Analyzing Image Filtrations by Enhanced Fuzzy Logic with Multi Quality Inputs

Ramesh Tiwari
Department of CSE
Dr. B R Ambedkar NIT
Jalandhar, India

Renu Dhir
Department of CSE
Dr. B R Ambedkar NIT
Jalandhar, India

ABSTRACT
In this paper, we describe the Image Filtration through Fuzzy Logics in four different scenarios of Image Input as 3*3,9*9, 17*17 and 25*25 division blocks and iterating fuzzy equation on it for 2 and 8 times at constant amplification factor of 10. The input image selected for analysis is PGM (Portable Gray Map), dividing input images into matrix of m*n blocks. The input image is analyzed for multiple iterations and difference in output is significantly marked for MSE and PSNR.

General Terms
Image Processing, Noise Reduction and Impulse Noise.

Keywords
Fuzzy Filter and PGM image Filtration.

1. INTRODUCTION
A PGM image represents a grayscale graphic image. For most purposes, a PGM image can just be thought of an array of arbitrary integers, and all the programs in the world that think they're processing a grayscale image can easily be tricked into processing something else [1], this extra ordinary characteristics makes PGM better than other formats for analyzing Image Filtration when it comes to divide images into blocks of m*n for input.

Noise can be systematically introduced into images during acquisition and/or transmission of images [2, 3]. The impulse noise has the tendency of either relatively high or relatively low, thus it could severely degrade the image quality and some loss of information details [4].

Various filtration techniques have been proposed for removing such noise in the past and are well known that linear filters could produce serious image blurring[5,6]. Therefore non linear filters are widely exploited and improved. During the past years the variety of filter classes are developed such as (1) Classical filters; (2) Fuzzy Classical Filters i.e. fuzzy logic based filters that are modification or extension of classical filters; (3) Fuzzy Filters [7,8,9].

In this paper we exploit the use of Fuzzy Filters and resemble their usage with increase in number of iterations. The paper analyzed fuzzy filter for image inputs as block of 9*9, 17*17 and 25*25 at constant amplification factor of 10 and compared with the 3*3 block input image[10].

2. SYSTEM DESCRIPTION

2.1 Working Principle
Fuzzy Logic requires some numerical parameters in order to operate such as what is considered significant error and significant rate of change of error, but exact values of these numbers are usually not critical unless very responsive performance is required in which case empirical tuning would determine them. For example, automatic temperature control systems normally uses only one temperature feedback sensor for a single room whose data is subtracted from the command signal to compute "error" and then time-differentiated to yield the error slope or rate of change of error, hereafter called "divergence". Error might have units of ℃ and a small error considered to be ±2℃ while a large error is ±5℃. The "divergence" might then have units of degrees/min with a small divergence being ±2℃/min and a large one being ±5℃/min w.r.t. command signal. These values don't have to be symmetrical and can be tweaked once the system is operating in order to optimize performance.

2.2 Framework Developed
This system presents a technique for filtering noise in images by a fuzzy filter in two steps.

First, the filter estimates a "fuzzy derivative" in order to be less sensitive to local variations due to image structures such as edges.

Second, the membership functions are adapted accordingly to the noise level to perform "fuzzy smoothing."

For each pixel that is processed, the first stage computes a fuzzy derivative. Second, a set of varying fuzzy rules is fired to determine a correction term. These rules make use of the fuzzy derivative as input. Fuzzy sets are employed to represent the properties. While the membership functions are fixed and adapted after each iteration The general idea behind the filter is to average a pixel using other pixel values from its neighborhood, but simultaneously to take care of important image structures such as edges. To determine neighborhood, we divide the input image into blocks of m*n and identify new boundaries. The main concern of the proposed filter is to distinguish between local variations due to noise and due to image structure. In order to accomplish this, for each pixel we derive a value that expresses the degree in which the derivative in a certain direction is small. Such a value is derived for each direction corresponding to the neighboring pixels of the processed pixel by a fuzzy rule. The further construction of the filter is then based on the observation that a small fuzzy derivative most likely is caused by noise, while a large fuzzy derivative most likely is caused by an edge in the image.

