IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY & MEDICINE

UNIVERSITY OF LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Study of two new spatial domain image fusion methods

Laurent Oudre
Supervisor : Dr. T. Stathaki

This report is submitted in partial fulfilment of the requirements for the Degree of Master of Science (MSc) and the Diploma of Imperial College (DIC)

SEPTEMBER 2007

Abstract

The purpose of image fusion is to create a perceptually enhanced image from *K* multi-focus or multi-sensors images. In the methods we are about to describe we do not *a priori* know the ground truth image: these are blind fusion methods. There are mostly two groups of fusion methods depending on the way they are applied: transform domain methods and spatial domain methods. The Dispersion Minimisation (DMF) and Kurtosis Maximisation (KMF) techniques we are going to discuss are spatial domain methods, that is to say that the fusion is simply performed on the image itself by combining the *K* input images using appropriate weights for the different pixels.

Our aim is therefore to estimate the different weights that optimally measure the contribution of each pixel in source images to the fused one. The key issue is to improve visual perception by summing the weighted source images. In order to evaluate the weights we are using iterative methods which use cost functions based on two parameters: the dispersion (for the DMF method) and the kurtosis (for the KMF). The optimisation of these cost functions enables us to get a fused image which is supposed to be least-distorted that the input ones.

We also introduce some improvements to these methods such as optimal learning rates or the use of the notion of neighbourhood.

Table of contents

Acknowledgments

I would really like to thank Dr. Tania Stathaki for her precious help and support throughout this project.

I would also like to thank Dr. Nikolaos Mitianoudis for his help especially in the Matlab implementation of the methods.

Chapter One: Introduction

1.1 Image fusion

Let have K source images: $\boldsymbol{X}_1, \cdots, \boldsymbol{X}_K$ describing the same true scene $\boldsymbol{F}$ from different sensors of the same basic type or from different types of sensors. Our aim is to create from these images a fused image $\boldsymbol{Y}$ which is going to be perceptually enhanced. The composite image should contain a more useful description of the scene than provided by any of the individual source. This fused image should be more useful for human visual or machine perception. This task of combining images and form one better image is called *image fusion*.

Image fusion has been used in many fields such as aerial and satellite imaging, medical imaging, robot vision etc… In recent years image fusion has become an important and useful technique for image analysis, computer vision, concealed weapon detection and autonomous landing guidance [1]. Image fusion can be performed in two domains: the transform domain or the spatial domain.

1.2 Spatial domain and transform domain

As far as the transform domain fusion methods are concerned the input images are first transformed then fused and the result is converted back by an inverse transform. Popular transform domain fusion methods are for example the DT-WT (Dual-Tree Wavelet Transform) [2] or the ICA (Independent Component Analysis) [3]. In these methods the fusing coefficients are calculated with fusion rules which use either a pixel base or a region base.

The methods we are going to describe are spatial domain ones. That is to say that we just work on the input images directly. We estimate weights for each input image and for each pixel basically with iterative methods which optimize a cost function.

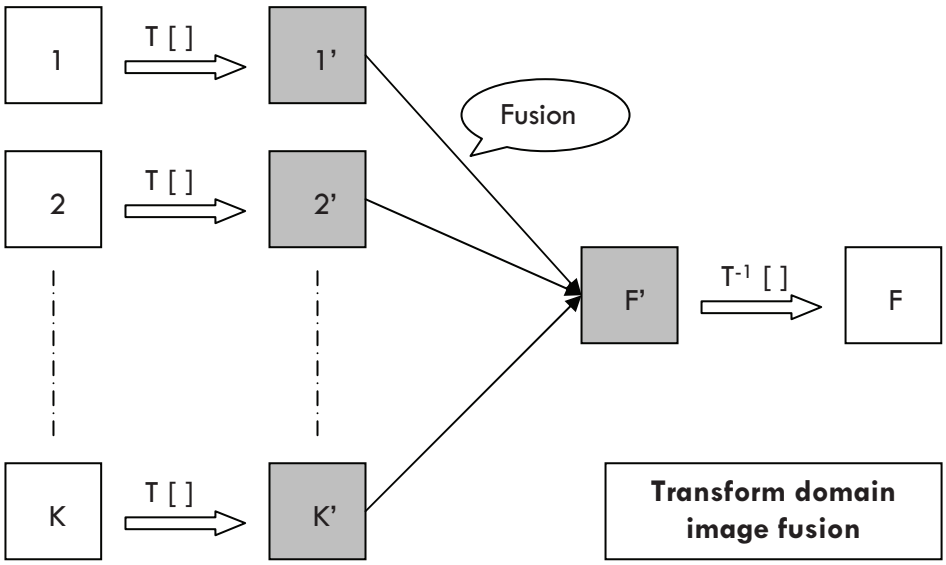


Fig 1.1 : Block diagram of a generic transform domain image fusion scheme

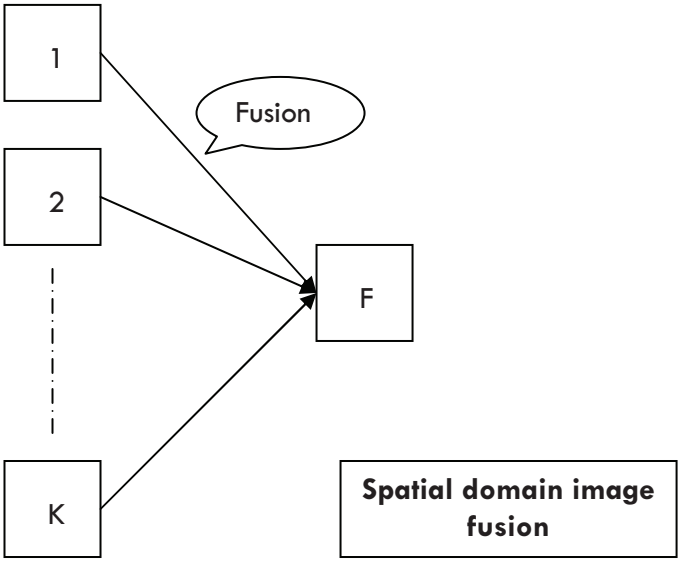


Fig 1.2 : Block diagram of a generic spatial domain image fusion scheme

1.3 Project outline

In order to really understand the mechanisms behind the fusion rules a real mathematical background and some precise notations are needed. That is why the **second chapter** of this thesis will be dedicated to the notations, the definitions and the problem formulation. Even if this part can seem a little off-putting we really thought it was necessary for the understanding.

The **third chapter** is dedicated to the description of the methods, that is to say the DMF and the KMF, as well as the improved methods (Robust and With Neighbourhood).

The first method we shall relate is the DMF (Distortion Minimization Fusion). This spatial domain fusion technique utilizes the cost function of one of the most studied and implemented method: the CM (Constant-Modulus) algorithm, and the notion of dispersion [4]. This iterative process updates at every step the weights for the pixels minimizing the cost function which is also a function of the dispersion of the unknown original image.

The second method we shall describe is actually an extension of the first one. Instead of working with the dispersion we use the Central Limit Theorem and the property of smooth operators to assume that the non-Gaussianity is an indicator of image quality. The parameter we take to measure this non-Gaussianity is the absolute value of kurtosis. This is why this method is called the KMF (Kurtosis Maximization Fusion) [5]. The methods works basically just like the DMF that is to say that at every step the weights for the pixels are updated in order to maximize a cost function.

The three left methods we shall introduce are actually just improvements of the previous ones. The Robust methods enable us to calculate automatically the parameters of the cost function which has for effect to decrease the number or iterations needed for good results. The With Neighbourhood one uses the idea that often in image processing it is better to work with estimation over a neighbourhood than just with a pixel value.

The **fourth chapter** is a complete presentation of the results we got with these methods.

We first introduce all the metrics useful in order to evaluate the performances with numbers instead of just the human visual perception. We will see that it is actually quite tricky to say objectively which method gives the best results and that it often depends on the expected results or the field of application.

We are then going to test six sets of images (multi-focus and multi-sensors) with different types of distortion in order to know where our methods work better. We are of course going to try common used and famous methods so we can objectively evaluate our methods.

Finally we shall assess our methods and see in which fields they can be relevant.

The **fifth chapter** is a description of the work that has been done during the project, that is to say the development of the software, and the algorithms of the methods.

Chapter Two: Notations, definitions and problem formulation

2.1 Notations

In the following paper :

$\boldsymbol{X}$ is an image

x is a scalar (eg : pixel)

$\underline{\mathbf{x}}$ is a vector (eg : image in lexicographic order)

$\underline{\underline{\mathbf{X}}}$ is a matrix (eg : image in matrix form)

If $\underline{\mathbf{x}}$ is a vector, $x(i)$ is a scalar corresponding the the ith element of the vector $\underline{\mathbf{x}}$.

If $\underline{\underline{\mathbf{X}}}$ is a matrix, $\underline{\mathbf{x}(i)}$ is a column vector corresponding to the ith column of the matrix $\underline{\underline{\mathbf{X}}}$.

* * *

If $\boldsymbol{X}$ is a non-zero mean image, then $\widetilde{\boldsymbol{X}}$ is the zero mean version of the image $\boldsymbol{X}$.

$$\widetilde{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{m}_X$$

* * *

If $\underline{\mathbf{x}} = [x(1) \quad \cdots \quad \cdots \quad x(N)]$ is a row vector, we will abuse the notation and consider that $\underline{\mathbf{x}}^k = [x^k(1) \quad \cdots \quad \cdots \quad x^k(N)]$.

2.2 Different notions of expectation

In the following paper, we use two notions of expectation :

- The **global expectation** E can apply to images, row vectors, and matrix. This is actually exactly the mean.

eg : * $E\{\boldsymbol{X}\}$ is the mean value of the image $\boldsymbol{X}$

* $E\{\underline{\mathbf{x}}\}$ is the mean value of the vector $\underline{\mathbf{x}}$

* $E\{\underline{\underline{\mathbf{X}}}\}$ is the mean value of the matrix $\underline{\underline{\mathbf{X}}}$

- The **local expectation** E_n can apply to scalars and column vectors. In this case it is the mean of the values of the pixels belonging to the $L \times L$ neighbourhood of the considered pixel.

eg : * $E_n\{x(i)\}$ is the value of the mean of the pixels in the neighbourhood of the ith pixel.

* If $\underline{\mathbf{x}(i)} = \begin{bmatrix} x_1(i) \\ \vdots \\ \vdots \\ x_K(i) \end{bmatrix}$ is a column vector, then $\underline{\mathbf{E}_n\{\underline{\mathbf{x}(i)}\}} = \begin{bmatrix} E_n\{x_1(i)\} \\ \vdots \\ \vdots \\ E_n\{x_K(i)\} \end{bmatrix}$ is just the column vector containing the local expectations of the different pixels.

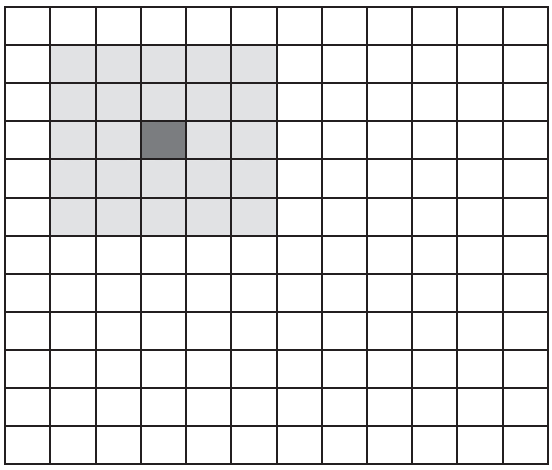


Fig 2.1 : Example of a 5 x 5 neighbourhood

2.3 Problem formulation

Let have K source (degraded) images : $\boldsymbol{X}_1, \cdots, \boldsymbol{X}_K$ of size $N \times M$ describing the same true scene $\boldsymbol{F}$, and $\boldsymbol{Y}$ the fused image.

Let put them in the lexicographic order : $\underline{\mathbf{x}_1}, \ldots, \underline{\mathbf{x}_K}$ and $\underline{\mathbf{y}}$ are now $1 \times NM$ row vectors.

