Available online at www.sciencedirect.com
ScienceDirect
Pattern Recognition 40 (2007) 890–897
PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY
www.elsevier.com/locate/pr

Fast mean filtering technique (FMFT)

S. Rakshit, A. Ghosh*, B. Uma Shankar

Machine Intelligence Unit, Indian Statistical Institute, 203 B.T. Road, Kolkata - 700108, India

减少的时间是通过使用基本的存储和获取操作实现的，与图像或邻域大小无关。该方法已在各种灰度图像和邻域大小上进行了测试，取得了良好的效果。

Abstract

This article presents a novel method for mean filtering that reduces the required number of additions and *eliminates the need for division altogether*. The time reduction is achieved using basic *store-and-fetch* operations and is irrespective of the image or neighbourhood size. This method has been tested on a variety of greyscale images and neighbourhood sizes with promising results. These results indicate that the relative time requirement reduces with increase in image size. The method's efficiency also improves significantly with increase in neighbourhood size thereby making it increasingly useful when dealing with large images.

Keywords: Mean filtering; Spatial averaging; Image smoothing; Noise reduction; Image processing

1. Introduction

A major component of image processing deals with spatial filtering of images by convolution masks. Blurring, sharpening, noise reduction, edge-detection and various other spatial enhancements necessitate the use of masks. The *mean* or *averaging filter* is one of the more important masks and finds extensive use in a variety of *image processing* and pre-processing tasks [1–5].

While dealing with large images and/or large neighbourhood (window) sizes, application of these filters can be quite time-consuming, if not efficiently implemented. For example, the basic implementation of a mean filter for smoothing a 1024×1024 image with a 7×7 neighbourhood needs over 50×10^6 additions and 1×10^6 divisions. We must address and exploit the inherent redundancies in this basic implementation and do away with re-computations. In this article we propose a method that reduces the number of additions to a large extent and completely eliminates the need for any division in calculating the mean.

The rest of the paper is organized as follows. Section 2 deals with the concept of mean filter and its applications. Section 3 presents the basic method of implementing the mean filter and explains its inherent redundancies. This is followed by the proposed method in Section 4. Implementation details and results are tabulated in Section 5. Section 6 gives the conclusion and scope for further research.

2. Mean filtering

Image smoothing refers to any image-to-image transformation designed to *smoothen* or *flatten* an image by reducing the rapid pixel-to-pixel variation in greylevels [1–5]. Smoothing may be accomplished by applying an averaging mask that computes a weighted sum of the pixel greylevels in a neighbourhood and replaces the centre pixel with that greylevel. The image is blurred and its brightness retained as the mask coefficients are all-positive and sum to one. The mean filter is one of the most basic smoothing filters. Mean filtering is usually thought of as a convolution operation as the mask is successively moved across the image until every pixel has been covered [2]. Like other convolutions it is based around a kernel, which represents the shape

* Corresponding author. Tel.: +91 33 2575 3110; fax: +91 33 2578 3357.
E-mail address: ash@isical.ac.in (A. Ghosh).

0031-3203/$30.00
doi:10.1016/j.patcog.2006.02.008

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{n^2}\begin{bmatrix}1 & 1 & \cdots & 1\\ 1 & 1 & \cdots & 1\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 1 & \cdots & 1\end{bmatrix}$$

(a) (b)

Fig. 1. (a) 3×3 mean filter; (b) general $n \times n$ mean filter.

Fig. 2. Effect of mean filtering: (a) original crowd image; (b) Blurred image with $n = 7$ and (c) 11.

and size of the neighbourhood to be sampled when calculating the mean. Larger kernels are used when more severe smoothing is required. Fig. 1(a) shows a mean mask for a 3×3 window, while a more general $n \times n$ mask is shown in Fig. 1(b).

Variations on the mean filter include *threshold averaging* [6], wherein smoothing is applied subject to the condition that the centre pixel greylevel is changed only if the difference between its original value and the average value is greater than a preset threshold. This causes the noise to be smoothed with a less blurring in image detail.

