IMPROVED MEDIAN FILTER USING CONDITIONAL TECHNIQUE AND ITS HARDWARE IMPLEMENTATION

Marek Kraft and Andrzej Kasiński

Institute of Control and Information Engineering, Poznań University of Technology, Piotrowo 3A, 60-965, Poznań, Poland
email: Marek.Kraft(at)put.poznan.pl

ABSTRACT

Median filtering is a very important preprocessing operation in many computer vision applications. This paper presents an improved median filtering algorithm, which uses a simple conditional technique. The presented algorithm shows significantly better performance than standard median filter and is suitable for hardware implementation. To illustrate the performance of the algorithm, the results of comparison with the standard median filter and resulting images for both algorithms are given as illustration. A hardware implementation of the conditional median filter is also presented, that extends the applicability of this algorithm in the area of real time image processing. The designed hardware filter enables achieving of frame rates of hundreds of frames per second for a 1024 × 1024, 8-bit gray-scale picture.

1. INTRODUCTION

Median filtering is a powerful instrument used in image processing. The traditional median filtering algorithm, without any modifications gives good results. There are many variations to the classical algorithm, aimed at reducing computational cost or to achieve additional properties. Such modified median filters are for example the center-weighted median filters [1], weighted median filters [2] (e.g. for edge detection) etc. Median filters are used mainly to remove salt-and-pepper noise. Doing this, they preserve edges in the image (preserve their location and do not affect their steepness, unlike gaussian filters), but unfortunately median filtering may destroy small features in the image [3]. A way to avoid it is to apply center-weighted median filtering instead of a plain median, but the drawback of this solution is the deterioration of the filter’s ability to suppress impulse noise.

Common drawback of various kinds of the median filtering is their computational cost. Computing a two-dimensional median for a \( N \times N \) window, requires sorting of \( N \times N \) elements for every image pixel and choosing the median value for the output. After the sorting, each queue element is assigned a value called a rank, specifying its position in the queue as a result of sorting.

Therefore, using median filtering in any real-time vision system requires a significant computational power. One way to speedup the computations is to implement the algorithm in hardware, e.g. with the help of FPGA circuits. The rationale behind this is to use the FPGA’s inherent ability to execute operations in parallel. Moreover, programmable logic creates the possibility to tailor the implementation to the user’s needs. All this results in a significant speedup over the software implementations by using sequential processors. One drawback of hardware-based algorithm development is the complexity of the design process as implementing algorithmically complex operations is very difficult. Median filtering, like many other low-level image processing algorithms is fairly simple - the main problem in this case is the amount of data to handle.

In this paper we present the possibility to improve the performance of median filter by introducing a simple conditional technique to the filter’s architecture. We also present the results of a working, performant FPGA implementation of this algorithm with a \( 3 \times 3 \) window, along with test images and the results of comparison of the implemented algorithm and the traditional median filter.

2. MEDIAN FILTERING

The median filter picks each pixel in the image in turn and looks at its neighbors to decide whether or not it is representative of its context. Instead of simply replacing the pixel value with the mean of neighborhood pixel values, it replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the neighborhood by their values and then by replacing the pixel being considered with the median pixel value. Doing so, it sorts out the pixel values that differ significantly from the other values in the window and thus being classified as noise. This can however lead to a situation, in which the fine details in the image are also treated as noise, because they ‘occupy’ only a small part of the window under inspection and also their intensity values significantly differ from the other pixels in the window. To prevent this, a modification to the original median filtering algorithm called the center weighted median has been proposed. In this type of filtering, to the central pixel of the processed window is assigned a weight. The weight \( n \) means, that the center pixel intensity value will be replicated \( n \) times before ordering the pixels queue. This increases the chance, that fine details would not be destroyed in the process of filtering, but unfortunately this degrades the filter’s ability to handle noise.