Consequently, for each direction we will apply two fuzzy rules that take this observation into account and thus distinguish between local variations due to noise and due to image structure.

2.3 Fuzzy Rule

2.3.1 Fuzzy Derivative Estimation

For filtering we want a good indication of the edges, while to find these edges we need filtering. In our approach, we start by looking for the edges. We try to provide a robust estimate by applying fuzzy rules.

Consider the 3*3 neighborhood of a pixel (x, y) as display in Figure 1. A simple derivative at the central pixel position (x, y) in the direction D (D ∈ dir = {NW, W, SW, S, SE, E, NE, N}) is defined as the difference between the pixel at (x, y) and its neighbor in the direction D. This derivative value is denoted by ∇ D(x, y). For example,

$$\nabla_N(x, y) = I(x, y - 1) - I(x, y)$$

$$\nabla_{NW}(x, y) = I(x - 1, y - 1) - I(x, y)$$

Next, the principal of the fuzzy derivative is based on the following observation. Consider an edge passing through the neighborhood of a pixel (x, y) in the SW – NE direction. The derivative value $\nabla_{NW}(x, y)$ will be large, but also derivative values of neighboring pixels perpendicular to the edge's direction can expected to be large.

NW	N	NE
W	(x, y)	E
SW	S	SE

Figure 1: Neighborhood of a central pixel (x, y).

In table 1, we give an overview of the pixels we use to calculate the fuzzy derivative for each direction. Each direction corresponds to a fixed position and the sets specify which pixels are considered with respect to the central pixel (x, y).

To compute the value that express the degree to which the fuzzy derivative in a certain direction is small, we will make use of the fuzzy set small. The membership function $m_K(u)$ for small properties is the following:

$$m_K(\mathrm{u}) = \begin{cases} 1 - \frac{|u|}{K}, & 0 \le |u| \le K \\ 0, & |u| > K \end{cases}$$

where K is an adaptive parameter [10].

2.3.2 Adaptive Threshold Selection

We start by dividing the image in small m*n non-overlapping blocks. For each block B, we compute a rough measure for the homogeneity of this block by considering the maximum and minimum pixel value [11].

$$\mu = 1 - \frac{max_{(x,y)\epsilon B}\, I(x,y) - \; min_{(x,y)\epsilon B}\, I(x,y)}{L}$$

Where L represent the number of gray levels.

Table 1. Pixel involved in calculating the fuzzy derivatives in each direction

Direction	Position	Set w.r.t. (x, y)
NW	(x - 1, y - 1)	{(-1,1),(0,0),(1,-1)}
W	(x - 1, y)	{(0,1),(0,0),(0,-1)}
SW	(x - 1, y + 1)	{(1,1),(0,0),(-1,-1)}
S	(x , y + 1)	{(1,0),(0,0),(-1,0)}
SE	(x + 1, y + 1)	{(1,-1),(0,0),(-1,1)}
E	(x + 1, y)	{(0,-1),(0,0),(0,1)}
NE	(x + 1, y - 1)	{(-1,-1),(0,0),(1,1)}
N	(x , y - 1)	{(-1,0),(0,0),(1,0)}

2.4 Implementation

The proposed system is implemented in language java as,

```
//smoothing algorithm
int K=-1,prevK=0,L=imgin.getMaxGray();
int neighbor[]=new int[8];
int simpderiv[][]=new int[3][8];
double fofxsmall[][]=new double[3][8];
double fuzzyderiv[]=new double[8];
double fofxpositive[]=new double[8];
double fofxnegative[]=new double[8];
double positivetruthness[]=new double[8];
double negativetruthness[]=new double[8];
double delta,fK;
//divide image into m*n blocks
int kvals[]=new int[9*9];
//for each block
for(int m=0;m<imgin.getRows();m=m+dim)
{
for(int n=0;n<imgin.getCols();n=n+dim)
{
//get pixelvalues
for(int tr=m,t=0;tr<m+dim;tr++)
{
for(int tc=n;tc<n+dim;tc++)
{
kvals[t]=imgin.getPixel(tr,tc);
t=t+1;
}
}
//get neighborhood pixel intensity values
neighbor[0]=imgin.getNeighbor(r,c,Globals.NW);
```

```
neighbor[1]=imgin.getNeighbor(r,c,Globals.W );
neighbor[2]=imgin.getNeighbor(r,c,Globals.SW);
neighbor[3]=imgin.getNeighbor(r,c,Globals.S );
neighbor[4]=imgin.getNeighbor(r,c,Globals.SE);
neighbor[5]=imgin.getNeighbor(r,c,Globals.E );
neighbor[6]=imgin.getNeighbor(r,c,Globals.NE);
neighbor[7]=imgin.getNeighbor(r,c,Globals.N );
```