It must be noted here that the smoothing operation is equivalent to low-pass filtering as it eliminates edges and regions of sudden greylevel change by replacing the centre pixel greylevel by the neighbourhood average. It effectively eliminates pixel greylevels that are unrepresentative of their surroundings. Noise, due to its spatial decorrelatedness, generally has a higher spatial frequency spectrum than the normal image components. Hence, a simple low-pass filter can be very effective in noise cleaning [1].

Smoothing filters thus find extensive use in *blurring* and *noise removal* [1]. Blurring is usually a preprocessing step bridging gaps in lines or curves, helping remove small unwanted detail before the extraction of relevant larger objects. Figs. 2(b) and (c) show the effect of averaging for various window sizes n (7 and 11) [2].

The mean filter is an important image processing tool and finds use in Gaussian noise reduction [1–5], blurring before thresholding to eliminate small detail, bridging gaps in broken characters for improved machine perception in OCRs, cosmetic processing of human faces images to reduce fine skin lines and blemishes, etc. The mean is also used as a derived or texture feature in image segmentation process [7,8]. Let us now take a look at the basic implementation of this mean filter.

3. The basic method

Consider an L level image $F(P \times Q)$, whose pixel greylevels are stored in a 2-D array, say, $data[P][Q]$ such that $data[0][0]$ and $data[P-1][Q-1]$ contains the first and last pixels, respectively [2]. In order to apply the mean-filter to this image over a rectangular neighbourhood window $m \times n$ (m & n are odd positive integers), the image must be *appropriately padded* by some method like replication of boundary rows and columns [1] at the four sides of the input image. The average greylevel of a pixel $data[row][col]$ over an $m \times n$ neighbourhood is then calculated as follows,

For row $= 0$ *to* $P - 1$
  For col $= 0$ *to* $Q - 1$
    Sum = Sum of $m \times n$ *pixel greylevels in the neighbourhood centred at* $data[row][col]$
    $newdata[row][col] = Sum/(m \times n)$
  End
End

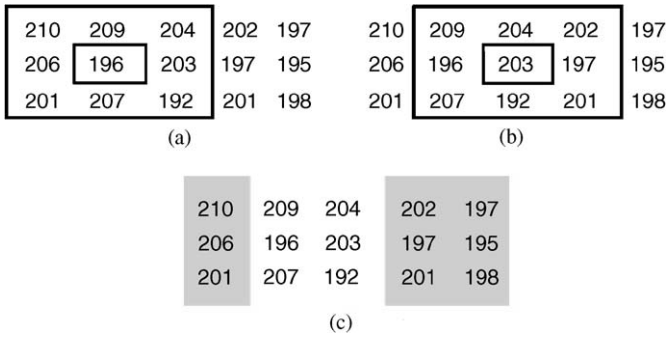

Fig. 3. (a) Neighbourhood around 196; (b) neighbourhood around 203; (c) un-shaded portion represents 6 out of 9 common pixels.

Here *newdata*[P][Q] is a 2-D array containing the computed averages. This basic method, though simple to implement is inefficient due to unnecessary re-computations in terms of both additions and divisions. The required number of additions is $\boldsymbol{P \times Q \times m \times n}$, while that of divisions is $\boldsymbol{P \times Q}$. Thus, both are in direct relation to the image size and increases proportionately with it.

A major redundancy stems from the fact that the sum over a neighbourhood is repeatedly computed as the mask is shifted from one pixel to the next. As the mask is shifted by only one pixel at a time, we need only add that new pixels and subtract the old ones instead of computing the whole sum all over again [2]. Consider the 3×3 neighbourhood window around the pixel 196 as shown in Fig. 3(a). When this neighbourhood is shifted right to the next pixel, 203 we get Fig. 3(b). The pixels common to both these neighbourhoods are shown un-shaded in Fig. 3(c) and their sum can be carried forward to the next neighbourhood without the need for re-computation.