Let call $\underline{\underline{\mathbf{x}}} = \begin{bmatrix} \underline{\mathbf{x}_1} \\ \vdots \\ \vdots \\ \underline{\mathbf{x}_K} \end{bmatrix}$ the $K \times NM$ matrix containing all the source images.

As in the notation section, we define $\underline{\mathbf{x}(n)} = \begin{bmatrix} x_1(n) \\ \vdots \\ \vdots \\ x_K(n) \end{bmatrix}$ as the $K \times 1$ column vector containing of the values of the nth pixel for the different source images.

We are using the same notations for the weights, that is to say that

$$\underline{\underline{\mathbf{w}}} = \begin{bmatrix} \underline{\mathbf{w}_1} \\ \vdots \\ \vdots \\ \underline{\mathbf{w}_K} \end{bmatrix} \text{ and } \underline{\mathbf{w}(n)} = \begin{bmatrix} w_1(n) \\ \vdots \\ \vdots \\ w_K(n) \end{bmatrix}.$$

Each pixel $y(n)$ of the fused image $\underline{\mathbf{y}}$ is a weighted sum of the same pixels $x_1(n), \cdots, x_K(n)$ of the source images. We can write :

$$\boxed{y(n) = \sum_{i=1}^{K} w_i(n) x_i(n) = \underline{\mathbf{w}(n)}^{\mathrm{T}} \underline{\mathbf{x}(n)}} \quad \text{(Eq 2.1)}$$

It seems obvious that all the weights have to be positive and that $\sum_{i=1}^{K} w_i(n) = 1$.

The aim of the algorithm is to determine the matrix $\underline{\underline{\mathbf{w}}}$.

Chapter Three: Presentation of the local image fusion methods

3.1 The Dispersion Minimization Fusion method (DMF)

3.1.1 Notion of dispersion

Let have an image $\boldsymbol{F}$. We define the dispersion factor D_F as :

$$\boxed{D_F = \frac{\mathrm{E}\left\{\widetilde{\boldsymbol{F}}^4\right\}}{\mathrm{E}\left\{\widetilde{\boldsymbol{F}}^2\right\}}} \quad \text{(Eq 3.1)}$$

where $\widetilde{\boldsymbol{F}}$ is the zero mean version of the image $\boldsymbol{F}$.

This is the division of the 4th order central moment and the variance of the image $\boldsymbol{F}$. The higher this parameter is, the worse the quality is. Therefore the fused image should be less dispersed than the source ones. With this definition it is obvious that the parameter D_F is positive.

3.1.2 Minimisation of the cost function

One can notice that in the previous definition, we need to deal with zero mean images. That is why in the rest of the paper we will use $\underline{\widetilde{\mathbf{x}}(n)}$ instead of $\underline{\mathbf{x}(n)}$ (zero mean version of the source images). We will just use the non zero mean version of the source images for the final step, that is to say the reconstruction of the image with the final weights.

We have seen that the dispersion parameter is a way to evaluate the distortion of an image. It seems then natural to describe a fusion method as a process of minimising the dispersion of the fused image. We introduce here a Constant-Modulus cost function that we want to minimize [4] :

$$\boxed{J_{CM} = \mathrm{E}\left\{\left(\underline{\tilde{\mathbf{y}}}^2 - D_F\right)^2\right\}} \quad \text{(Eq 3.2)}$$

where $\underline{\tilde{\mathbf{y}}}$ is the zero mean version of the image $\underline{\mathbf{y}}$

and D_F is the dispersion factor or the original image $\boldsymbol{F}$

The particularity of the cost function is that we do not know the value of D_F. We are then going to update at each iteration the value of D_F.

We know that $\tilde{y}(n) = \underline{\mathbf{w}(n)}^{\mathrm{T}}\,\underline{\tilde{\mathbf{x}}(n)}$ (Eq 2.1) so we can rewrite the cost function :

$$\boxed{J_{CM}\left(\underline{\mathbf{w}(n)}, D_F\right) = \mathrm{E}\left\{\left(\left(\underline{\mathbf{w}(n)}^{\mathrm{T}}\,\underline{\tilde{\mathbf{x}}(n)}\right)^2 - D_F\right)^2\right\}} \quad \text{(Eq 3.3)}$$

We therefore need to minimize a cost function depending on two unknown parameters : $\underline{\mathbf{w}(n)}$ and D_F. To solve this problem we are going to use a gradient descent method, using two learning rates: μ and η. We need then to calculate the gradient of $J_{CM}\left(\underline{\mathbf{w}(n)}, D_F\right)$ relative to $\underline{\mathbf{w}(n)}$ and D_F.

3.1.3 Calculation of the gradients

The proofs for the lemmas can be found in the annexe part.

Calculation of $\dfrac{\partial J_{CM}}{\partial \underline{\mathbf{w}(n)}}$

We know that :

$$J_{CM} = \mathrm{E}\left\{\left(\underline{\tilde{\mathbf{y}}}^2 - D_F\right)^2\right\} = \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^4\right\} - 2D_F\,\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\} + D_F^{\,2}$$

So : $$\frac{\partial J_{CM}}{\partial \underline{\mathbf{w}(n)}} = \frac{\partial \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^4\right\}}{\partial \underline{\mathbf{w}(n)}} - 2D_F\,\frac{\partial \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}{\partial \underline{\mathbf{w}(n)}}$$

And with the Lemma 2 we have that :

$$\boxed{\frac{\partial J_{CM}}{\partial \underline{\mathbf{w}(n)}} = \frac{4}{NM}\left(\tilde{y}(n)^2 - D_F\right)\tilde{y}(n)\underline{\tilde{\mathbf{x}}(n)}} \quad \text{(Eq 3.4)}$$

<u>Calculation of</u> $\frac{\partial J_{CM}}{\partial D_F}$

We know that :

$$J_{CM} = \mathrm{E}\left\{\left(\underline{\tilde{\mathbf{y}}}^2 - D_F\right)^2\right\} = \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^4\right\} - 2D_F\,\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\} + D_F^{\,2}$$

So : $\frac{\partial J_{CM}}{\partial D_F} = -2\,\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\} + 2D_F$

And finally :

$$\boxed{\frac{\partial J_{CM}}{\partial D_F} = 2\left(D_F - \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\right)} \quad \text{(Eq 3.5)}$$

3.1.4 <u>Proposed algorithm</u>

<u>Initialisation</u>

- Set all the weights at the value $\frac{1}{K}$. The first iteration of the fused image will then be simply the mean of the K source images.
- Set the original value of D_F as the mean of the dispersion parameters of the K source images.

<u>Iteration</u>

- Update the values of $\underline{\mathbf{w}(n)}$

$$\underline{\mathbf{w}(n)}' \Leftarrow \underline{\mathbf{w}(n)} - \mu \frac{\partial J_{CM}}{\partial \underline{\mathbf{w}(n)}}$$

- Normalise the values of $\underline{\mathbf{w}(n)}$

$$\underline{\mathbf{w}(n)}' \Leftarrow \text{abs}\left(\frac{\underline{\mathbf{w}(n)}}{\left\| \underline{\mathbf{w}(n)} \right\|} \right)$$

- Update the value of D_F

$$D_F' \Leftarrow D_F - \eta \frac{\partial J_{CM}}{\partial D_F}$$

- Check that D_F is positive

$$D_F' \Leftarrow \text{abs}(D_F)$$

We stop when the results are good. We must not forget once we have found a good matrix $\underline{\underline{\mathbf{w}}}$ to reconstruct the image with the non-zero mean source images.

3.2 The Robust Dispersion Minimization Fusion method (RobustDMF)

The parameters μ and η have a very important role in the convergence and the success of the method. With bad values for these learning rates, the cost function could find a local minimum and not the real minimum of the cost function. That is why before updating the values for $\underline{\underline{\mathbf{w}}}$ and D_F we try some particular values for μ and η and choose the values which actually minimize the cost function. These values are from 10^{-8} to 1 but common value for μ is 10^{-6} and for η is 0.9. This enables us to get better results and to do it faster than with the original method. We shall refer to this method as the Robust DMF method [4].

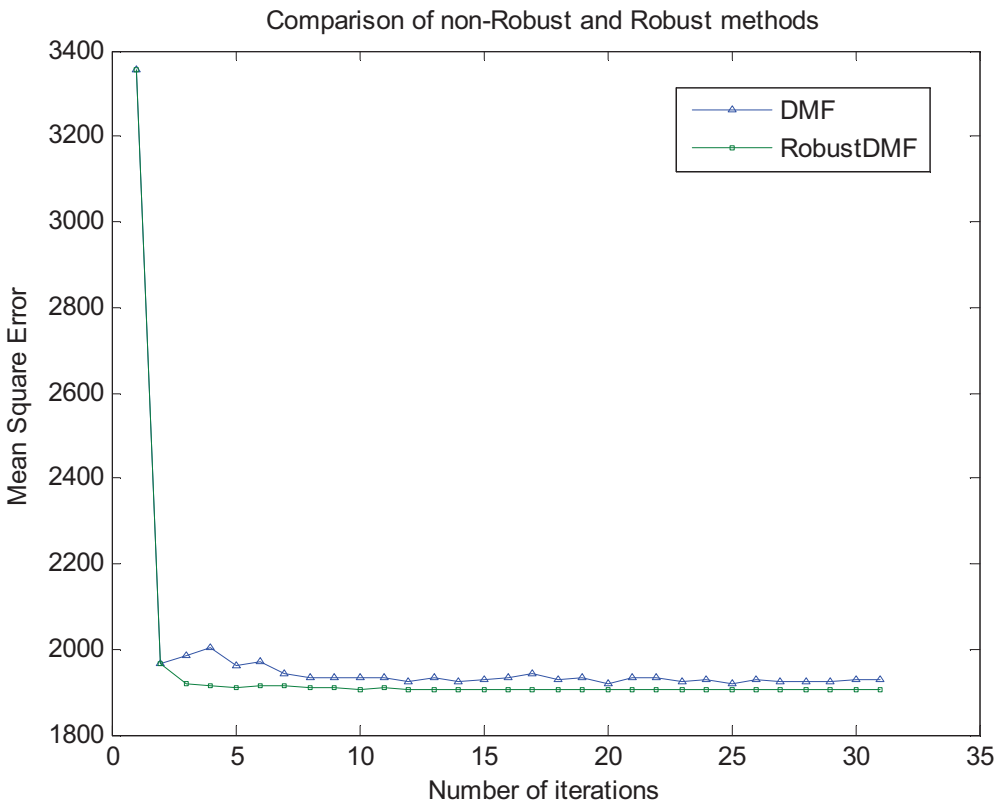


Fig 3.1 : Differences between DMF and RobustDMF methods on one example.

3.3 The Dispersion Minimization Fusion method With Neighbourhood (DMF-WN)

In this method, instead of taking $\frac{\partial J_{CM}}{\partial \underline{\mathbf{w}(n)}} = \frac{4}{NM}\left(\tilde{y}(n)^2 - D_F\right)\tilde{y}(n)\underline{\tilde{\mathbf{x}}(n)}$,

that is to say what we found with the real calculation, we decide to take for the gradient this definition :

$$\frac{\partial J_{CM}}{\partial \underline{\mathbf{w}(n)}} = \underline{\mathbf{E}_{\mathbf{n}}\left\{\frac{4}{NM}\left(\tilde{y}(n)^2 - D_F\right)\tilde{y}(n)\underline{\tilde{\mathbf{x}}(n)}\right\}} \text{ that is to say}$$

$$\boxed{\frac{\partial J_{CM}}{\partial \underline{\mathbf{w}(n)}} = \begin{bmatrix} \mathrm{E}_n\left\{\frac{4}{NM}\left(\tilde{y}(n)^2 - D_F\right)\tilde{y}(n)\tilde{x}_1(n)\right\} \\ \vdots \\ \vdots \\ \mathrm{E}_n\left\{\frac{4}{NM}\left(\tilde{y}(n)^2 - D_F\right)\tilde{y}(n)\tilde{x}_K(n)\right\} \end{bmatrix}} \qquad \text{(Eq 3.6)}$$

(where E_n is the mean of the values of the pixels belonging to the $L \times L$ neighbourhood of the considered pixel) (further explanations in Chapter Two)

The method is actually exactly the same than the DMF but the gradient is evaluated over a neighbourhood instead of just one pixel value. The optimal size of neighbourhood depends on the image and on the distortion.