//fuzzy derivative estimation

//calc simple derivative of (r,c)

```
for(int t=0;t<8;t++) simpderiv[0][t]=neighbor[t]-inval;
```

//calc simple derivative of perpendicular point-1

```
simpderiv[1][0]=imgin.getPixel(r-1 +1,c-1 -1)-
imgin.getPixel(r+1,c-1);
simpderiv[1][1]=imgin.getPixel(r+1,c-1)-imgin.getPixel(r+1,c);
simpderiv[1][2]=imgin.getPixel(r+1 +1,c-1 +1)-
imgin.getPixel(r+1,c+1);
simpderiv[1][3]=imgin.getPixel(r+1,c+1)-imgin.getPixel(r,c+1);
simpderiv[1][4]=imgin.getPixel(r+1 -1,c+1 +1)-
imgin.getPixel(r-1,c+1);
simpderiv[1][5]=imgin.getPixel(r-1,c+1)-imgin.getPixel(r-1,c);
simpderiv[1][6]=imgin.getPixel(r-1 -1,c+1 -1)-imgin.getPixel(r-
1, c-1);
simpderiv[1][7]=imgin.getPixel(r-1,c-1)-imgin.getPixel(r,c-1);
```

//calc simple derivative of perpendicular point-2

```
simpderiv[2][0]=imgin.getPixel(r-1 -1,c-1 +1)-imgin.getPixel(r-
1,c+1);
simpderiv[2][1]=imgin.getPixel(r-1,c-1)-imgin.getPixel(r-1,c);
simpderiv[2][2]=imgin.getPixel(r+1 -1,c-1 -1)-imgin.getPixel(r-
1, c-1);
simpderiv[2][3]=imgin.getPixel(r+1,c-1)-imgin.getPixel(r,c-1);
simpderiv[2][4]=imgin.getPixel(r+1 +1,c+1 -1)-
imgin.getPixel(r+1,c-1);
simpderiv[2][5]=imgin.getPixel(r+1,c+1)-imgin.getPixel(r+1,c);
simpderiv[2][6]=imgin.getPixel(r-1 +1,c+1 +1)-
imgin.getPixel(r+1,c+1);
simpderiv[2][7]=imgin.getPixel(r-1,c+1)-imgin.getPixel(r,c+1);
}
```

//find standard deviation

```
int sum=0;
double mean,dsum=0;
for(int t=0;t<dim*dim;t++) sum=sum+kvals[t]; //find sum
sd=Math.sqrt(dsum/((dim*dim)-1));
sumsd=sumsd+sd;
cnt=cnt+1.0;
}
mean=(double)sum/(double)(dim*dim);
for(int t=0;t<dim*dim;t++) dsum=dsum+((kvals[t]-
mean)*(kvals[t]-mean));//for each pixel
for(int r=0;r<imgin.getRows();r++)
{
for(int c=0;c<imgin.getCols();c++)
{
int inval=imgin.getPixel(r,c);//current pixel intensity value
//calc membership value for 'small' fuzzy set
for(int t=0;t<8;t++)
{
fofxsmall[0][t]=SmallFuzzySet.fofx(simpderiv[0][t],K);
fofxsmall[1][t]=SmallFuzzySet.fofx(simpderiv[1][t],K);
fofxsmall[2][t]=SmallFuzzySet.fofx(simpderiv[2][t],K);
}
```

The java interface developed is shown in Figure 2.