The algorithm for the aforementioned addition reduction procedure is summarized below,

For $row = 0$ *to* $P - 1$
  $col = 0$
  $Sum = Sum$ *of* $m \times n$ *pixel greylevels in the neighbourhood centred at data*[row][col]
  newdata [row][col] $= Sum/(m \times n)$
  For $col = 1$ *to* $Q - 1$
    $Sum = Sum - m$ *number of pixel greylevels in column*$(col - n/2)$
    $Sum = Sum + m$ *number of pixel greylevels in column*$(col + n/2)$
                                [$n/2$ is the integer part of n by 2]
    newdata [row][col] $= Sum/(m \times n)$
  End
End

At the beginning of every new row, the sum of $m \times n$ pixel greylevels is computed. This total neighbourhood pixel greylevel summation is carried out P times as against the $P \times Q$ times in the normal case. The required number of additions is now reduced to $\boldsymbol{P \times m \times n + P \times Q \times m \times 2}$, which is almost equal to $\boldsymbol{P \times Q \times m \times 2}$.

The proposed column-summation store-and-fetch method [9,10] will reduce this number drastically.

4. Proposed methods

In this section we discuss the proposed algorithms for addition and division reduction.

4.1. Proposed addition reduction

The number of additions can be further reduced by grouping the m pixels in each column of a window (around the central pixel) having m rows and n columns, and storing their sums in an array, say, $s[n]$ as shown in Fig. 4. The size of the array is n and it is used as a circular array.

In this case, when the neighbourhood is shifted one pixel to the right, only one new column (of the new window) sum needs to be added and one old column (of the old window) sum subtracted. The number of additions is further reduced

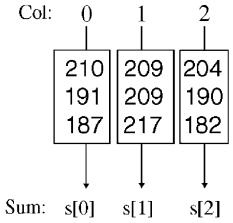

Fig. 4. Grouping of pixels in columns over a window of 3 × 3.

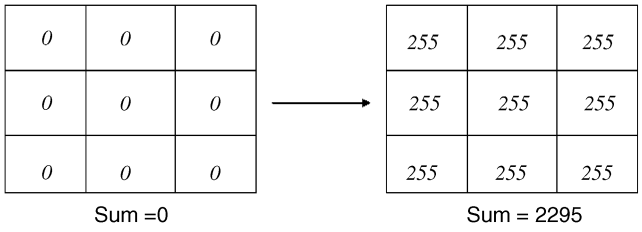

Fig. 5. Range of summations of 9 pixel greylevels for a 3 × 3 neighbourhood.

to $m + 2$ per pixel from $m \times 2$. Thus we now require only $\boldsymbol{P \times Q \times (m + 2)}$ additions. The algorithm is as follows,

For row $= 0$ *to* $(P - 1)$
    $col = 0$
    Store sum of pixel greylevels of columns $0, 1, \ldots, n - 1$ *in* $s[0], s[1], \ldots, s[n - 1]$
                [each column contains m elements.]
    $Sum = s[0] + s[1] + \cdots + s[n - 1]$
    $newdata[row][col] = Sum/(m \times n)$
    $j = 0$
    For $col = 1$ *to* $(Q - 1)$
        $Sum = Sum - s[j]$
        $s[j] =$ *Sum of pixel greylevels in column* $(col + n/2)$
                [$n/2$ is the integer part of n by 2]
        $Sum = Sum + s[j]$
        $newdata[row][col] = Sum/(m \times n)$
        $j = j + 1$
        If $j >= n$
            $j = 0$ [Note: this ensures the circular nature of the array]
        End
    End
End

For a 1024×1024 image, filtered over a 7×7 neighbourhood, the required number of additions is reduced to a mere 9×10^6, which is less than 20% of the original number. Let us now look into the aspect of division reduction.