One could want to apply the "Robust" algorithm here. The tests have proved that it is actually not relevant : it does not increase the performances and lead to a very long computation time. There is therefore no "RobustDMF_WN".

3.4 The Kurtosis Maximization Fusion method (KMF)

3.4.1 Motivation

The degraded versions of the original image are often captured with sensors which add smoothing effects or additional noise [5]. The motivation of using kurtosis maximization comes from two facts :

- A smoothing operator often acts like a low-pass filter so the resultant distribution tends to be flatter (that is to say more Gaussian) than the original one.
- The *Central Limit Theorem* states that the sum of several independent random variables tends towards a Gaussian distribution. Additional noise which can be assumed to be independent from the original scene increases the Gaussianity of the sensor.

The combination of these two facts shows us that the probability distribution of the ground truth image should be less Gaussian than the distorted sensor images. *A fortiori* the fused image since it is supposed to be close to the image scene has also this property of non Gaussianity. That is why the base of the method we are about to describe is the non-Gaussianity and the parameter we are going to calculate to evaluate this is the kurtosis.

3.4.2 Notion of kurtosis

Let have an image $\boldsymbol{F}$. We define the kurtosis K_F as :

$$\boxed{K_F = \frac{\mathrm{E}\left\{\widetilde{\boldsymbol{F}}^4\right\}}{\mathrm{E}^2\left\{\widetilde{\boldsymbol{F}}^2\right\}} - 3} \qquad \text{(Eq 3.7)}$$

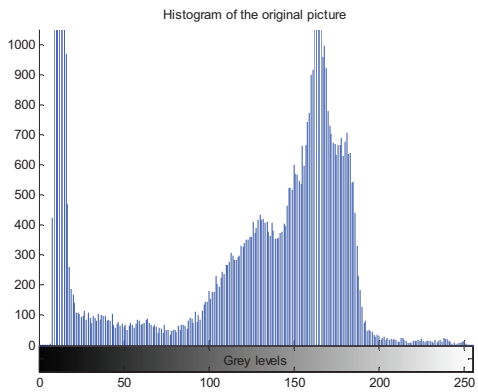
where $\widetilde{\boldsymbol{F}}$ is the zero mean version of the image $\boldsymbol{F}$.

This is the division of the 4th order central moment and the square of the variance of the image $\boldsymbol{F}$. The "minus 3" at the end of this formula is often explained as a correction to make the kurtosis of the normal distribution equal to zero. Therefore the

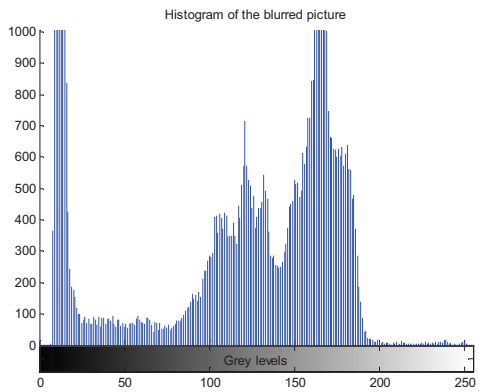
higher the absolute value of the kurtosis is, the “less Gaussian” the distribution of values of pixels is.

Original picture

Blurred picture



Kurtosis = -0.9085



Kurtosis = -0.6850

Fig 3.2 : Histograms and kurtosis of two pictures

3.4.3 <u>Maximization of the cost function</u>

It seems now natural to choose for the cost function the absolute value of the kurtosis:

$$J_K = |K_F| = \left| \frac{E\{\tilde{\underline{y}}^4\}}{E^2\{\tilde{\underline{y}}^2\}} - 3 \right| \qquad \text{(Eq 3.8)}$$

where $\underline{\tilde{\mathbf{y}}}$ is the zero mean version of the image $\underline{\mathbf{y}}$

Considering Eq 2.1 we can rewrite the cost function this way :

$$\boxed{J_K\left(\underline{\mathbf{w}(n)}\right)=\left|\frac{\mathrm{E}\left\{\left(\underline{\mathbf{w}(n)}\,\underline{\tilde{\mathbf{x}}(n)}\right)^4\right\}}{\mathrm{E}^2\left\{\left(\underline{\mathbf{w}(n)}\,\underline{\tilde{\mathbf{x}}(n)}\right)^2\right\}}-3\right|} \qquad \text{(Eq 3.9)}$$

We therefore need to maximize a cost function depending on one unknown parameter: $\underline{\mathbf{w}(n)}$. To solve this problem we are going to use a gradient descent method, using one learning rate: λ. We need then to calculate the gradient of $J_K\left(\underline{\mathbf{w}(n)}\right)$ relative to $\underline{\mathbf{w}(n)}$.

3.4.4 Calculation of the gradient

The proofs for the lemmas can be found in the annexe part.

$$J_K=\frac{\left|\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^4\right\}-3\,\mathrm{E}^2\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\right|}{\mathrm{E}^2\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}=\frac{\left|\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right|}{\mathrm{E}^2\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}$$

$$\frac{\partial J_K}{\partial\underline{\mathbf{w}}(n)}=\frac{1}{\mathrm{E}^4\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}\times\left[\frac{\partial\left|\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right|}{\partial\underline{\mathbf{w}}(n)}\mathrm{E}^2\left\{\underline{\tilde{\mathbf{y}}}^2\right\}-\frac{\partial\,\mathrm{E}^2\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}{\partial\underline{\mathbf{w}}(n)}\left|\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right|\right]$$

But :

- $$\frac{\partial\,\mathrm{E}^2\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}{\partial\underline{\mathbf{w}}(n)}=2\,\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\frac{\partial\,\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}{\partial\underline{\mathbf{w}}(n)}=\frac{4}{NM}\,\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\tilde{y}(n)\underline{\tilde{\mathbf{x}}}(n) \quad \text{(Lemma 2)}$$
- $$\frac{\partial\left|\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right|}{\partial\underline{\mathbf{w}}(n)}=\mathrm{sgn}\left(\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right)\left[\frac{\partial\,\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^4\right\}}{\partial\underline{\mathbf{w}}(n)}-3\frac{\partial\,\mathrm{E}^2\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}{\partial\underline{\mathbf{w}}(n)}\right]$$

  $$\frac{\partial\left|\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right|}{\partial\underline{\mathbf{w}}(n)}=\mathrm{sgn}\left(\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right)\left[\frac{4}{NM}\,\tilde{y}^3(n)\underline{\tilde{\mathbf{x}}}(n)-\frac{12}{NM}\,\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\tilde{y}(n)\underline{\tilde{\mathbf{x}}}(n)\right] \quad \text{(Lemma 2)}$$

so :

$$\frac{\partial J_K}{\partial\underline{\mathbf{w}}(n)}=\frac{4}{NM}\times\frac{\mathrm{sgn}\left(\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right)}{\mathrm{E}^4\left\{\underline{\tilde{\mathbf{y}}}^2\right\}}\left[\mathrm{E}^2\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\tilde{y}^3(n)\underline{\tilde{\mathbf{x}}}(n)-3\,\mathrm{E}^3\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\tilde{y}(n)\underline{\tilde{\mathbf{x}}}(n)-\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\tilde{y}(n)\underline{\tilde{\mathbf{x}}}(n)\,\mathrm{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right]$$

$$\frac{\partial J_K}{\partial \underline{\mathbf{w}}(n)} = \frac{4}{NM} \times \frac{\operatorname{sgn}\left(\operatorname{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right)}{\mathrm{E}^3\left\{\underline{\tilde{\mathbf{y}}}^2\right\}} \left[\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\tilde{y}^3(n)\underline{\tilde{\mathbf{x}}}(n) - \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^4\right\}\tilde{y}(n)\underline{\tilde{\mathbf{x}}}(n)\right]$$

$$\boxed{\frac{\partial J_K}{\partial \underline{\mathbf{w}}(n)} = \frac{4}{NM} \times \frac{\operatorname{sgn}\left(\operatorname{cum}_4\left\{\underline{\tilde{\mathbf{y}}}\right\}\right)}{\mathrm{E}^3\left\{\underline{\tilde{\mathbf{y}}}^2\right\}} \left[\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^2\right\}\tilde{y}^2(n)\underline{\tilde{\mathbf{x}}}(n) - \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^4\right\}\right]\tilde{y}(n)\underline{\tilde{\mathbf{x}}}(n)} \qquad \text{(Eq 3.10)}$$

3.4.5 Proposed algorithm

Initialisation

- Set all the weights at the value $\frac{1}{K}$. The first iteration of the fused image will then be simply the mean of the K source images.

Iteration

- Update the values of $\underline{\mathbf{w}(n)}$

$$\underline{\mathbf{w}(n)}' \Leftarrow \underline{\mathbf{w}(n)} - \lambda \frac{\partial J_K}{\partial \underline{\mathbf{w}(n)}}$$

- Normalise the values of $\underline{\mathbf{w}(n)}$

$$\underline{\mathbf{w}(n)}' \Leftarrow \operatorname{abs}\left(\frac{\underline{\mathbf{w}(n)}}{\left\|\underline{\mathbf{w}(n)}\right\|}\right)$$

We stop when the results are good. We must not forget once we have found a good matrix $\underline{\underline{\mathbf{w}}}$ to reconstruct the image with the non-zero mean source images.

3.5 The Robust Kurtosis Maximization Fusion method (RobustKMF)

As for the DMF method we can also use here an optimized learning rate λ. Before updating the values for $\underline{\underline{\mathbf{w}}}$ we try some particular values for λ and choose the value which actually maximizes the cost function.

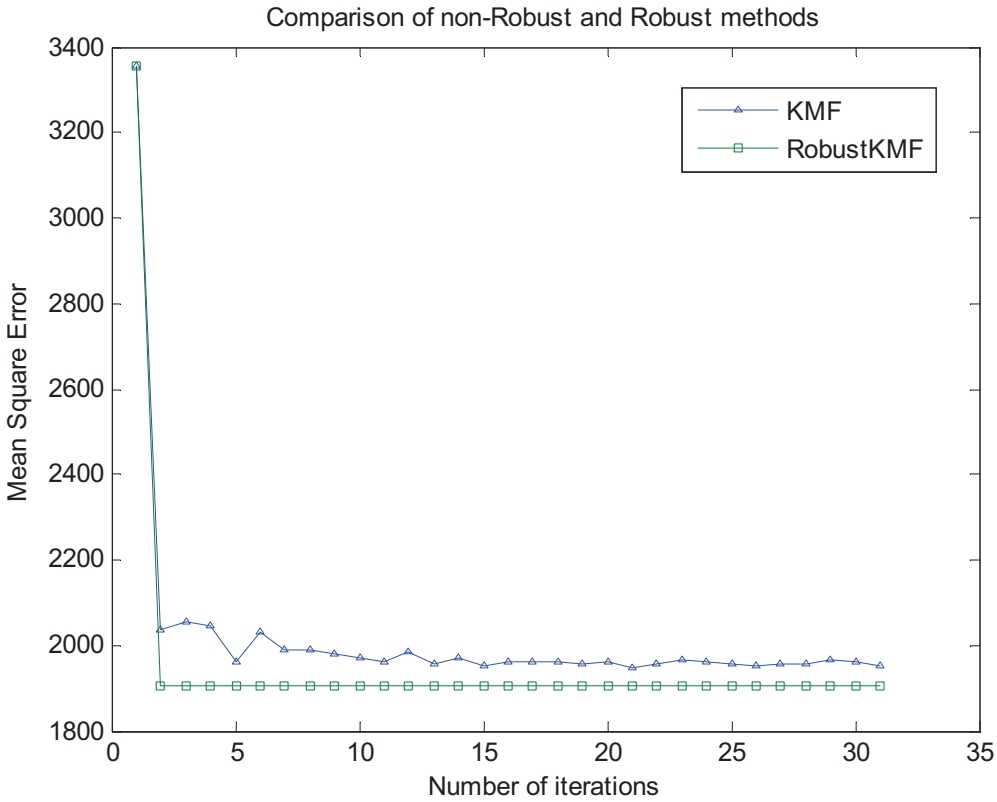


Fig 3.3 : Differences between KMF and RobustKMF methods on one example.