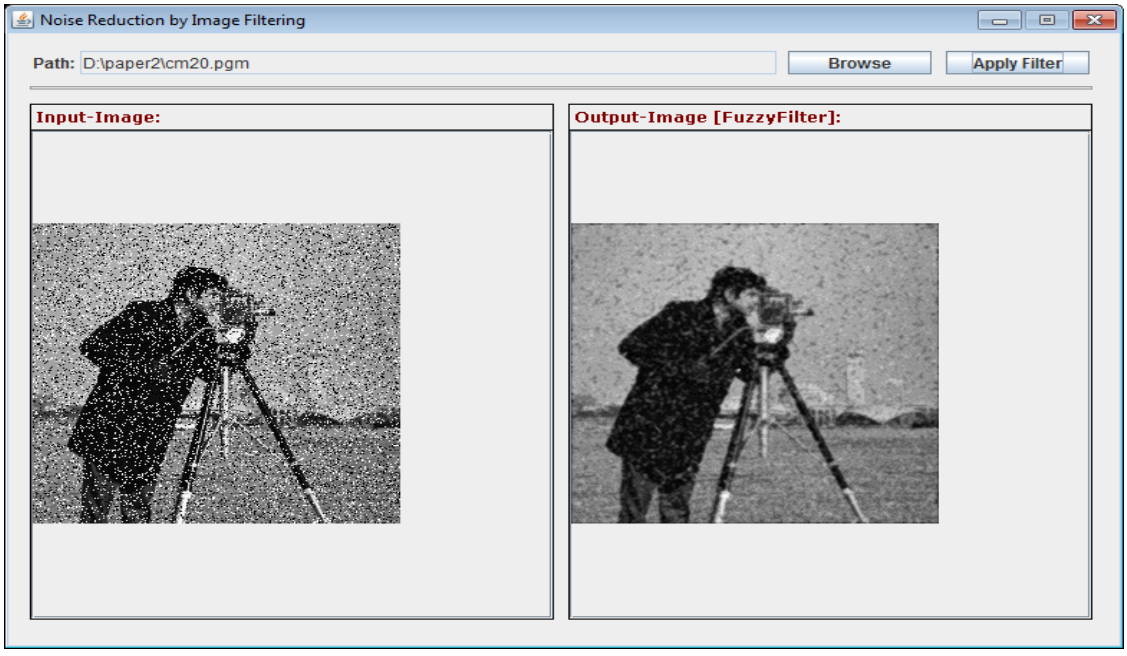

Figure 2: Developed java interface.

3. RESULTS AND DISCUSSIONS

To apply Fuzzy Logic on Image we first break images into blocks, this image partition brings more input for fuzzy Equations and enhance the results. Here we study the system for four cases 3*3, 9*9, 17*17, 25*25 block inputs with an constant amplification factor of 10.

The PGM image with 20% noise density is used for the analysis in this paper, is shown in Figure 3.

Figure 3: Corrupted Image with 20% noise density.

3.1 3*3 Partition

Figure 4 and Figure 5 shows Fuzzy Filtered image whose input block is divided into 3*3 matrices after 2 and 8 iterations.

Figure 4: After Two Iterations.

Figure 5: After Eight Iterations.

3.2 9*9 Partition

Figure 6 and Figure 7 shows Fuzzy Filtered image whose input block is divided into 9*9 matrices after 2 and 8 iterations.

Figure 6: After Two Iterations.

Figure 7: After Eight Iterations.

3.3 17*17 Partition

Figure 8 and Figure 9 shows Fuzzy Filtered image whose input block is divided into 17*17 matrices after 2 and 8 iterations.

Figure 8: After Two Iterations.

Figure 9: After Eight Iterations.

3.4 25*25 Partitions

Figure 10 and Figure 11 shows Fuzzy Filtered image whose input block is divided into 25*25 matrices after 2 and 8 iterations.

Figure 10: After Two Iterations.

Figure 11: After Eight Iterations.

The performance evaluation of the filtering operation is quantified by the MSE(Mean Squared Error) and PSNR(Peak Signal-to-Noise Ratio) are calculated using the following standard formula:

$$\text{MSE} = \frac{1}{mn} + \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [\, \text{I(i,j)} - \text{K(i,j)} \,]^2$$

where,
I(i,j) is original image without noise,
K(i,j) is filtered image,
m and n is the total number of pixels in the horizontal and vertical dimensions of the image I(i,j) and K(i,j).

$$\text{PSNR} = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right)$$

where, MAX_I is the maximum possible pixel value of the image.