4.2. Proposed division reduction

Division, being a floating-point operation, contributes most significantly towards the computation time in any filtering algorithm. Thus, it is of primary importance to try and reduce their number. Let us look into the innate redundancy in the basic method as regards divisions, by considering a 3×3 neighbourhood-averaging example. The possible range of summation of 9 greylevels, (considering an 8-bit greylevel image), is from 0 to 2295 (255×9) as shown in Fig. 5. It may be noted here that in case of 7-bit images we only need to store the averages from 0 to 1143 (127×9).

Thus if all pixels are black, the summation is 0, while if all pixels are white the summation is 2295, with each sum having a corresponding average as shown in Fig. 6.

For example when the sum of 9 pixel greylevels is between 0 and 8, the average is 0 (considering the floor function for simplicity). Similarly for a sum in between 2286 and 2294, the average is 254. So we need only store these averages in an array

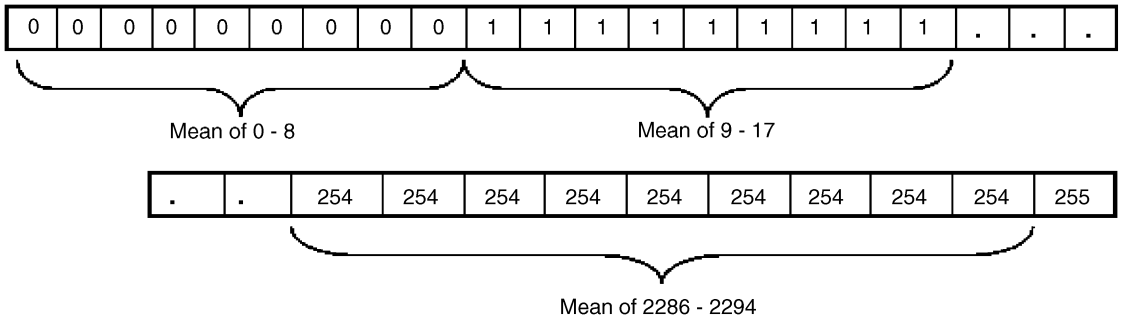

Fig. 6. Storing averages in an array with the sum as index.

(say *mean*[]) of length 2296, $(255 \times m \times n + 1)$, and access them using the sum of neighbourhood pixel greylevels as the array index. The mean storing algorithm is as follows,

$j = k = 0$
For $i = 0$ *to* $255 \times m \times n$
  If $k = m \times n$
    $k = 0$
    $j = j + 1$
  End
  $mean[i] = j$
  $k = k + 1$
End

In this case the number of divisions is reduced to $\mathbf{255 \times m \times n + 1}$. This number may be reduced to *zero* by using multiple additions, as they are far less computationally intensive. The final algorithm is as follows,

For $row = 0$ *to* $(P - 1)$
  $col = 0$
  Store sum of pixel greylevels of columns $0, 1, \ldots, n - 1$ *in* $s[0], s[1], \ldots, s[n - 1]$
    [each column contains m elements.]
  $Sum = s[0] + s[1] + \cdots + s[n - 1]$
  $\boldsymbol{newdata[row][col] = mean[Sum]}$
  $j = 0$
  For $col = 1$ *to* $(Q - 1)$
    $Sum = Sum - s[j]$
    $s[j] =$ *Sum of pixel greylevels in column* $(col + n/2)$
      [$n/2$ is the integer part of n by 2]
    $Sum = Sum + s[j]$
    $\boldsymbol{newdata[row][col] = mean[Sum]}$
    $j = j + 1$
    If $j >= n$
      $j = 0$ [Note: this ensures the circular nature of the array]
    End
  End
End

It is identical to the one in Section 4.1, except for the lines marked bold, where the method of mean storage has been changed to one of direct-access. Going back to our original example, a 1024×1024 image with a neighbourhood of 7×7 now requires around 9×10^6 additions and 0 divisions instead of the original 50×10^6 additions and 1×10^6 divisions.