Chapter Four: Evaluation of the performances

4.1 Different metrics for the evaluation of the performances

In this section we shall present different quality metrics for image fusion which will enable us to evaluate the performances of the different methods.

4.1.1 Image Quality Index

This section is inspired by the reference [6]. One can refer to this paper for more details.

This image quality index is easy to calculate and applicable to various image processing applications. It measures the similarity between the true scene ***F*** and fused image ***Y*** One can notice that with this metric we need to know the original image that is to say that it is only applicable to non-blind methods.

The definition of *image quality index* is :

$$\boxed{Q_0 = \frac{4\sigma_{FY}\, m_F\, m_Y}{\left(m_F^{\,2} + m_Y^{\,2}\right)\left(\sigma_F^{\,2} + \sigma_Y^{\,2}\right)}} \qquad \text{(Eq 4.1)}$$

where : $m_F = \frac{1}{NM}\sum_{i=1}^{NM} f(i)$; $\sigma_F^{\,2} = \frac{1}{NM-1}\sum_{i=1}^{NM}\left(f(i) - m_F\right)^2$ and

$$\sigma_{FY} = \frac{1}{NM-1}\sum_{i=1}^{NM}\left(f(i) - m_F\right)\left(y(i) - m_Y\right)$$

The dynamic range of Q_0 is [-1;1]. The best value 1 is achieved if and only if the fused image is exactly equal to the original one. The closer to 1 the value of Q_0 is, the better the fusion performances are.

4.1.2 Mean Gradient

This section is inspired by the reference [7]. One can refer to this paper for more details.

The mean gradient is a metric which applies to the fused image. In theory, a larger MG value for the image ***Y*** is preferred as it reflects a higher contrast within the detailed variation of a pattern in the fused image and more clarity.

The definition of *mean gradient* is then :

$$\boxed{MG = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\sqrt{\frac{\Delta y_i^2 + \Delta y_j^2}{2}}} \qquad \text{(Eq 4.2)}$$

where $\Delta y_i = y(i+1,j) - y(i,j)$ and $\Delta y_j = y(i,j+1) - y(i,j)$

4.1.3 <u>Petrovic Index</u>

This metric is calculated with the degraded input images and the fused image [8].

Let have a degraded image $\boldsymbol{X_k}$ and the fused image $\boldsymbol{Y}$.

First we need to extract the information about the edge strength $g_{X_k}(n)$ and the orientation $\alpha_{X_k}(n)$ for each pixel $X_k(n)$. We are going to use the Sobel edge operator. Let call $s^x_{X_k}(n)$ and $s^y_{X_k}(n)$ the outputs of the horizontal and vertical Sobel filter. Now $g_{X_k}(n)$ will be a « magnitude » and $\alpha_{X_k}(n)$ the « phase ». So :

$$g_{X_k}(n) = \sqrt{s^x_{X_k}(n)^2 + s^y_{X_k}(n)^2}$$

$$\alpha_{X_k}(n) = \tan^{-1}\left(\frac{s^y_{X_k}(n)}{s^x_{X_k}(n)}\right)$$

Now the relative strength and orientation values of $G_{X_k}(n)$ and $A_{X_k}(n)$ of $\boldsymbol{X_k}$ with respect to $\boldsymbol{Y}$ are formed as :

$$G^{X_k Y}(n) = \begin{cases} \dfrac{g_Y(n)}{g_{X_k}(n)} & \textit{if} \quad g_Y(n) > g_{X_k}(n) \\ \dfrac{g_{X_k}(n)}{g_Y(n)} & \textit{otherwise} \end{cases}$$

$$A^{X_k Y}(n) = 1 - \frac{\left|\alpha_{X_k}(n) - \alpha_Y(n)\right|}{\pi / 2}$$

These are used to derive the edge strength and orientation preservation values :

$$S_g^{X_k Y}(n) = \frac{\Gamma_g}{1 + e^{\kappa_g \left(G^{X_k Y}(n) - \sigma_g\right)}}$$

$$S_\alpha^{X_k Y}(n) = \frac{\Gamma_\alpha}{1 + e^{\kappa_\alpha \left(A^{X_k Y}(n) - \sigma_\alpha\right)}}$$

The parameters $\Gamma_g, \Gamma_\alpha, \kappa_g, \kappa_\alpha, \sigma_g, \sigma_\alpha$ determine the exact shape of the sigmoid functions used to form the edge strength and orientation preservation values.

Now the edge information preservation values are :

$$S^{X_k Y}(n) = S_g^{X_k Y}(n) S_\alpha^{X_k Y}(n)$$

Considering the K input images we define the *Petrovic metric* :

$$\boxed{S = \frac{\sum_{k=1}^{K} \sum_{n=1}^{NM} S^{X_k Y}(n) g_{X_k}(n)^L}{\sum_{k=1}^{K} \sum_{n=1}^{NM} g_{X_k}(n)^L}} \qquad \text{(Eq 4.3)}$$

with L being a constant.

One can notice that : $0 \leq S \leq 1$ and that the closer to 1 the value of S is, the better the fusion performances are.

4.1.4 <u>Piella Index Metrics</u>

These metrics are just variants [9] of the index presented in 4.1.1 (Q_0 Image Quality Index). In this presentation we we assume that we have two input images $\boldsymbol{X_1}$, $\boldsymbol{X_2}$ and the fused image $\boldsymbol{Y}$.

Since image signals are generally non-stationary, it is more appropriate to measure the image quality index Q_0 over local regions and then combine the different results into a single measure.

We are going to use a sliding window approach : starting from the top-left corner of two images, a sliding window of fixed size $L \times L$ moves pixel by pixel over the entire image until the bottom-right corner is reached.

For each window ω and each set of two images ***A*** and ***B*** we define the local quality image $Q_0(A, B|\omega)$ as the image quality index which is computed with the pixels of ***A*** and ***B*** lying in the sliding window ω.

Let call $\sigma_{X_1}(\omega)$ and $\sigma_{X_2}(\omega)$ the variances of images $\boldsymbol{X_1}$ and $\boldsymbol{X_2}$ within the window ω. We compute a local weight $\lambda(\omega)$ between 0 and 1 indicating the relative importance of image $\boldsymbol{X_1}$ compared to image $\boldsymbol{X_2}$: the larger $\lambda(\omega)$ the more weight is given to image $\boldsymbol{X_1}$. We decide to take :

$$\lambda(\omega) = \frac{\sigma_{X_1}(\omega)}{\sigma_{X_1}(\omega) + \sigma_{X_2}(\omega)}$$

Now we define the *fusion quality index* $Q(X_1, X_2, Y)$ as :

$$\boxed{Q(X_1, X_2, Y) = \frac{1}{|W|} \sum_{\omega \in W} \left(\lambda(\omega) Q_0(X_1, Y|\omega) + (1 - \lambda(\omega)) Q_0(X_2, Y|\omega) \right)} \quad \text{(Eq 4.4)}$$

At this point, our model has produced a quality index which gives an indication of how much of the salient information contained inn each of the input images has been transferred into the fused image without introducing distortions. However, the different quality measures obtained within each window have been treated equally. This is in contrast with the human visual system which is known to give higher importance to visually salient regions in an image. We now define another variant of the fusion quality index by giving more weight to those windows where the saliency of the input images is higher.

The overall saliency of a window is defined as $C(\omega) = \max(\sigma_{X_1}(\omega), \sigma_{X_2}(\omega))$. The *weighted fusion quality index* $Q_w(X_1, X_2, Y)$ is then defined as :

$$\boxed{Q_w(X_1, X_2, Y) = \frac{1}{|W|} \sum_{\omega \in W} c(\omega) \left(\lambda(\omega) Q_0(X_1, Y|\omega) + (1 - \lambda(\omega)) Q_0(X_2, Y|\omega) \right)} \quad \text{(Eq 4.5)}$$

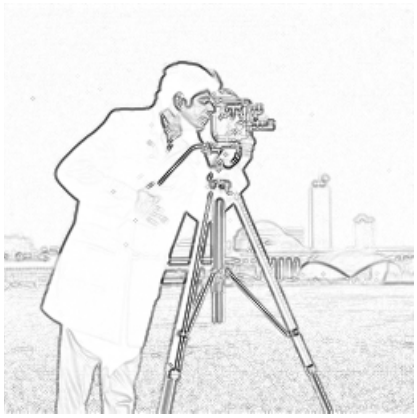
where $c(\omega) = \dfrac{C(\omega)}{\sum_{\omega' \in W} C(\omega')}$

We define one final modification of the fusion quality index that takes into account the importance of edge information. We can evaluate Q_w using 'edge images' (for example the Euclidian norm of the horizontal and vertical gradient images). Let call $\boldsymbol{X_1}', \boldsymbol{X_2}'$ and $\boldsymbol{Y}'$ the 'edge-images' associated to $\boldsymbol{X_1}, \boldsymbol{X_2}$ and $\boldsymbol{Y}$.

We define the *edge-dependent fusion quality* as :

$$\boxed{Q_e(X_1, X_2, Y) = Q_w(X_1, X_2, Y) \times Q_w(X_1', X_2', Y')} \qquad \text{(Eq 4.6)}$$

Original image



'Edge' image

Fig 4.1 : Example of "edge" image

4.2 Results

4.2.1 Some remarks

- In order to evaluate our methods we will compare them to well known methods on a various set of images.
  - ***ICA*** is Independent Component Analysis. This is a transform domain method which is very popular. Two options will be used in the simulation : the weighted and the regional ones. More details about these methods can be found in [3]
  - ***DT_WT*** is Dual-Tree Wavelet Transform. This is a transform domain which uses wavelets. This is for now one of the most powerful method. The option we chose is the max-abs one. One can find further explanations in [2].
  - ***EEF*** is Error Estimation Fusion. This is a spatial domain method which has been developed very recently and uses the robust error estimation theory. Just like our methods this is an iterative method. A developed analysis can be found in [10] and [11].

- In order to give numerical results we will use the following metrics:
  - $\mathbf{Q_0}$ stands for image quality index (4.1.1) [6]
  - ***MG*** stands for mean gradient (4.1.2) [7]
  - ***S*** stands for Petrovic index (4.1.3) [8]
  - **Q** stands for the original Piella index (4.1.4) [9]
  - $\mathbf{Q_w}$ stands for the Piella's weighted fusion quality index (4.1.4) [9]
  - $\mathbf{Q_e}$ stands for the Piella's edge-dependent fusion quality index (4.1.4) [9]

- When we have the choice of Robust and non-Robust method we will always choose Robust methods since they are quicker and converge better. In these cases 15 iterations are often enough.

- In the DMF_WN we will do 10 iterations since the computation time is often very long.