3. THE IMPLEMENTED ALGORITHM

The algorithm is based on a simple conditional technique. Like in the traditional median filtering algorithm, the pixels within the processed window are first subject to sorting. The rank of the central pixel is used to determine the output of the filter. If this rank is higher or equal than some higher threshold value \( T_H \) or lower or equal than some lower threshold value \( T_L \), the output of the filter would be the median value, otherwise the pixel remains unchanged. This allows the algorithm to be tuned to the expected noise characteristics (unlike the center weighted median filter) and also assures, that sig-
significant less change is made to the original noiseless pixels (unlike in the original median filtering algorithm). Let us denote the processed window by \( w \) the output of the filter by \( F(w) \), and the center pixel in the window by \( w_c \). Then the equation for the filter can be formulated as follows:

\[
F(w) = \begin{cases} 
    \text{MED}(w) & \text{for } \text{rank}(w_c \geq T_H) \text{ or } \text{rank}(w_c \leq T_L) \\
    w_c & \text{otherwise.}
\end{cases}
\]

For \( T_H = T_L \), the conditional median filter reduces to a standard median filter. It is also a less general case of a LUM filter.

We have performed several tests to examine the properties of the algorithm described above. In our test, we used a standard test image and the same image corrupted with the salt-and-pepper noise with a density of 10%. The threshold values \( T_H = 8 \) \( T_L = 2 \) have been selected. Conditional and traditional median filtering algorithms were then applied to the noisy image. PSNR and MSE of both of the filtered images in relation to the original image without noise has also been computed. The resulting values can be found in table 1.

The conditional algorithm produces results that are evidently better. The filtered image is not as 'flattened' as it is the case with images that are the result of traditional median filtering. This 'flattening' is the result of destroying fine details in the image as the result of the filtering process. Also the PSNR values show, that the similarity measure is much better in the case of the conditional algorithm. Please note, that the PSNR and MSE for the traditional median filter are even worse than for the noisy image (although the filtered image is more pleasing to the eye...). This confirms, that many fine details in the image were lost. Filtering with a \( 5 \times 5 \) window with \( T_H = 24 \) and \( T_L = 2 \) reveals, that the results obtained from the conditional median filter outperform the traditional median filter even further, even for a severe noise with a 30% density. Resulting PSNR and MSE for this case are also given in table 1. Resulting images are shown in figure 1.

### Table 1: PSNR and MSE of the noisy image, and the noisy image filtered with standard and conditional median filter with respect to the original image.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Noisy image</th>
<th>Standard median</th>
<th>Conditional median</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR [dB]</td>
<td>37.05</td>
<td>36.25</td>
<td>38.5</td>
</tr>
<tr>
<td>MSE</td>
<td>12.83</td>
<td>15.43</td>
<td>9.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Noisy image</th>
<th>Standard median</th>
<th>Conditional median</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSNR [dB]</td>
<td>32.35</td>
<td>34.04</td>
<td>38.66</td>
</tr>
<tr>
<td>MSE</td>
<td>37.86</td>
<td>25.67</td>
<td>8.84</td>
</tr>
</tbody>
</table>

### 4. THE ARCHITECTURE DESCRIPTION

Because hardware implementations of median filtering are considered faster than their software counterparts, many different architectures to compute a median have been proposed. The most basic and intuitive of those architectures is the 2D or odd-even sorter - an equivalent to the bubble sorting algorithm, swapping the adjacent samples that are not in order on every iteration [4]. This architecture, although simple, requires a large amount of resources - the number of the pipeline stages in this case is equal to the number of samples within the processed window. Modifications have been proposed to reduce the amount of resources required for the implementation. Other approaches rely on specific properties of median filtering (like the triple-input sorter presented in [5]), or make use of the possibilities provided by the hardware. The latter include for example a very resource-efficient bit-serial sorter presented in [6].

The drawback of those algorithms from our point of view is that although a median value is determined, the rank values in most of the cases are not. Therefore, an algorithm that supports the rank evaluation for each sample within the window has to be used. Upon investigation, the median evaluation method presented in [7] has been chosen. Except for the above mentioned property, the advantages of this method are its flexibility and anticipated processing speed. The method is based on the concept of rank modification as a result of comparation of samples within the processing window with the samples that come in and go out of the processing window. Principal drawback of this method is its resource demand. However even the cheapest of the modern FPGAs are able to handle this task without much problems. For our \( 3 \times 3 \) conditional median filter implementation, we used a device from the Spartan 3 family from Xilinx. The filter’s working principle is based on the observation, that every sample leaving the window will modify the rank of the sample remaining in the window buffer by decreasing it if the value of the remaining sample is greater then the one of the outgoing sample. Such modification is
made for each outgoing sample. On the other hand, the rank of every respective sample is increased if the value of this sample is greater or equal to the one of an incoming sample. Again, such modification is repeated for every incoming sample. The architecture for performing the task of rank update consists therefore of two groups of comparators. One group compares each the samples remaining in the window with each of the incoming samples, while the other compares the remaining samples with the outgoing samples. The ranks of the incoming pixels is determined by checking the results of comparison with the samples remaining in the processed window. The schematic outline of this architecture is depicted in figure 2.