5. Implementation and results

The images used for the purpose of testing the time requirements of the above methods were 7-bit and 8-bit greyscale images of various sizes, like 128×128, 256×256, 480×640, 512×512, 600×800, 768×1024, 1024×1024, 1200×1600, etc. The average was calculated over various neighbourhood window sizes ranging from 3×3 to 11×11.

Fig. 7. (a) Lenna image; (b) blurred version.

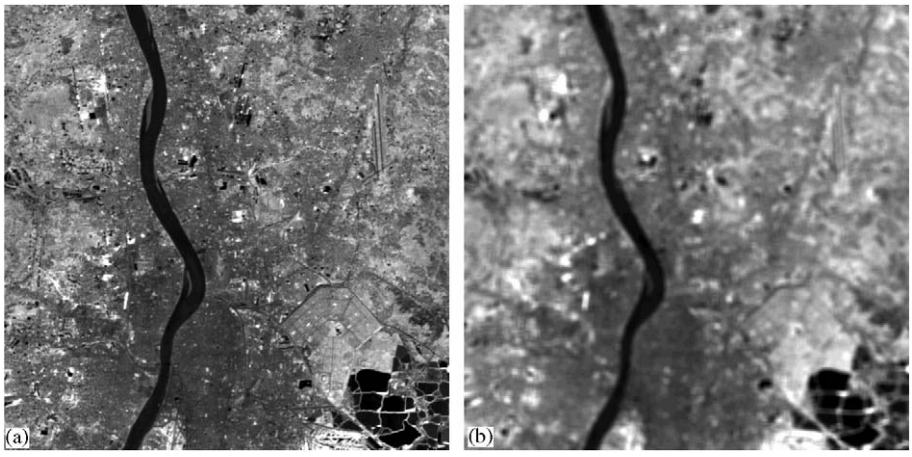

Fig. 8. (a) Enhanced (contrast stretched) Kolkata IRS image, (b) blurred version.

Table 1
Number of computations

Method	Additions	Divisions
Basic method	$P \times Q \times m \times n$	$P \times Q$
Common addition reduction	$P \times Q \times m \times 2$	$P \times Q$
Proposed addition reduction	$P \times Q \times (m+2)$	$P \times Q$
Proposed division reduction	$P \times Q \times (m+2)$	0

Figs. 7(a) and 8(a) give two such sample images, Lenna and the IRS-1A (Band-4) contrast-enhanced image of Kolkata [6]. Due to poor illumination of the IRS-1A image the actually object classes present are not clearly visible in the original image. A contrast-enhanced image is thus presented here for better viewing. Figs. 7(b) and 8(b) give the results of 7×7 mean filtering on these images. The aforementioned methods were all implemented in the Standard C++ language [11].

Number of computations for each of the above methods in terms of image size, $P \times Q$ and neighbourhood window size, $m \times n$ is tabulated above in Table 1.

The relative computation time for various image and neighbourhood sizes are summarized below in Table 2. Here *BM* stands for the basic method, *RAM* for the reduced addition method and *ZDM* for the zero-division method. (Note: Here the time taken by the 128×128 image for *ZDM* over a 3×3 neighbourhood is considered to be the basis for comparison).

Here, Figs. 9(a)–(c) show the plot of relative time taken against image size by the basic method (diamond) and the *ZDM* (square) for 3×3, 7×7, and 11×11 neighbourhood windows, respectively. As is evident from the graph, the utility of the

Table 2
Relative time taken for different implementations over different neighbourhood sizes for various image sizes