So for every set of images we will try :

RobustKMF, 15 iterations

RobustDMF, 15 iterations

DMF_WN, small size of neighbourhood

DMF_WN, big size of neighbourhood

ICA, regional option (non-spatial transform)

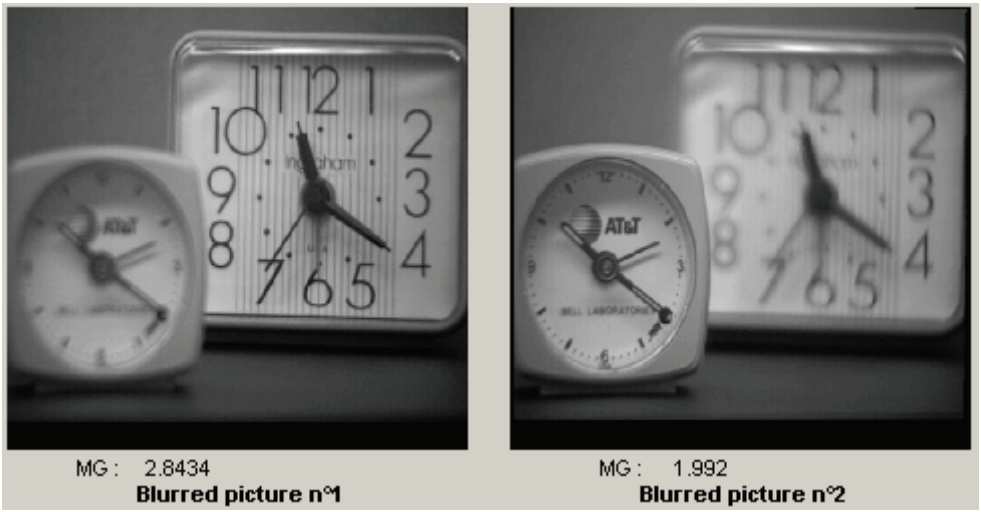
ICA, weighted option (non-spatial transform)

DT_WT, max-abs option (non-spatial transform)

EEF, 15 iterations

4.2.2 Case one : Multi-focus images, small distortion

Example 1 : Clocks



The results are not very good, for any of our method (DMF and KMF), compared for example to the DT_WT method. The edges of the big clock still seem blurred and the number '10' is very blurred.

Maybe only the DMF_WN with a big neighbourhood (9x9) can do something acceptable. Considering that the input images are big (512x512) this is the biggest size of neighbourhood we can take without being out of memory.

	RobustKMF	RobustDMF	DMF_WN 3x3	DMF_WN 9x9	ICA regional	ICA weighted	DT_WT	EEF
Q	0.8259	0.8258	0.8298	**0.8408**	0.7932	0.7742	0.7387	0.8404
Q_w	0.8554	0.8553	0.8492	0.8835	0.8858	0.8872	**0.9120**	0.8761
Q_e	0.5839	0.5843	0.5846	0.6676	0.7820	0.7921	**0.8092**	0.6552
S	0.58606	0.58627	0.59304	0.62443	0.70464	**0.71568**	0.67478	0.6356
MG	2.328	2.3236	2.3039	2.2832	3.1386	3.3124	**3.4056**	2.2302

RobustKMF

RobustDMF

DMF_WN 3x3

DMF_WN 9x9

ICA regional

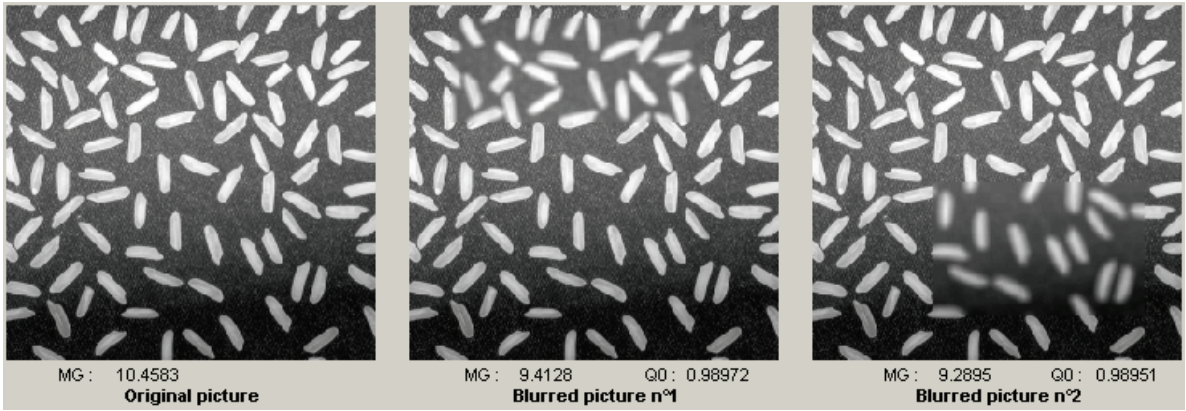
ICA weighted

DT_WT

EEF

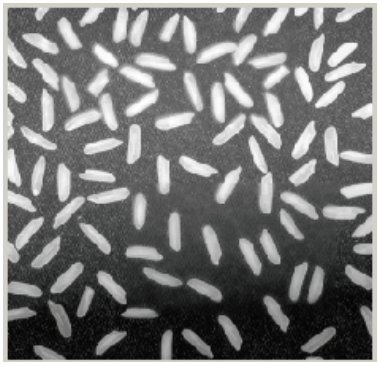
Fig 4.2 : The “Clocks’ example

Example 2 : Rice

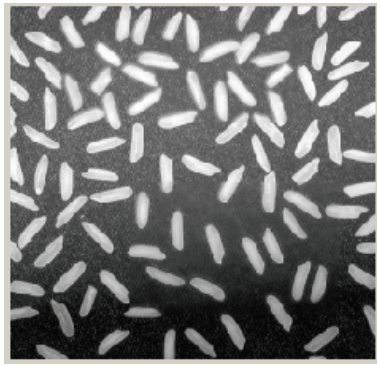


We applied just a little blur on the 'rice' image. While the DT_WT or the weighted ICA work very well, our methods aren't visually very efficient. We can still see that the image has been blurred and the result is not very detailed. Nevertheless the metrics are here very good for our 4 methods compared for example to the ICA ones. This shows us that even if the metrics are supposed to give us information they sometimes fail doing that.

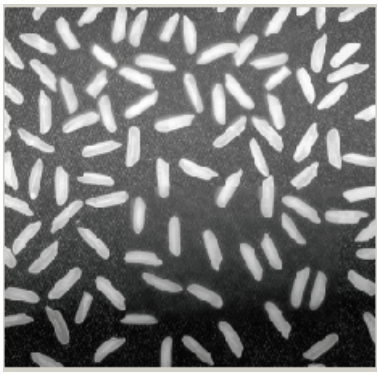
	RobustKMF	RobustDMF	DMF_WN 3x3	DMF_WN 15x15	ICA regional	ICA weighted	DT_WT	EEF
Q_0	0.995	0.99491	0.99616	0.99562	0.91257	0.91198	**0.99901**	0.99643
Q	0.9440	0.9433	0.9522	**0.9524**	0.8223	0.8236	0.9520	0.9498
Q_w	0.9757	0.9754	0.9786	**0.9792**	0.8884	0.8785	0.9734	**0.9792**
Q_e	0.9110	0.9102	0.9246	0.9198	0.7463	0.7277	**0.9338**	0.9262
S	0.8768	0.87583	0.88626	**0.88988**	0.75927	0.74119	0.86172	0.88272
MG	9.294	9.3048	9.278	9.1277	11.6055	**12.3522**	10.5312	9.1099



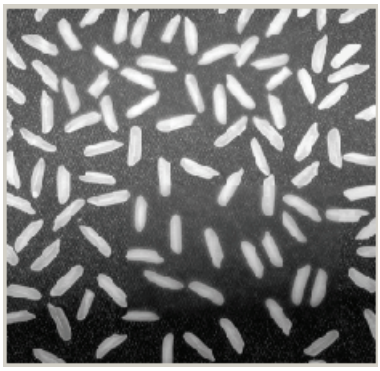
RobustKMF



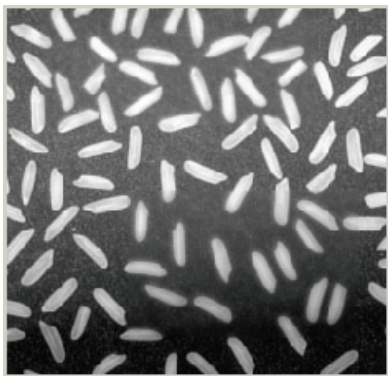
RobustDMF



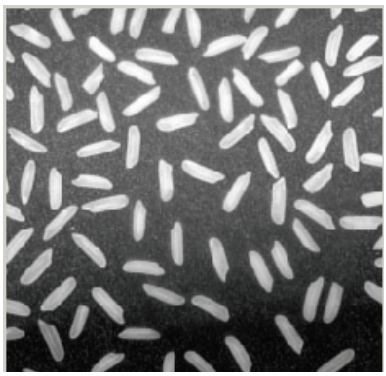
DMF_WN 3x3



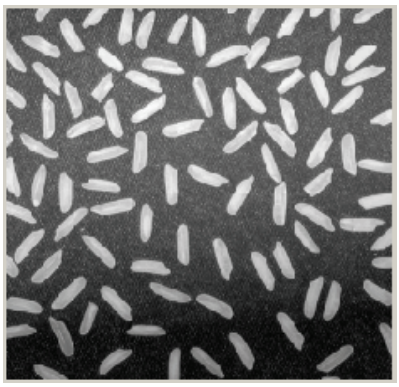
DMF_WN 15x15



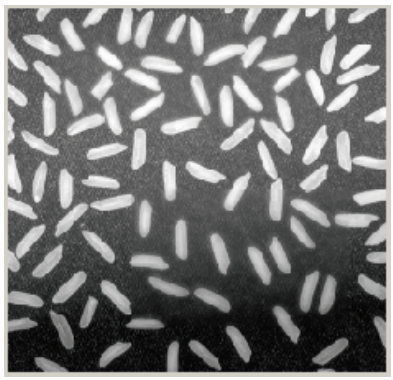
ICA regional



ICA weighted



DT_WT



EEF

Fig 4.3 : The 'Rice' Example

4.2.3 Case two : Multi-focus images, severe distortion

Example 1 : Cameraman

We applied here a very strong distortion on the 'cameraman' image. Our methods are now really better. The RobustDMF and RobustKMF are now much better than the spatial-method EEF.

With big sizes of neighbourhood here we get very good results in the method DMF_WN, even better than the DT_WT.

The methods we have been discussing are actually very relevant for these kinds of distortion. The DT_WT tends to create some discontinuities in the image while the RobustDMF or RobustKMF give really good visual results. Out methods are here the only ones where the original frames of distortion are not visible.

	RobustKMF	RobustDMF	DMF_WN 3x3	DMF_WN 15x15	ICA regional	ICA weighted	DT_WT	EEF
Q_0	0.99265	0.99261	0.99334	**0.99605**	0.98257	0.98397	0.99455	0.98919
Q	0.8948	0.8948	0.8963	**0.9062**	0.7991	0.8357	0.8729	0.9040
Q_w	0.9509	0.9506	0.9542	0.9672	0.9291	0.9366	**0.9681**	0.9491
Q_e	0.8901	0.8892	0.9031	0.9302	0.8480	0.8651	**0.9487**	0.9022
S	0.87097	0.87019	0.8734	**0.89113**	0.81835	0.84979	0.88752	0.86997
MG	8.8364	8.8375	9.0073	9.0431	7.7221	8.16	**10.1157**	8.8082

RobustKMF

RobustDMF

DMF_WN 3x3

DMF_WN 15x15

ICA regional

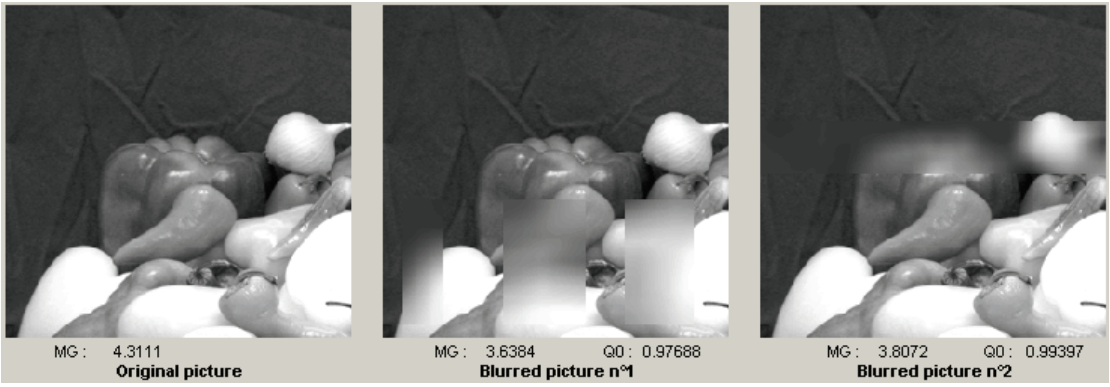
ICA weighted

DT_WT

EEF

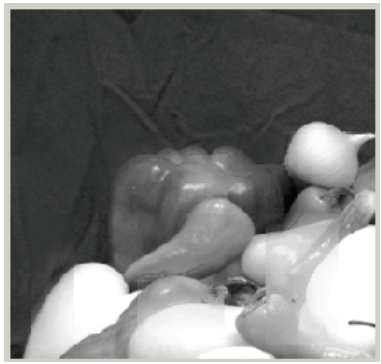
Fig 4.4 : The 'Cameraman' example

Example 2 : Peppers

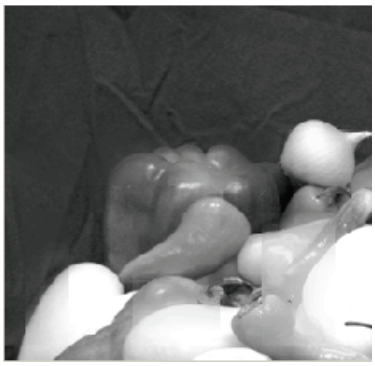


Once again a very strong blur is applied to the 'peppers' image. Now there are several frames where the distortion is applied. In methods such as DT_WT or ICA the frames are still visible but in our methods it is more difficult to see where the original distortion was. That is why we can claim that in this example our methods give better results.