The solution is pipelined to assure reasonable propagation delays. In the first cycle, the value of every sample remaining in the window is compared to each of the incoming sample values with. The output of each comparator is a logical '1' if a remaining sample is greater or equal than the incoming sample. For a $3 \times 3$ we have 3 1-bit comparisons. These results are grouped in a register. Such registers (marked A on figure 2) are invoked for every sample remaining in the window. In a similar way, the remaining samples are compared with the outgoing samples. If a respective sample value is greater than the outgoing sample value, the output of the comparator is set to a logical '1'. Therefore, we have 3 single bit comparison results. These results are grouped in other registers (marked B on figure 2) and additionally by comparing the incoming samples among themselves. The schematic outline of this architecture is depicted in figure 2.

The conditional median processor was implemented and tested by using VHDL as hardware description language together with Xilinx’s ISE 8.1 package and Mentor Graphics’ ModelSim XE III 6.1e. The design was targeted at FPGA’s from Xilinx’s Spartan 3 family – XC3S200-4FT256 part (one of the smallest in the Spartan 3 family, with the lower speed grade) has been used. The FPGA was fitted on the S3BOARD evaluation board from Digilent. The initial testing was done via behavioral and post place and route
Table 2: Resource usage and speed of the implemented design. The values in percent are given with respect to all resources available.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of block RAMs</td>
<td>2 (16%)</td>
</tr>
<tr>
<td>No. of flipflops</td>
<td>292 (7%)</td>
</tr>
<tr>
<td>No. of LUTs</td>
<td>682 (18%)</td>
</tr>
<tr>
<td>No. of slices</td>
<td>488 (25%)</td>
</tr>
</tbody>
</table>

Figure 3: Results of filtering with the conditional median filter implemented in hardware. From left to right: original image, noisy image, filtered image.

Simulation. After confirming the circuit’s proper operation in this way, the design was implemented in the FPGA. For a functional test, a gray-scale image with 256 × 256 resolution, corrupted with salt and pepper noise has been sent to the board via the serial RS232 interface. The output samples were received with the same interface and visualized on PC. For testing purposes, an additional serial interface receiver and transmitter IP core along with a supervising finite state machine for synchronization was implemented. We used the Xplorer script from Xilinx to determine the best implementation parameters with the maximum clock speed chosen as the priority for this design. The summary of resources used for the circuit, can be found in table 2. The maximum operating speed of the median processor with the FIFO buffers is 175 MHz, according to the synthesis and implementation tools from Xilinx. Such processing speed should be enough to handle a 1024×1024 8-bit gray-scale image data stream with the speed of about 150 FPS by using low-cost FPGA. The output image, along with the noisy image and the original image, is shown in figure 3. The results confirm the circuit’s correct operation.

6. CONCLUSIONS

An improved median filtering algorithm, demonstrating significantly better performance than the classical median filter, has been discussed in this paper. This filtering algorithm preserves the ability of a standard median filter in terms of handling impulse noise, but also overcomes the most important drawbacks of traditional median filtering, i.e. the possibility of destroying fine details in the original image. The filtering algorithm is relatively simple, allowing a straightforward hardware implementation. Such implementation, tested on a physical device, has also been presented. The maximum clock rate of the designed filter let it to meet the demands of real-time applications. The use of inexpensive FPGA reduces the size, cost, and power consumption of a complete solution. The designed hardware filter does not require an external memory, and can be plugged directly into the incoming data-stream, i.e. from a camera. Although the hardware implementation uses significantly more resources than some proposed architectures, it can be easily handled by modern FPGAs, leaving enough resources to implement further processing stages. It can be also easily extended to process color images by implementing median filtering on single channels in the RGB, YUV or YCrCb representations.

Further research plans includes the implementation of the conditional median filter with an increased window size and incorporating the filter into a complete computer vision system, along with other developed algorithms.

REFERENCES