Image no.	Image size	Size of neighbourhood														
		3 × 3			5 × 5			7 × 7			9 × 9			11 × 11		
		BM	RAM	ZDM	BM	RAM	ZDM	BM	RAM	ZDM	BM	RAM	ZDM	BM	RAM	ZDM
1	128 × 128	61	11	10	77	33	11	71	55	22	88	55	33	88	60	44
2	256 × 256	72	61	22	93	65	61	135	68	66	159	72	66	208	71	66
3	512 × 512	127	88	77	209	99	82	330	110	88	472	110	93	670	126	110
4	480 × 640	143	93	88	236	110	93	379	132	110	576	143	138	818	165	154
5	600 × 800	181	121	99	324	132	110	538	160	126	835	159	143	1192	198	171
6	768 × 1024	269	165	104	456	132	117	873	214	131	1340	186	154	1944	214	176
7	1024 × 1024	341	148	114	648	237	132	1143	220	171	1736	253	209	2560	335	292
8	1200 × 1600	577	263	231	1104	319	285	2011	379	297	3070	494	351	4449	505	412

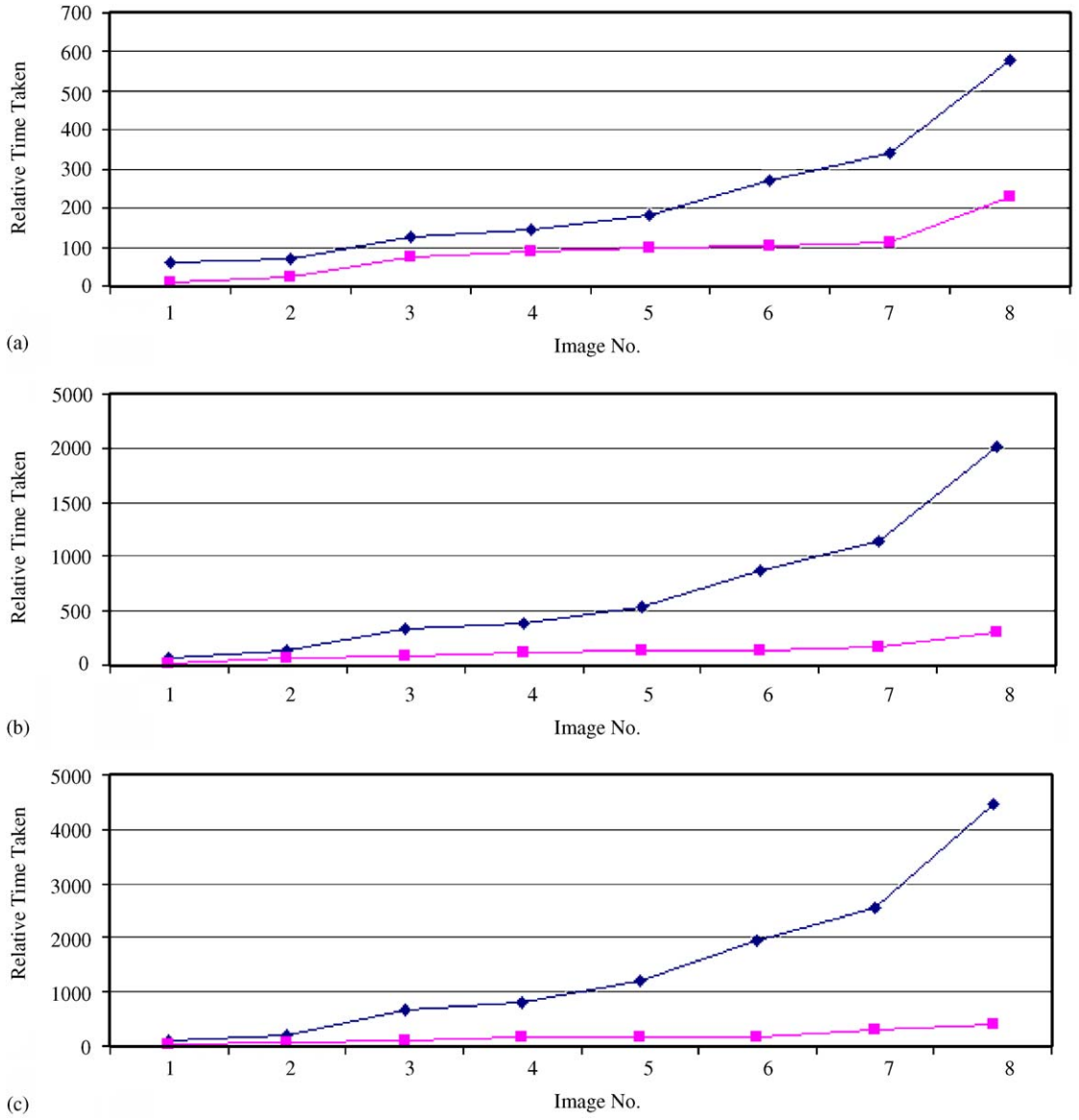

Fig. 9. Plot of time taken by BM (diamond) and ZDM (square) for (a) 3 × 3, (b) 7 × 7 and (c) 11 × 11 neighbourhood against image size.

optimized implementation increases with increase in image as well as neighbourhood size. This property is particularly useful when working on large remote sensing images like the IRS-1A image (25.2×10^6 pixels in 4 bands).

It is interesting to note that the gain in time reduction (ZDM/BM) for the 1200×1600 image increases from 2.5 times for the 3×3 neighbourhood window to 10.8 times for the 11×11 neighbourhood. In other words an image-processing task that would previously have taken 11 h to complete would now take only 1 h.

6. Conclusions and future work

The fast mean filtering technique (FMFT) presented here has successfully reduced the time requirements for smoothing operations with mean filters, especially for large images. This implementation reduces the number of additions to approximately $1/n$th of the original number, where $n \times n$ is the neighbourhood size and completely eliminates the division operation by store-and-fetch methods very efficiently. The gain in the performance and usefulness of this method has been demonstrated for different images.