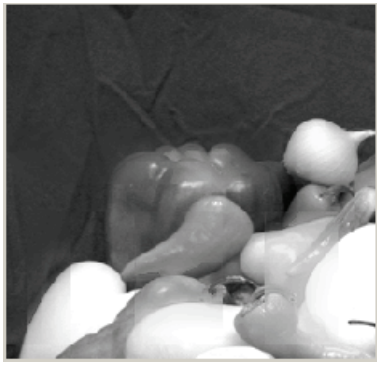
	RobustKMF	RobustDMF	DMF_WN 3x3	DMF_WN 15x15	ICA regional	ICA weighted	DT_WT	EEF
Q_0	0.99421	0.99435	0.99446	**0.99551**	0.98903	0.98947	0.99497	0.9941
Q	0.8964	0.8970	0.8971	0.9100	0.8564	0.8708	0.8549	**0.9140**
Q_w	0.8969	0.8981	0.8998	0.9210	0.9360	0.9412	**0.9462**	0.9292
Q_e	0.7495	0.7538	0.7651	0.8063	0.8626	0.8774	**0.9009**	0.8402
S	0.80689	0.80947	0.81066	0.82009	0.81357	**0.83373**	0.82661	0.82606
MG	3.7699	3.7244	3.7378	3.6913	3.7309	3.931	**4.5186**	3.7229



RobustKMF

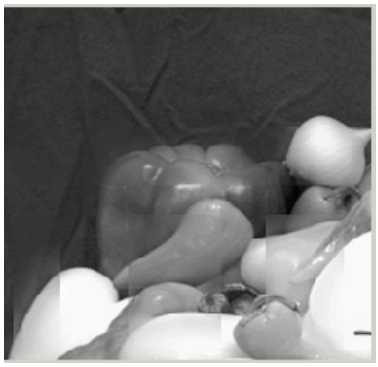


RobustDMF

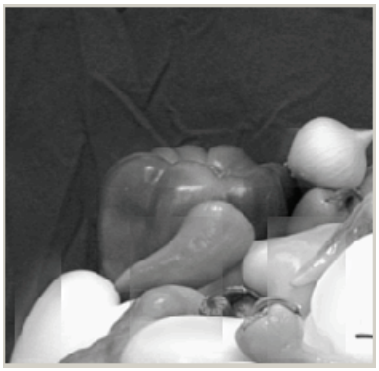


DMF_WN 3x3

DMF_WN 15x15

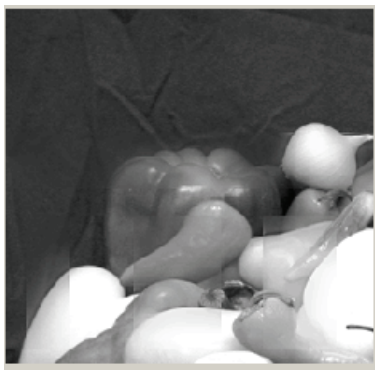


ICA regional



ICA weighted

DT_WT

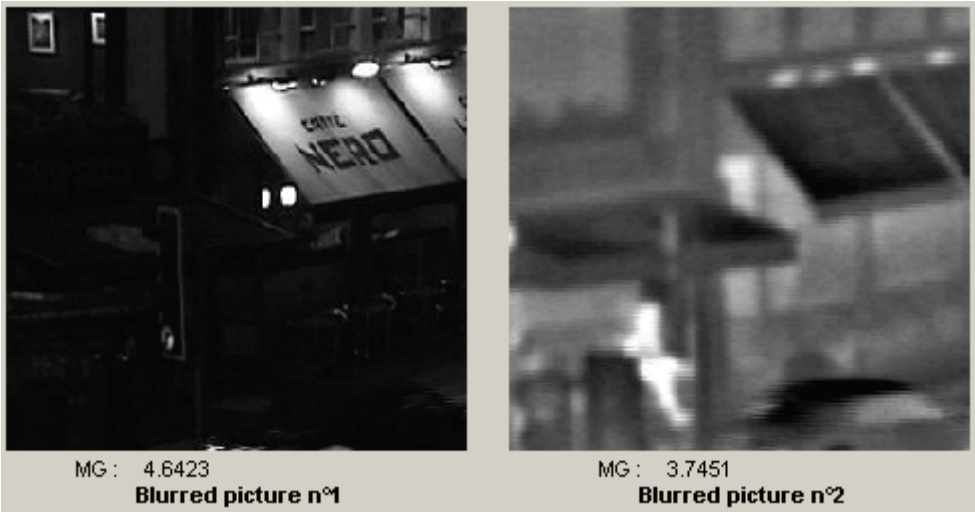


EEF

Fig 4.5 : The 'Peppers' example

4.2.4 Case three : Multi-sensor images

Example 1 : Infra Red / Dark photo



One can notice that here the RobustDMF is the most detailed one, even if some "pepper and salt" noise seems to appear. The RobustKMF is nevertheless the clearest. Depending on what one expects from the fusion these two methods give very good and complementary results. But the DMF_WN which gave us good results previously fails here.

	RobustKMF	RobustDMF	DMF_WN 3x3	DMF_WN 15x15	ICA regional	ICA weighted	DT_WT	EEF
Q	0.5537	0.3650	0.3761	0.4631	0.5393	0.5649	**0.6809**	0.5718
Q_w	0.6673	0.7055	0.7204	0.7921	0.7258	0.7336	**0.8402**	0.7680
Q_e	0.4184	0.4824	0.5375	0.6167	0.5526	0.5666	**0.7796**	0.6459
S	0.39238	0.41474	0.44032	0.49446	0.41524	0.45688	**0.60084**	0.48381
MG	3.6275	**7.6672**	7.2204	5.9965	3.4941	3.7251	6.7304	4.4191

RobustKMF

RobustDMF

DMF_WN 3x3

DMF_WN 15x15

ICA regional

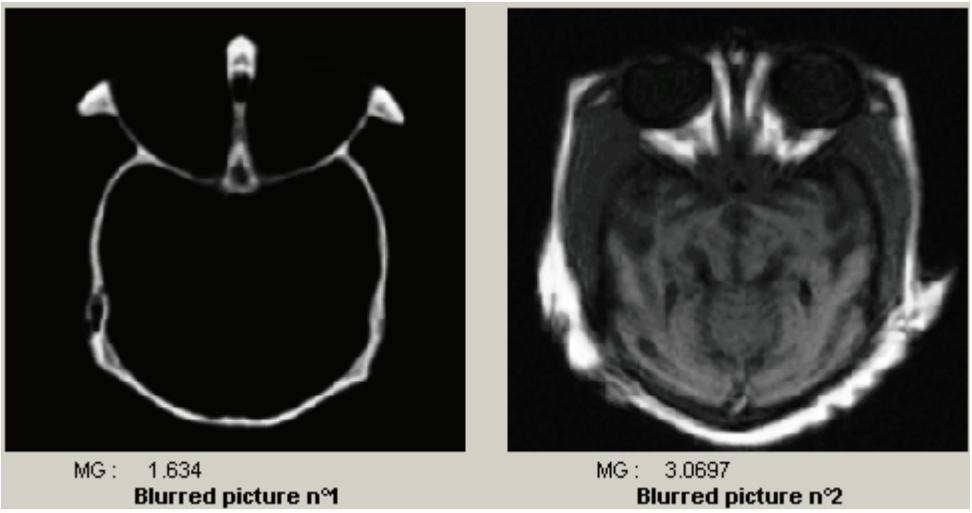
ICA weighted

DT_WT

EEF

Fig 4.6 : The 'Coffee Shop' example

Example 2 : Medical photos



The RobustKMF gives here good visual results even if the metrics are not very good. One can notice that the DMF_WN with 15x15 neighbourhoods is maybe the best one but one can see that the 15x15 blocks are visible which can be annoying.

	RobustKMF	RobustDMF	DMF_WN 3x3	DMF_WN 15x15	ICA regional	ICA weighted	DT_WT	EEF
Q	0.6451	0.7114	**0.8138**	0.8085	0.6656	0.6403	0.7939	0.6499
Q_w	0.6417	0.7997	0.8231	0.7394	0.7396	0.7373	**0.8301**	0.7042
Q_e	0.3618	0.5632	0.6271	0.5086	0.5689	0.5622	**0.6605**	0.4818
S	0.51682	0.63921	0.7067	0.6442	**0.74725**	0.71922	0.70103	0.55065
MG	2.5105	4.1956	3.8514	3.5105	5.1035	**5.8861**	4.2258	2.4108