For better image filtration MSE should be low and PSNR should be high. When the two images are identical, the MSE will be zero and PSNR is undefined.
The MSE and PSNR of Corrupted Image with 20% noise density is 0.06221 and 12.06113dB respectively. The MSE and PSNR of the filtered image after two iteration is shown in the table 2.

Table 2. MSE and PSNR of the filtered image after two iteration

	MSE	PSNR
3*3 (Figure 4)	0.00928	20.32530
9*9 (Figure 6)	0.00607	22.16980
17*17 (Figure 8)	0.00624	22.04841
25*25 (Figure 10)	0.00649	21.87345

The MSE and PSNR of the filtered image after eight iteration is shown in the table 3.

Table 3. MSE and PSNR of the filtered image after eight iteration

	MSE	PSNR
3*3 (Figure 5)	0.00897	20.47085
9*9 (Figure 7)	0.00600	22.21845
17*17 (Figure 9)	0.00615	22.11345
25*25 (Figure 11)	0.00644	21.90952

4. CONCLUSIONS

The focus of this paper is to analyze non linear Fuzzy Image filter when same input image is divided into different input blocks and iterated multiply. The multiple iteration on input image for constant amplification factor of 10, results into sharp and much clear image due to removal of high and low pixel intensity noise by taking multiple fuzzy derivatives on neighboring pixel and repetitively joining them into a group, for fuzzy smoothing. Table 2 and

3 depicts better result for 9*9 input image with increase in iterations.

5. REFERENCES

[1] Jef Poskanzer on his official site http://netpbm.sourceforge.net/doc/pgm.html

[2] Zhou, Yan, Tang, Quan-hua, Jin, Wei-dong, 2008. Adaptive fuzzy median filter for images corrupted by impulse noise in Congress on image and signal processing IEEE Conference. Volume 5 Page(s): 265 - 269

[3] Schulte, S., De Witte, V., Nachtegael, M., Van der Weken, D., Kerre, E.E., 2006. Fuzzy two-step filter for impulse noise reduction from color images in IEEE Transactions on Image Processing Volume: 15, Issue: 11 Page(s): 3567 – 3578.

[4] Pei-Eng Ng, Kai-Kuang Ma, 2006. A switching median filter with boundary discriminative noise detection for extremely corrupted images in IEEE Transactions on Image Processing Volume: 15, Issue: 6 Page(s): 1506 – 1516.

[5] Z. Deng, Z Yin, and Y Xiong., 2007 High probability impulse noise-removing algorithm based on mathematical morphology in IEEE Signal Process Lett. Page(s): 31-34.

[6] Young Sik Choi, Krishnapuram, R., 1997, A robust approach to image enhancement based on fuzzy logic in IEEE Transactions on Image Processing, Volume: 6 Issue: 6, Page(s): 808 - 825,

[7] Yuewei Lin, Bin Fang, Yuanyan Tang, 2010, Image Restoration Using Fuzzy Impulse Noise Detection and Adaptive Median Filter in Chinese Conference on Pattern Recognition, Page(s): 1 - 4 .

[8] Haixiang Xu, Xiaorui Yue, 2009, An Adaptive Fuzzy Switching Filter for Images Corrupted by Impulse Noise in Sixth International Conference on Fuzzy Systems and Knowledge Discovery Volume: 3, Page(s): 383 – 387.

[9] Nachtegael, M., Schulte, S., Van der Weken, D., De Witte, V., Kerre, E.E., 2005. Fuzzy filters for noise reduction: the case of gaussian noise in The 14th IEEE International Conference on Fuzzy Systems, FUZZ '05. Page(s): 201 – 206.

[10] Van De Ville, D., Nachtegael, M., Van der Weken, D., Kerre, E.E., Philips, W., Lemahieu, I., 2003. Noise reduction by fuzzy image in IEEE Transactions on Fuzzy Systems Volume: 11 , Issue: 4 Page(s): 429–436.

[11] H.Haussecker and H.Tizhoosh,1999 Handbook of Computer Vision and Applications, Page(s): 708– 753.