 move their values to the adjacent blocks on the left, while the values of \( R_3 \), \( P_3 \) are reset to zero. Thus \( R_3 \) becomes 3, \( R_4=135 \), \( R_5=31 \), \( P_4=0 \), \( R_5=0 \), \( P_5=2 \), \( P_3=4 \), \( P_4=3 \), \( P_5=1 \), \( P_4=0 \) while the aged sample 140 is removed from the window.

At any even clock cycle, the input sample is compared to all values stored in the registers. The comparators (C) produce true signals \( T_i=1 \) if \( X>R_i \); otherwise \( T_i=0 \). Thus, since \( X=145 \) is larger than the value of the register \( R_2 \) but lower than that of \( R_1 \), the signal \( T_1 \) generated by \( C_1 \) becomes 0 while \( T_2 \) (from \( C_2 \)) is 1. These signals enforce the multiplexor in front of \( R_2 \) to select input 1 while all the multiplexors at the right side of \( R_2 \) select input 2. Thus the contents of registers \( R_2-R_5 \) is moved one position the right and the sample \( X \) is written to the \( R_2 \). Due to this partial data movement left and right, the circuit preserves the sample ordering obtained at the present cycle, while resoring only the new incoming sample at a new clock cycle.

C. Threshold generator and noise detector

Fig.7 shows the internal structure of the threshold generation and the noise detection unit. Here \( C \) denotes comparators and \( + \) denotes adders. The circuit implements equations (1)-(3) in one clock cycle, producing the signal \( S=1 \), when the pixel \( (i,j) \) is noisy and \( S=0 \), otherwise.

Based on this signal, the multiplexer in Fig.3 selects either the median or the original pixel value and writes it to the location \( (i,j) \) in the register file. After that the content of the RF is shifted left and the processing repeats. Overall the circuit (Fig.3) takes six cycles of internal (sorting) clock to process one pixel.

D. Experimental evaluation

To evaluate the circuit, we experimentally designed two hardware implementations for \( W=3 \times 3 \) and \( 5 \times 5 \), respectively, (8-bit word-length of samples) by using Altera Design Tools and implemented the circuits in Altera Cyclone II FPGA IC board (EP2C20F484C7N, 20K logic cells, 234K memory cells, 1.2V power supply).

Table IV summarizes the designs in terms of the total number of total logic cells (LC) and memory cells (MC) used in the designs, the critical path delay, the maximum clock frequency and the power consumption. The total power consumption was evaluated based on 50 random samples as a sum of dynamic and static power computed from the generated Altera layout. Based on the number of logic and memory cells used, we conclude that the designs are very compact, fast and consume low power. Although the designs were automatically generated, the maximum clock frequency achievable is high.

V. CONCLUSIONS

This paper presented a novel approach for impulse-noise reduction. Although the technique has very low computational complexity, it provides superior quality of results in terms of PSNR and image quality. Two FPGA implementations of the proposed noise reduction technique for the window size of 3x3, and 5x5 show the technique results in compact hardware which can operate at high clock frequency and low power. Currently we are working on custom VLSI chip implementation of the proposed method.

### REFERENCES