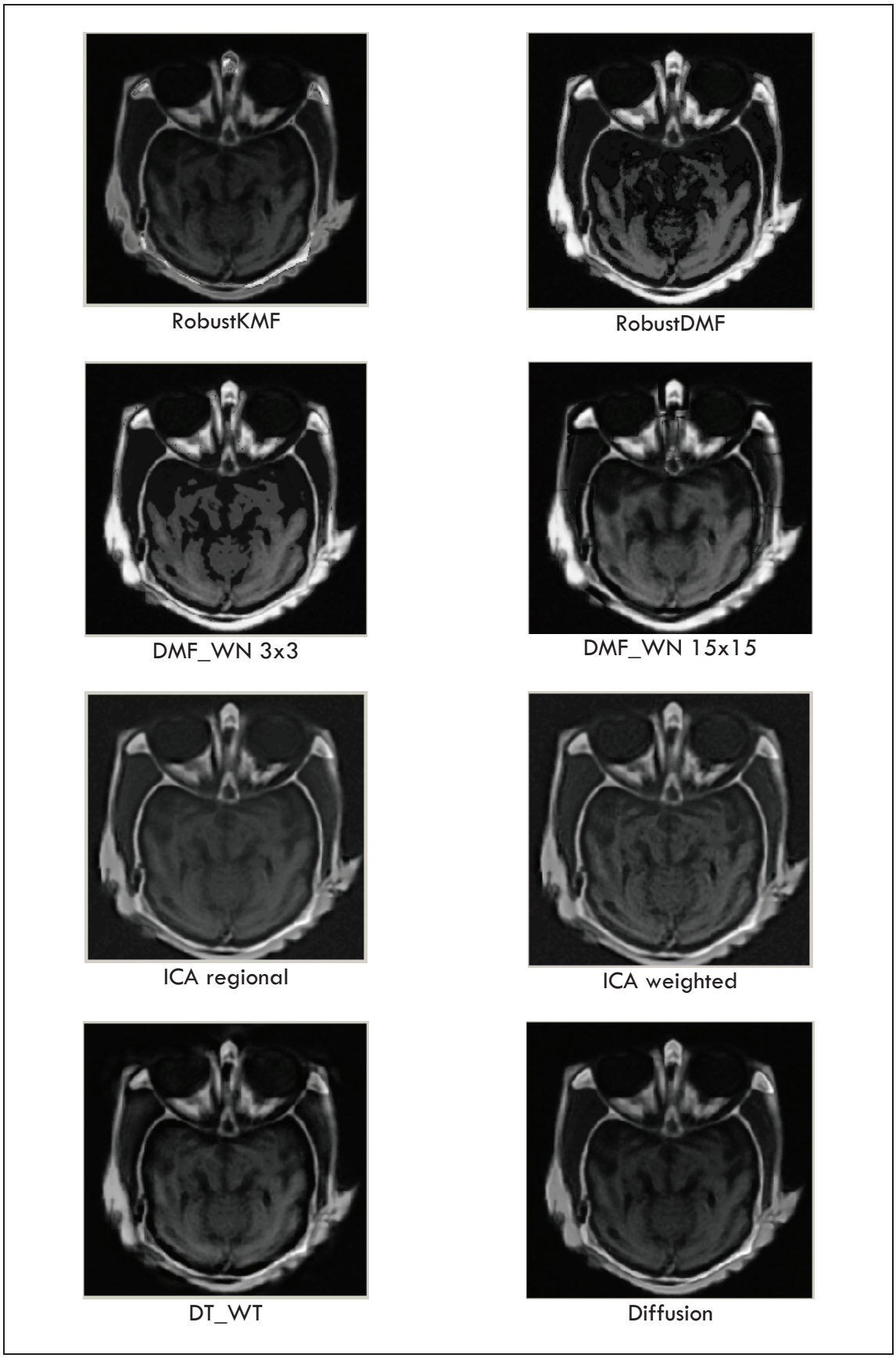


Fig 4.7 : The 'Medical' example

4.3 Assessment

In the presented results we can really highlight that our methods give very good results in almost every situation. The only problem we shall meet is that they are maybe not really relevant for multi-focus images with little distortion.

The evaluation of the results is of course tricky because it depends on the perception of the viewer but in most of the cases our methods are really acceptable.

Even if the DT_WT seem to be the best method we shall not forget that it is a transform domain method and that ours are spatial domain ones. In some applications one can prefer to work in the spatial domain and in this case our methods are clearly better.

The introduction of the Robust methods does not really enhance the results but it enables us to have good results with less iterations. The Neighbourhood method gives us better results but the optimal size of the neighbourhood is still something depending on the picture and the expected results.

Chapter Five: The image fusion simulation software

Since we had lots of methods with lots of parameters to implement, test and compare we decided to create a simulation tool (Matlab GUI).

The aim of this software is to have a clear and nice interface that enables us to compare easily the different methods of image fusion and to choose the parameters (such as Robust or not Robust, number of iterations etc…) directly.

We implemented the DMF and KMF methods and we just transformed existing codes for the ICA, DT_WT and EEF methods in order everything to fit in the software. One can see all the results on the same window which makes the comparison easier.

We added some little functions which add blur to specific regions of good quality images. We can therefore build our own distorted images : this is actually very useful since it is the only way to know the ground truth image. That is to say that we can choose for inputs either two common used images (like the Clocks) or two "home-made" blurred images.

The software also calculates all the needed metrics for the different methods.

This tool was actually really useful throughout the project since we had lots of sets of images to test and that it is quite impossible to evaluate the methods without comparing them to the commonly used ones.

Nevertheless it took a long time to create it and to make it work especially with all the methods that we did not implement. But this time was not wasted since it made the evaluation of the methods really easier and more efficient.

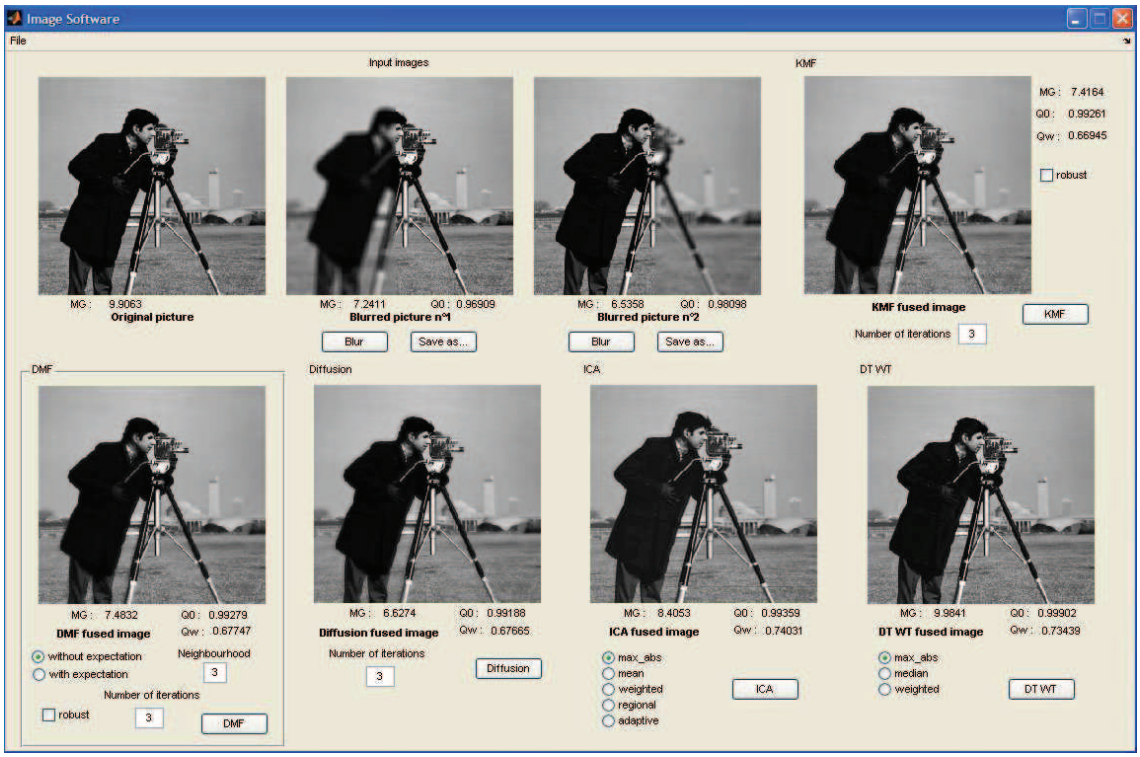


Fig 5.1 : Interface of the software

Conclusion

Throughout this thesis we have described some new spatial domain methods for the fusion of images. Based on iterative algorithms and the optimization of cost functions these methods give relevant and very acceptable results. The introduction of the notion of neighbourhood and of automatically calculated learning rates enabled us to increase the performances and highlight new versions of the algorithms.

Even if it is very difficult to really choose one method to be the best we can say that all our methods are complementary. For multi focus images one will rather choose the DMF_WN or the RobustKMF while for multi-sensors images the RobustDMF or the RobustKMF are more adapted.

The test of our methods shows us that our algorithms can stand comparison with the common used methods. The results we get are as good not to say sometimes better than with transform domain image fusion methods.

References

[1] Yang, J., Blum, R.S., A Statistical Signal Processing Approach to Image Fusion using Hidden Markov Models for the book *Multi-Sensor Image Fusion and Its Applications*, Marcel Dekker/CRC, 2005. Editors: R. Blum and Z. Liu.

***

[2] Selesnick I.W., Baraniuk, R.G., Kingsbury N.G., The Dual-Tree Complex Wavelet Transform, *IEEE Signal Processing Magazine*, pp.123-151, November 2005

[3] Mitianoudis, N., Stathaki, T., Pixel-based and Region-based Image Fusion schemes using ICA bases, to appear in *Elsevier Information Fusion Journal*, 2005

[4] Li, Q., Stathaki, T., Image fusion using dispersion minimisation, *Proceeding of IEEE International Conference on Acoustic, Sound and Signal Processing*, Toulouse, France, 2006

[5] Li, Q., Stathaki, T., Local image fusion using Kurtosis minimisation with optimal learning rates, submitted to *Elsevier Information Fusion Journal*, 2006

***

[6] Wang, Z. and Bovik, A.C., A universal image quality index, *IEEE Signal Processing Letters*, vol. 9, no. 3, pp. 81-84, 2002

[7] Wald, L., Ranchin, T., Mangolini, M., Fusion of Satellite images of different spatial resolution : Assessing the quality of resulting images, *Photogrammetric Engineering and Remote Sensing*, vol. 63, no. 6, pp. 691-699, 1997

[8] Xydeas, C.S., Petrovic, V., Objective image fusion performance measure, *Electronics Letters*, vol. 36, no. 4, pp. 308-309, 2000

[9] Piella G., Heijmans, H., A new quality metric for image fusion, *Proceedings of the IEEE International conference on Image Processings*, vol. 3-4, pp. 173-176, 2003.

[10] Mitianoudis, N., Stathaki, T., Joint Fusion and Blind Restoration for Multiple Image Scenarios with Missing Data, to appear in *The Computer Journal*, 2007

[11] John, S., Vorontsov, M. Multi-frame selective information fusion from robust error estimation theory, *IEEE Transactions on Image Processing*, vol. 14, pp. 577-584, 2005

Annexes

A. Proof of lemmas

A.1. Lemma 1

Lemma 1

$$\forall k > 0 \quad \frac{\partial \tilde{y}^{k}(n)}{\partial \underline{\mathbf{w}(n)}} = k\,\tilde{y}^{k-1}(n)\underline{\tilde{\mathbf{x}}(n)}$$

Proof :

Let show it by induction on k.

- *Initialisation*: $k=1$

$$y(n) = \underline{\mathbf{w}(n)}^{\mathrm{T}}\,\underline{\tilde{\mathbf{x}}(n)}$$

so obviously $\dfrac{\partial \tilde{y}(n)}{\partial \underline{\mathbf{w}(n)}} = \underline{\tilde{\mathbf{x}}(n)}$.

The result is therefore true for $k=1$.

- *Iteration* : Let suppose it is true for k, let show that it is true for $k+1$

$$\tilde{y}^{k+1}(n) = \tilde{y}^{k}(n)\,\tilde{y}(n)$$

$$\frac{\partial \tilde{y}^{k+1}(n)}{\partial \underline{\mathbf{w}(n)}} = \frac{\partial \tilde{y}^{k}(n)}{\partial \underline{\mathbf{w}(n)}}\,\tilde{y}(n) + \tilde{y}^{k}(n)\frac{\partial \tilde{y}(n)}{\partial \underline{\mathbf{w}(n)}}$$

$$\frac{\partial \tilde{y}^{k+1}(n)}{\partial \underline{\mathbf{w}(n)}} = k\,\tilde{y}^{k-1}(n)\,\underline{\tilde{\mathbf{x}}(n)}\,\tilde{y}(n) + \tilde{y}^{k}(n)\,\underline{\tilde{\mathbf{x}}(n)}$$

$$\frac{\partial \tilde{y}^{k+1}(n)}{\partial \underline{\mathbf{w}(n)}} = (k+1)\tilde{y}^{k}(n)\,\underline{\tilde{\mathbf{x}}(n)}$$

The result is true for $k+1$.