The various applications of mean filtering, as described in detail at the beginning of this paper can all benefit tremendously from this improved implementation. Noise removal by *threshold averaging* in remote-sensing applications is one such important example. The effect of this filtering technique will be tremendous in such an application. This method can easily be extended to higher bit-level greyscale images or colour images.

Acknowledgements

Authors would like to thank the DST—Govt. of India and University of Trento, Italy, the sponsors of the India—Trento Project on Advanced Research (ITPAR), under which a project titled "Advanced Techniques for Remote Sensing Image Processing" is being carried out at the Machine Intelligence Unit, Indian Statistical Institute, Kolkata. The valuable appreciation and suggestions of the reviewer is also gratefully acknowledged.

References

[1] W.K. Pratt, Digital Image Processing, third ed., Wiley, New York, 2001.
[2] R.C. Gonzalez, R.E. Woods, Digital Image Processing, second ed., Pearson Education, 2002.
[3] R.C. Gonzalez, R.E. Woods, S.L. Eddins, Digital Image Processing Using MATLAB, Pearson Education, 2004.
[4] A.K. Jain, Fundamentals of Digital Image Processing, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[5] A. Rosenfeld, A.C. Kak, Digital Picture Processing, vols. I & II, Academic Press, New York, 1982.
[6] J.A. Richards, X. Jia, Remote Sensing Digital Image Analysis An Introduction, third ed., Springer, Berlin, 1999.
[7] S.K. Pal, A. Ghosh, B. Uma Shankar, Segmentation of remotely sensed images with fuzzy thresholding, and quantitative evaluation, International J. Remote Sensing 21 (11) (2000) 2269–2300.
[8] A. Jain, D. Zongker, Feature Selection: Evaluation, Application, and Small Sample Performance, IEEE Trans. Pattern Anal. Mach. Intell. 19 (2) (1997) 153–158.
[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., Prentice-Hall of India, 2001.
[10] R. Sedgewick, Algorithms in C, Parts 1–4, third ed., Pearson Education, 1998.
[11] B. Stroustrup, The C++ Programming Language, Special ed., Pearson Education, 2000.