Therefore the result is true $\forall k > 0$

A.2. <u>Lemma 2</u>

<u>Lemma 2</u>

$$\forall k > 0 \quad \frac{\partial \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^{k}\right\}}{\partial \underline{\mathbf{w}(n)}} = \frac{k}{NM}\tilde{y}^{k-1}(n)\,\underline{\tilde{\mathbf{x}}(n)}$$

<u>Proof :</u>

$$\mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^{k}\right\} = \frac{1}{NM}\sum_{i=1}^{NM}\tilde{y}^{k}(i)$$

$$\frac{\partial \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^{k}\right\}}{\partial \underline{\mathbf{w}(n)}} = \frac{1}{NM}\sum_{i=1}^{NM}\frac{\partial \tilde{y}^{k}(i)}{\partial \underline{\mathbf{w}(n)}} = \frac{1}{NM}\frac{\partial \tilde{y}^{k}(n)}{\partial \underline{\mathbf{w}(n)}}$$

$$\frac{\partial \mathrm{E}\left\{\underline{\tilde{\mathbf{y}}}^{k}\right\}}{\partial \underline{\mathbf{w}(n)}} = \frac{k}{NM}\tilde{y}^{k-1}(n)\,\underline{\tilde{\mathbf{x}}(n)} \text{ (Lemma 1)}$$

B. Matlab Code

B.1. Code for the DMF methods

```
function [If,w1f,w2f,Df,J,Q,E]=DMF(I1,I2,robust,nbit,method,win);

clc

                    %%%%% Parameters %%%%%
%
%  Inputs :    I1 <- first degraded image
%              I2 <- second degraded image
%              robust <- 0=no 1=yes
%              nbit <-number of iterations
%              method <- 'noexpectation'=original   'expectation'=WN
%              win <-size of the neighbourhood (if wanted)
%
% Outputs :    If -> fused image
%              w1f -> final weights matrix for the image 1
%              w2f -> final weights matrix for the image 2
%              Df -> vector containing all the successive values of D
(distorsion)
%              J -> vector containing all the successive values of the
cost function
%              Q -> vector containing all the succesive values of the
image
%              quality index
%              E -> vector containing all the succesive values of the
mean
%              square error



% First we create the vector containing all the possible values for
the
% learning rates
num=[];
for i=1:8
    for k=1:9
        num=[num k*10^-i];
    end
end


switch lower(method)



   %%%%%% First method : The "normal" DMF (no WN)  %%%%%%

      case {'noexpectation'}

% Initialization
[N M]=size(I1);
global Ii
```

```
global f_mean


% We are going to work with zero mean pictures
m1=mean(mean(I1));
m2=mean(mean(I2));
f1=I1-m1;
f2=I2-m2;

% w is set as a matrix with 1/2
w1=1/2*ones(N,M);
w2=1/2*ones(N,M);

% D is set as the mean of the distorsion of the two input images
D1=distorsion(I1);
D2=distorsion(I2);
D=mean([D1 D2]);


% We compute the first iteration of the images
f_=w1.*f1+w2.*f2; % Zero mean
f_mean=w1.*I1+w2.*I2; % Non-zero mean


% We calculate the values of the different parameters we want to
evaluate
Df(1)=D;
J(1)=cost_function(f_,D);
E(1)=sqrt(sum(sum((Ii-f_mean).^2)));
Q(1)=petrovic(I1,I2,f_mean);


for k=1:nbit

    disp(['Iteration ' num2str(k) ' over ' num2str(nbit)])

%% First step : Updating the w vector and the fused image f_


    % We choose the value of the learning rate eta either with the
robust
    % method or with a fixed value
    if robust==1
        grad1=(f_.^2-D).*f_;
        for i=1:47
            w1i=w1-num(i)*grad1.*f1;
            w2i=w2-num(i)*grad1.*f2;
            w1_i=abs(w1i)./(abs(w1i)+abs(w2i));
            w2_i=abs(w2i)./(abs(w1i)+abs(w2i));
            f_meani=w1_i.*I1+w2_i.*I2;
            essaien(i)=cost_function(f_meani,D);
        end
        [ess,vrainum] = min(essaien);
        eta=num(vrainum);
    else
        eta=10^-6;
    end
```

```
    % We calculate the gradient and update the values of w and the
fused
    % image
    grad1=(f_.^2-D).*f_;
    w1=w1-eta*grad1.*f1;
    w2=w2-eta*grad1.*f2;
    w1_=abs(w1)./(abs(w1)+abs(w2));
    w2_=abs(w2)./(abs(w1)+abs(w2));
    f_mean=w1_.*I1+w2_.*I2;
    f_=f_mean-mean(mean(f_mean));



% Second step : Updating the D value


   % We choose the value of mu either with the robust method or with a
   % fixed value
   if robust==1
        grad2=(D-mean(mean(f_.^2)));
            for i=1:47
            essaien(i)=cost_function(f_,num(i));
            end
        [ess,vrainum] = min(essaien);
        mu=num(vrainum);
   else
        mu=10^-1;
   end

    % We calculate the gradient and update the D value
    grad2=(D-mean(mean(f_.^2)));
    D=abs(D-mu*grad2);


%% Third step : Evaluation of the relevant parameters

    l=k+1;
    Df(l)=D;
    J(l)=cost_function(f_,D);
    Q(l)=petrovic(I1,I2,f_mean);
    if not(isempty(Ii))
        Q(k)=image_quality(Ii,f_mean);
        E(l)=sqrt(sum(sum((Ii-f_mean).^2)));
    end
    k=k+1;

end

w1f=w1_;
w2f=w2_;
If=w1_.*I1+w2_.*I2;
```

%%%%%% Second method : The DMF_WN %%%%%%

```
    case {'expectation'}


% Initialization
[N M]=size(I1);
global Ii

% We are going to work with zero mean pictures
m1=mean(mean(I1));
m2=mean(mean(I2));
f1=I1-m1;
f2=I2-m2;

% w is set as a matrix with 1/2
w1=1/2*ones(N,M);
w2=1/2*ones(N,M);

% D is set as the mean of the distorsion of the two input images
D1=distorsion(I1);
D2=distorsion(I2);
D=mean([D1 D2]);

% We compute the first iteration of the images
f_mean=w1.*I1+w2.*I2; %Non-zero mean version
f_=w1.*f1+w2.*f2; %Zero mean version

Df(1)=D;
J(1)=cost_function(f_,D);


for k=1:nbit

    disp(['Iteration ' num2str(k) ' over ' num2str(nbit)])

%% First step : Updating the w vector and the fused image f_

    eta=10^-6;
    grad1=(f_.^2-D).*f_;
    w1=w1-eta*calculate_expectation(grad1.*f1,win);
    w2=w2-eta*calculate_expectation(grad1.*f2,win);
    w1_=abs(w1)./(abs(w1)+abs(w2));
    w2_=abs(w2)./(abs(w1)+abs(w2));
    f_mean=w1_.*I1+w2_.*I2;
    f_=w1_.*f1+w2_.*f2;

% Second step : Updating the D value


    mu=0.9
    grad2=(D-mean(mean(f_mean.^2)));
    D=abs(D-mu*grad2);


%% Third step : Evaluation of the relevant parameters
    l=k+1;
    Df(l)=D;
```

```
    J(l)=cost_function(f_mean,D);
    if not(isempty(Ii))
        Q(k)=image_quality(Ii,f_mean);
        E(k)=sqrt(sum(sum((Ii-f_mean).^2)));
    end
    k=k+1;

end

w1f=w1_;
w2f=w2_;
If=w1_.*I1+w2_.*I2;


end
```

B.2. Code for the KMF methods

```
function [If,w1f,w2f,K,Q,E]=KMF(I1,I2,robust,nbit);

clc


%% Parameters
%
%  Inputs :   I1 <- first degraded image
%             I2 <- second degraded image
%             robust <- 0=no 1=yes
%             nbit <-number of iterations
%
% Outputs :   If -> fused image
%             w1f -> final weights matrix for the image 1
%             w2f -> final weights matrix for the image 2
%             K -> vector containing all the successive values of the
%             kurtosis
%             Q -> vector containing all the succesive values of the
image
%             quality index
%             E -> vector containing all the succesive values of the
mean
%             square error



% First we create the vector containing all the possible values for
the
% learning rates
num=[];
for i=-5:6
    for k=1:2:9
        num=[num k*10^-i];
    end
end


% Initialization
[N M]=size(I1);
```

```
global Ii
global f_mean

% We are going to work with zero mean pictures
m1=mean(mean(I1));
m2=mean(mean(I2));
f1=I1-m1;
f2=I2-m2;

% w is set as a matrix with 1/2
w1=1/2*ones(N,M);
w2=1/2*ones(N,M);

% We compute the first iteration of the images
f_mean=w1.*I1+w2.*I2;

% We calculate the values of the different parameters we want to
evaluate
K(1)=abs(kurtosis(f_mean));
E(1)=sqrt(sum(sum((Ii-f_mean).^2)));
Q(1)=petrovic(I1,I2,f_mean);



for k=1:nbit

    disp(['Iteration ' num2str(k) ' over ' num2str(nbit)])

%% First step : Updating the w vector and the fused image f_


% We calculate the optimal value of the learning rate
    if robust==1
        f_=f_mean-mean(mean(f_mean));
        m2=mean(mean((f_).^2));
        m4=mean(mean((f_).^4));
        grad1=sign(kurtosis(f_mean))/m2^3.*(m2*f_.^2-m4).*f_;
        for i=1:60
            w1i=w1-num(i)*grad1.*f1;
            w2i=w2-num(i)*grad1.*f2;
            w1_i=abs(w1i)./(abs(w1i)+abs(w2i));
            w2_i=abs(w2i)./(abs(w1i)+abs(w2i));
            f_meani=w1_i.*I1+w2_i.*I2;
            essaien(i)=abs(kurtosis(f_meani));
        end
        [ess,vrainum] = max(essaien);
        eta=num(vrainum);
    else
        eta=0.9;
    end


    f_=f_mean-mean(mean(f_mean));
    m2=mean(mean((f_).^2));
    m4=mean(mean((f_).^4));

    % We calculate the gradient and update the values of w and the
fused image
    grad1=sign(kurtosis(f_mean))/m2^3.*(m2*f_.^2-m4).*f_;
```

```
    w1=w1-eta*grad1.*f1;
    w2=w2-eta*grad1.*f2;
    w1_=abs(w1)./(abs(w1)+abs(w2));
    w2_=abs(w2)./(abs(w1)+abs(w2));
    f_mean=w1_.*I1+w2_.*I2;



%% Second step : Evaluation of the relevant parameters

    l=k+1;
    K(l)=abs(kurtosis(f_mean));
    Q(l)=petrovic(I1,I2,f_mean);
    if not(isempty(Ii))
    E(l)=sqrt(sum(sum((Ii-f_mean).^2)));
    end
    k=k+1;

end


w1f=w1_;
w2f=w2_;
If=w1_.*I1+w2_.*I2;
```

B.3. Code for the function used to add blur

```
function [blurred_image]=add_blur(image,num,level)

[n,m]=size(image);

%% Computes the degradation
degradation=fspecial('average',[2*level 2*level]);

%%Computes the degraded picture;
blurred_image=image;

if num==1
blurred_image(10:floor(n/3),30:floor(4*m/5))=imfilter(image(10:floor(n
/3),30:floor(4*m/5)),degradation,'replicate');
end

if num==2
blurred_image(floor(n/2):floor(7*n/8),floor(m/3):floor(9*m/10))=imfilt
er(image(floor(n/2):floor(7*n/8),floor(m/3):floor(9*m/10)),degradation
,'replicate');
end
```

B.4. Code for the function used to calculate the neighbourhood

```
function [meanimage]=calculate_expectation(image,win)

%% We first work on the center of the image
[N,M]=size(image);
image_=reshape(mean(im2col(image,[win win],'sliding')),N-win+1,M-
win+1);

win=floor(win/2);
meanimage=zeros(N,M);

%% Then on the corners
for i=1:win
    for j=1:M
        A(i,j)=mean(mean(image(max(i-win,1):min(i+win,N),max(j-
win,1):min(j+win,M))));
    end
end

for i=1:N
    for j=1:win
        B(i,j)=mean(mean(image(max(i-win,1):min(i+win,N),max(j-
win,1):min(j+win,M))));
    end
end

for i=N-win+1:N
    for j=1:M
        C(i,j)=mean(mean(image(max(i-win,1):min(i+win,N),max(j-
win,1):min(j+win,M))));
    end
end

for i=1:N
    for j=M-win+1:M
        D(i,j)=mean(mean(image(max(i-win,1):min(i+win,N),max(j-
win,1):min(j+win,M))));
    end
end

meanimage(1:win,:)=A(1:win,:);
meanimage(:,1:win)=B(:,1:win);
meanimage(N-win+1:N,:)=C(N-win+1:N,:);
meanimage(:,M-win+1:M)=D(:,M-win+1:M);
meanimage(win+1:N-win,win+1:M-win)=image_;
```